Exploiting Redundancy for Robust Sensing

Suman Nath

CMU-CS-05-166
August 2005

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:

Srinivasan Seshan, Chair
Phillip B. Gibbons
M. Satyanarayanan
Deborah Estrin, University of California, Los Angeles

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy

This research was sponsored by the National Science Foundation under grant no. ANI-0092678, the State of Pennsylvania under award no. 6003221 and through a generous grant from the Intel Corporation. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Abstract

In this thesis, we explore the challenges in making an Internet-scale heterogeneous sensing system more robust. We target "end-to-end" robustness in that we address failures in collecting data from a large collection of wired and wireless sensors, and problems in making sensor readings available to end-users from storage on Internet-connected nodes. Although often overlooked, robustness is extremely crucial for such systems because they are often deployed in harsh environments and are not typically very well-maintained. Traditional robustness techniques generally involve tradeoffs between robustness and resource-efficiency; i.e., they mask failures by using additional resources (e.g., energy, storage). Unfortunately, these traditional tradeoffs are not well suited to resource constraints and large scales of typical sensing systems.

This dissertation puts forth the claim that more practical solutions can be developed by exploiting several unique deployment- and application-specific properties of typical sensing systems. We show that by slightly relaxing the requirements of exact or fresh answers, we can significantly improve the robustness of a system, without additional resource overheads. We argue that this approach is well suited to sensing systems since optimizing resource usage is one of the important goals of their designs and the applications can often tolerate approximate or slightly stale data. We support the above claim by proposing efficient solutions for robust data collection and storage in a sensing system.

For robust collection of data from wireless sensors, we present *Synopsis Diffusion*, a novel data aggregation scheme that exploits wireless sensors' broadcast communication and sensing applications' tolerance for approximate aggregate answers. Synopsis Diffusion, unlike previous schemes, decouples aggregation algorithms from underlying aggregation topologies, enabling highly robust aggregation with energy-efficient multi-path routing. We also present *Tributary-Delta*, a novel adaptive aggregation scheme that efficiently combines the benefits of existing schemes and uses application-aware adaptation to cope with the dynamics of deployment environments. Under typical loss rates, our techniques can provide five times more accurate results than existing energy-efficient schemes, without additional energy overhead.

For storing sensor readings on Internet-connected nodes, we show that existing design principles used to build highly available storage systems do not work well for an Internet-scale system where failures are often correlated. Our results show that, for sensing applications, weak quorum systems are more suitable than traditional strict quorum systems because weak quorum systems are more effective in tolerating correlated failures and sensing applications can tolerate the small data inconsistency caused by such quorum systems. We also show that configuring a system with parameters derived by using the correlation model we develop is more effective than existing techniques in optimizing resource usage and target availability. Finally, we show how several data- and query-characteristics of a typical sensing system can be exploited to design efficient self-repairing and load balancing techniques. Our techniques can improve the availability of a sensing system by orders of magnitude without any additional resource overhead.

We show the feasibility of our techniques through a combination of analysis, simulation, and implementation within IrisNet, an Internet-scale sensing infrastructure that we have developed as part of this dissertation.

To my parents,
Mukti Nath and Manik Lal Nath

Acknowledgements

Over the last five years of my graduate education at CMU, a number of people influenced me and helped me complete this dissertation. I hope that I can remember everyone who helped me through this difficult yet rewarding process.

First and foremost, I would like to thank my advisor, Professor Srinivasan Seshan, for taking me on as a student about four years ago, even though I had little experience in systems research. He has been an ideal advisor in every respect, in terms of technical advice on my research and in terms of professional advice. My choice of career path has been greatly influenced by Srini. He encouraged me to work on sensor networks and introduced me to the researchers at Intel Research Pittsburgh that later became an extraordinary piece of good fortune for me. He showed a constant optimism in the ultimate usefulness and success of my research that helped to sustain me as I worked through the hard problems.

I am indebted to Dr. Phillip B. Gibbons and Dr. Haifeng Yu from Intel Research Pittsburgh for their endless hours of advice and guidance regarding my research. There is no doubt that this dissertation would have been impossible without their help. It was my work with Phil in the IrisNet project and on Synopsis Diffusion that initiated the work in this dissertation. Haifeng guided my research on robustness of distributed storage systems that constitutes a significant part of this dissertation. Phil's tireless editorial effort vastly improved the quality of this dissertation; wherever there was a sentence too meandering or grammatically incorrect, he had been there to edit it. Phil and Haifeng were the best mentors I could have asked for, and I hope that I can live up to their high standards.

I thank the other members of my thesis committee, Professor M. Satyanarayanan and Professor Deborah Estrin, for helping me select interesting research direction to explore. I am grateful to Satya also because of his enthusiastic support for the IrisNet project while he was the director of Intel Research Pittsburgh. Intel Research Pittsburgh generously helped me by providing the resources required for my research. I am also grateful to Professor Bruce Maggs for kindly taking me as his student in my first year at CMU and then giving me the freedom to choose the topics and the collaborators of my research.

I have built upon the work of the many members of the IrisNet projcet. Shimin Chen, Amol Deshpande, and Yan Ke designed and implemented significant parts of IrisNet that I could play with in my research. The other members of the IrisNet project, including Brad Karp, Padmanabhan Pillai, Rahul Sukthankar, and Dilip Sundarraj, had important inputs on my research on IrisNet. People from the Argus project, including Mark Abbott, Ganesh Gopalan, Rob Holman, Chuck Sears, John Stanley, and Curt Vandetta, helped us evaluating the IrisNet system with a real application. They were very tolerant users of IrisNet and I hope that this software continues to serve them well.

I am grateful to Zachary Anderson and Amit Manjhi for their valuable help in my work on Synopsis Diffusion and Tributary-Delta. Zach helped me in evaluating many ideas regarding Synopsis Diffusion. The idea of Tributary-Delta took shape during my discussion with Amit (and Phil).

I owe a special debt of gratitude to my mother for encouraging and supporting me to go to graduate school and my father for inculcating the love for knowledge in me. My friends from BUET, Anjan Bhowmick, Liton Chakraborty, Shubhashish Ghosh, Ashikur Rahman, Amalendu Roy, have provided a support network that has withstood the test of time. I would also like to thank many new friends I have made here at CMU, including, but not limited to: Mukesh Agrawal, Aditya Akella, Ashwin Bharambe, Yan Ke, Amit Manjhi, Mahim Misra, Muralidhar Talupur, and Hong Yan.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Preview

Recent technological improvements have enabled high-fidelity instrumentation of physical space with large collections of cheap sensing devices (called *sensors*) that are equipped with processing and communication capabilities. Examples of such devices include resource-constrained wireless sensors like motes [HSW⁺00] and high bit-rate multimedia sensors like cameras and microphones. These sensors are able to provide dense sensing close to physical phenomena, process and communicate sensed information, and coordinate actions with each other. Such capabilities have opened the door to building useful software systems, as evidenced by motes being used for habitat monitoring [BRY⁺04, HTB⁺05, SMP⁺04], structure monitoring [XRC⁺04], health [SGW01], education [SMP01], etc., and cameras and microphones being used for ocean monitoring [HSOH03], surveillance [CLFK01], tracking mobile objects [KT02], etc.

Motivated by the increased availability of sensors and the accelerated trend toward ubiquitous Internet connectivity, we envision *Internet-scale* applications that can manipulate information derived from a large collection of *heterogeneous* sensors spread across a large geographic area, even all over the world. An example of such an *Internet-scale, heterogeneous sensing application* (or, *sensing application* in short) is a hypothetical Ocean Monitor that detects, stores, and processes interesting oceanographic events (e.g., rip-tides or sand-bar formations) on coasts all around the world. It uses a large collection of cameras and motes deployed along the coastlines; cameras are used for looking over the near-shore ocean surface to detect oceanographic events, while motes are used to sense temperature or chemical properties of near-shore water. Oceanographers from anywhere, via the Internet, can query, analyze, and correlate live (directly from the sensors) and historical (from an archival system) oceanographic events worldwide, and combine them with other data sources. For example, an oceanographer from the USA can correlate the rip-tide events detected over the last year on the Australian coast with the corresponding chemical properties of water to understand whether the properties of riptides are affected by the water pH level. Such unique opportunities would greatly assist scientists in better understanding the behavior of oceans. Additional examples of useful sensing applications include a Parking Space Finder, for directing drivers to available parking spots near their destination; a Bus Alert service, for notifying a user when to head to the bus stop; Waiting Time Monitors, for reporting on the queuing delays at post offices, food courts, etc.; a Lost and Found service, for tracking down lost objects; and a Person Finder service, for locating your colleagues or monitoring your children playing in the neighborhood.

1.1 Focus of This Thesis: Robust Sensing

An important, but often overlooked, requirement of such a sensing application is its robustness, i.e., its ability to mask failures of system components. The importance comes from several factors. First, wireless sensors are often deployed in a harsh, or even hostile, environment where component failures can be a norm. Over time, nodes can fail; they may run out of energy, overheat in the sun, be carried away by wind, crash due to software bugs, or be stepped on by a wild elephant. Even in fixed positions, nodes' communication ranges (and thus their topologies) can change dramatically due to the vagaries of RF propagation that result from its strong environmental dependence [PG03]. These changes are difficult to predict in advance. Similarly, sensors directly connected to the Internet may fail or become unavailable because of network failures, worms or DDoS attacks, software crashes, or maintenance downtime.

Second, sensors are often deployed in remote places and are operated mostly unattended. Fixing failures, if at all feasible, may take significant time. Traditional large-scale networks such as the Internet work in spite of brittle hardware and software partly because the number of people maintaining a network has grown along with the size of the network itself. In contrast, a single human will often be responsible for a large collection of sensors, and hence a sensor may not receive individual human attention. Therefore, it is desirable that a sensing application automatically mask failures.

Finally, and interestingly, some sensing applications become more important during catastrophic events that cause large failures. For example, an ocean storm may bring a significant part of the Ocean Monitor system down; but the same storm may generate oceanographic events that scientists are interested in studying. Similarly, a sensing application deployed inside a building may become more valuable during a fire since firefighters may use the application to locate people inside the building and rescue them. Therefore, the applications must survive such catastrophes.

In this thesis, we focus on providing end-to-end robustness of a sensing application. To understand different components of this end-to-end robustness, we next briefly describe the system model we consider and the software infrastructure we develop as part of this thesis.

1.2 Internet-scale, Heterogeneous Sensing System

We envision an Internet-scale sensing system that collects data from a large collection of sensors dispersed over geographic regions and stores the data so that the stored data, along with live sensor data, can be accessed and processed from anywhere in the Internet. Sensing applications are deployed over this sensing system and provide useful services to end users.

1.2.1 Sensing System Architecture

We here briefly describe a feasible design for an Internet-scale sensing system and the rationale behind such design. We also provide an overview of IrisNet, an implementation of this design.

At a high level, an Internet-scale sensing system consists of two components: the first is responsible for data collection and and the second for data storage.

Data Collection

The data collection component of a sensing system collects and processes data collected from different types of sensors, connected directly or indirectly to the Internet, such as the following:

- **Resource-constrained sensors:** These are small, low-cost, battery-powered devices each having limited processing speed and storage capability, short-range wireless radio, and sensing components capable of detecting local conditions such as temperature, light, sound, or movement of objects. An example of such a sensor is the Berkeley mote [HSW$^+$00]. Although a single mote has limited capability, when deployed in a large number, they have the ability to measure a given physical space in great detail. They generally collect information over a small geographic location, organize themselves into a wireless *ad hoc* network, and send the collected information to a special resource-rich node called the base station, which is often connected to the Internet.

- **Resource-rich sensors:** These are high bit-rate sensors such as cameras and microphones attached to devices (e.g., laptops) with PC-class processing and storage capabilities, sufficient energy, and good connectivity to the Internet. Such sensors often provide multimedia data and monitor the physical space in a "non-immersive" way.

- **Software sensors:** These are computer programs that, unlike the two previous types of sensors, provide useful information about computer systems. Examples of such sensors include programs reporting current available bandwidth or disk space of a set of computers. Like resource-rich sensors, they are not constrained by processing, storage, connectivity, or energy.

We refer to the first type of sensors as *wireless sensors* and the latter two types as *wired sensors*.[1]

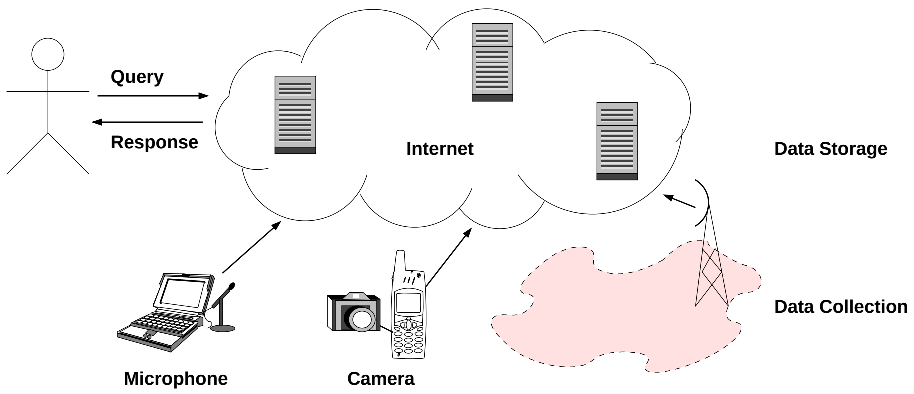
For scalability, it is desirable that sensor data is processed and filtered near its sources. The data collection component of a sensing system provides the general computation environment for such aggressive filtering. It uses the computation capabilities of the sensors to process raw data and transfers only the useful information over the network.

Data Storage

A sensing system indexes and stores the data collected or derived from the sensors in Internet-connected nodes (Figure 1.1). Storing data in such resource-rich nodes is important for several reasons. First, the nodes can efficiently index and archive sensor data and let users query them interactively from anywhere in the world; sensors themselves may not have enough storage to archive historical data required by many (e.g., scientific) applications. Second, these nodes can perform complex processing, such as cleaning, modelling, and correlation of data; such processing is not feasible in wireless sensors. Third, such storage nodes can act as a *cache* for sensor data; one application can reuse the data collected for others, thus improving system efficiency.

The right architecture of this storage component is decided by the following data and query characteristics of a sensing application: i) numerous high bit-rate live data sources, and ii) rich, expressive queries involving data that spans a large geographic region. The first property implies that a centralized or clustered system like Google is not suitable, while the second property suggests that a distributed file system such as NFS is not sufficient for this purpose. A more suitable architecture is a read-write

[1]Note that resource-rich sensors can use point-to-point wireless communication; we use the term *wired sensors* just for the classification purpose.



This diagram shows the architecture of an Internet-scale heterogenous sensing system we consider in this thesis. At a high level, it consists of a data collection component that collects data from a large collection of wired and wireless sensors, and a data storage component that stores and organizes that data in a way such that end-users can process and query it from anywhere in the Internet.

Figure 1.1: An Internet-scale, heterogeneous sensing system.

distributed database that is capable of frequent updates and efficient processing of queries on distributed sensor data. Moreover, users may care about only a small fraction of the total data produced by the sensors; storing them near the data sources and querying them only on demand saves valuable network bandwidth. This also highlights the importance of a distributed query processing mechanism for the storage component.

Sensing applications, running on the sensing system, let users from anywhere in the Internet query and perform complex processing on stored sensor data. The sensing system also provides generic features like load balancing, fault-tolerance, security, etc. required by a sensing application. An architecture for a generic sensing system is shown in Figure 1.1.

1.2.2 IrisNet: A Shared Infrastructure for Sensing Applications

In the IrisNet (**I**nternet-scale **R**esource-**i**ntensive **S**ensor **Net**work Services) project [Int03b, GKK+03], we use the above design rationale to build a sensing system. Moreover, our goal is to make IrisNet an easy-to-use software infrastructure that someone can use to easily write new sensing applications. IrisNet is the first general-purpose shared software infrastructure tailored for developing and deploying new sensing applications. To simplify the task of developing new sensing applications, we take two complimentary steps. First, we build the first general-purpose shared software infrastructure tailored for developing and deploying new sensing applications. IrisNet provides generic functionalities, e.g., data collection, query processing, robustness, etc., required by most sensing applications. Second, we provide a simple programming model that lets application developers express application-specific tasks and the intended use of IrisNet functionalities. Thus, IrisNet works as a customizable building block for developing new sensing applications.

The IrisNet architecture comprises two tiers. The lower data collection tier, consisting of a large collection of *Sensing Agents* (SAs), collects and processes data from sensors. SAs run on laptop- or PDA-class machines. Wired sensors are directly connected to SAs that process high bit-rate sensor data to extract useful information for applications. Wireless sensors run in self-organizing clusters and connect to SAs (running on base stations) over multi-hop wireless networks. SAs on base stations install appropriate programs on the wireless sensors [LGC05, LMG+04], monitor them [ZGE03], and collect useful information sent by sensors [MFHH02]. Data collected by SAs is sent to the upper data storage tier of IrisNet consisting of a collection of *Organizing Agents* (OAs) running on Internet-connected PC-class machines. OAs index and store sensor data such that users can easily query it. OAs also provide mechanisms for load balancing, fault tolerance, security, etc.

Building a sensing system requires addressing a large number of challenges related to data collection, data storage, query processing, robustness, load balancing, security, privacy, etc. We address the robustness aspect in this thesis. We briefly describe the other challenges in the context of IrisNet in Chapter 3.

1.3 End-to-end Robustness of a Sensing Application

To ensure end-to-end robustness of a sensing application, both the data collection (SAs in IrisNet) and the data storage (OAs in IrisNet) components must be able to tolerate failures. For example, a live mote should be able to send its data to the base station despite other motes' failures or poor connectivity; a camera disconnected from the rest of the network should be able to locally log its data and send it to the storage nodes later; disconnected or crashed storage nodes should not impact the availability of stored data, etc. Failure of either the data collection or the storage component would make users unable to access live sensor data, while failure of the storage component would make them unable to access archival data.

In this thesis, we focus on end-to-end robustness of a sensing system. However, for the data collection component, we mostly focus on *aggregate* data collection from wireless sensors. Aggregate data provides holistic information about the whole or a part of a sensor network (e.g., *average* temperature sensed by the sensors, *frequently occurring* events in the system, etc.). Because aggregate data can often be collected in an energy-efficient way, it assumes greater importance than individual sensor readings that are too detailed for many applications. Robustness of the data collected from wired sensors can be ensured by the storage component: we assume that these sensors have good network connectivity to the storage nodes and they can handle transient disconnection by locally logging the sensor data for later transmission to the storage nodes.

1.4 Preview: Two Fundamental Challenges of End-to-end Robustness

There are two independent factors that affect the end-to-end robustness of a sensing application: the fragility of data collection from wireless sensors and the vulnerability of large-scale data storage on Internet-connected nodes. The main stumbling block in combating the first challenge is wireless sensors' poor connectivity and resource constraints. The second challenge stems from large-scale correlated failures that are common in today's Internet. Although such failures have adverse effects on the availability of stored sensor data, their precise impact and nature is poorly understood.

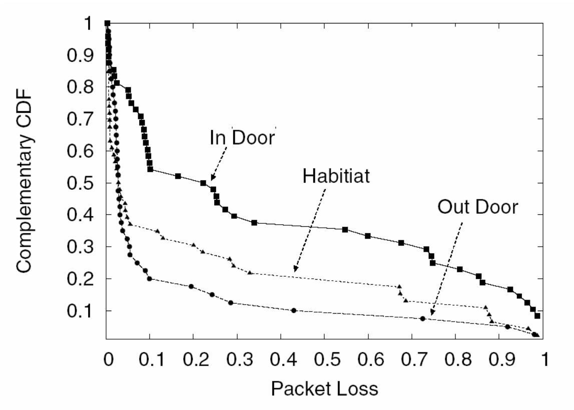
In this thesis, we address these two challenges, using a combination of novel algorithms, techniques, measurements, and analysis. The rest of this section summarizes the challenges and outlines our solutions to them.

1.4.1 Challenge#1: Robust Data Collection

Robust collection of aggregate data from wireless sensors is challenging due to the following factors:

1. *High transient loss rates:* Due to their harsh deployment environment and low-power radio hardware, sensors experience high communication loss rates. For example, studies of real deployments have shown that 50% of nodes experience greater than 10% and 30% of nodes experience greater than 30% loss rates (Figure 1.2), a significantly higher loss rate than that experienced by other wireless networks [ZG03].

2. *Long-term dynamics:* The operating condition of a long-running wireless sensor network may change over time. For example, message loss rates between nodes can change due to environmental dynamics (Figure 1.3), existing nodes can leave the



This graph, taken from [ZG03], shows the average transient loss rates experienced by the sensors in a number of real deployments. The legend `In Door` *represents a deployment inside a building,* `Out Door` *represents an outdoor deployment, and* `Habitat` *represents a real habitat monitoring deployment.*

Figure 1.2: Transient loss rates experienced by sensors in real deployments.

network due to hardware failure or resource depletion, and new nodes may join it due to new deployments. We call these dynamics "long term" as they last for hours to days.

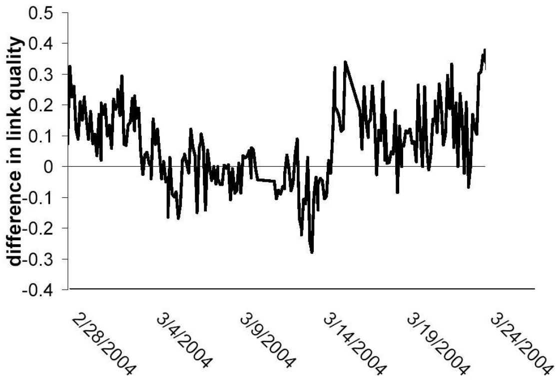
3. *Energy-constraints:* Sensors are generally powered by batteries that are not feasibly replaced on depletion. Therefore, they should avoid energy-expensive operations to minimize energy consumption and to maximize life time.

We now describe the impact of transient failures and long-term dynamics on data aggregation from wireless sensor networks.

Transient Failures

Existing aggregation schemes have the following properties:

1. *Aggregation over a tree topology:* Common approaches for computing aggregate information in a wireless sensor network use a tree aggregation topology and combine partial results at the intermediate nodes during message routing (more details in Chapter 2). The latter technique, called *in-network aggregation*, is required for energy-efficiency. A tree topology is required to avoid double-counting—with other topologies, data from a node may be accounted for by multiple next hop nodes, resulting in an overestimation of the actual answer.



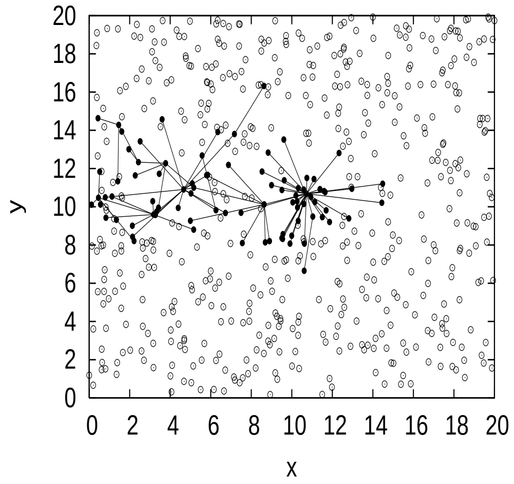
This graph, taken from [PG03], shows the difference of message loss rates (shown in the y-axis) from a node to two of its neighbors in a real sensor deployment at Intel Research Berkeley and over a month period of time (shown in the x-axis). Because the difference is both positive and negative, the best choice of the next-hop node can change over time.

Figure 1.3: Difference of message loss rates from a node to two of its neighbors in a real sensor deployment and over a month period of time.

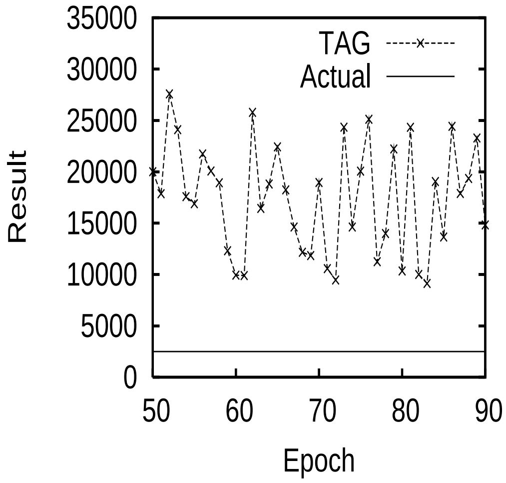
2. *Unreliable communication:* Because reliable communication is expensive, in order to save valuable energy, sensors tend to use unreliable communication that provides no guarantee of successful data delivery. Therefore, data sent by a sensor may be dropped silently before it reaches the base station.

However, a tree topology with unreliable communication is very susceptible to node and transmission failures. Because each of these failures loses an entire subtree of readings, a large fraction of the readings are typically unaccounted for in a tree-based system (Figure 1.4(a)). This introduces significant error in the query answer [CLKB04,MFHH02, ZGE03]. This is depicted in Figure 1.4(b), which shows that aggregation over a tree consistently overestimates the actual average value. Moreover, the high variance of the computed aggregate suggests that simply scaling the measured value up or down will not solve the problem.

A common approach to addressing this problem is to fix the second property above; i.e., to use a reliable communication protocol such that a sender can detect lost messages and retransmit them [SH03]. This can considerably improve the accuracy of aggregation over a tree. However, this improvement comes at the cost of high overheads. Namely, the acknowledgement messages required for reliable communication reduce channel utilization by 25%, retransmission increases the latency of aggregation, and most importantly, incurs significant energy overhead (under a $\approx 30\%$ loss rate, latency and energy consumption increase by around 3 times). This in turn reduces the lifetime of battery-powered sensors.



(a) Nodes counted over a tree



(b) Computing Average over a tree

In this experiment, 600 sensors are randomly placed in a square area, with the base station at the center of the area. Sensors use unreliable communication and we assume a realistic loss model (details in Chapter 4). Each sensor has a value inversely proportional to the square of its distance from the querying node at the center, emulating the intensity readings of a radiation source at the center. We run continuous query, with one Average value computed at the base station in every epoch (an epoch is a round of communication, explained in Chapter 4). Figure (a) shows the activity of the sensors during a typical epoch. The lines show the path taken by sensors readings that reached the querying node at the center. Readings from the sensors not connected by lines fail to reach the base station, for link-failures in upstream nodes, and therefore are not accounted for in the final answer. Figure (b) shows the Average value computed with TAG, a tree-based aggregation scheme, by the base station in different epochs.

Figure 1.4: Fragility of tree-based aggregation.

Our approach to addressing this problem is to fix the first property above; i.e., to use robust multi-path topology, instead of a tree, for aggregation. We exploit the fact that wireless sensors use broadcast communication and therefore, when a node sends a message, more than one of its neighbors may be able to receive that message. By allowing a subset of the neighbors to include the message in in-network aggregation, we can perform aggregation over multi-path routing. Multi-path is more robust than a tree, because a message can reach the base station through multiple alternate paths and unless all the paths fail, the message is successfully delivered. Thus, multi-path routing makes the aggregation process robust, even with unreliable communication. Moreover, because a node needs to broadcast only once to communicate with all the neighbors, multi-path routing can be efficiently implemented in a wireless sensor network.

However, performing aggregation over a multi-path topology is not trivial. The fundamental stumbling block is that multi-path routing often results in message duplication that causes double-counting of a large fraction of sensor readings. For example, in computing the Sum aggregate, if an individual reading or a partial sum is sent along four paths (to improve the likelihood that at least one path succeeds), and three of them happen to succeed, that value or partial sum will contribute to the total sum three times instead of once. Moreover, values along multiple paths may be aggregated in different orders, which might be problematic for order-sensitive aggregates. A tree topology does not have such duplication and ordering problems and is therefore used by existing aggregation schemes. This also means that, in these schemes, the aggregation algorithm and the required routing topology are *tightly coupled*; it is therefore not possible to use arbitrarily robust routing, such as multi-path routing.

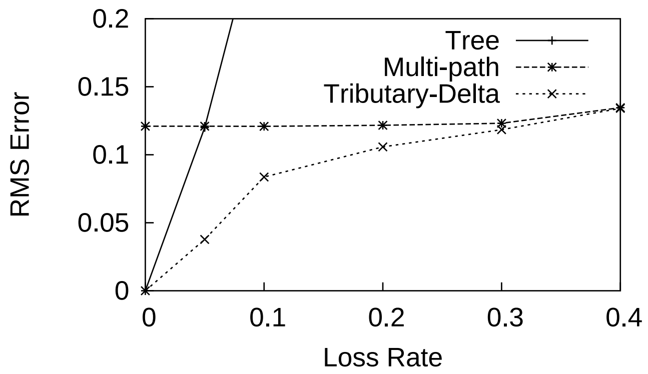
To address this, we exploit the fact that due to energy constraints and large deployment scales, most sensing applications require only approximate aggregate answers, and it is possible to compute approximate answers by methods that avoid the above problems. To this end, we propose *Synopsis Diffusion* [NGSA04], a framework well-suited to energy-efficient aggregation over arbitrary (multi-path) routing topologies. The basic idea is to map a target aggregate function to a set of order- and duplicate-insensitive (ODI) functions, and to use them over an energy-efficient multi-path aggregation topology. The ODI property of the functions ensures correctness of aggregation over multi-path topology which in turn provides robustness. As an example, under typical loss rates, a Synopsis Diffusion Sum algorithm provides a five times more accurate result than the existing tree-based algorithm, without additional energy overhead. We provide approximate algorithms to compute several aggregates in this framework and establish the formal foundation that would enable the development of more such algorithms.

Long-term Failures

To cope with relatively long-term dynamics of a sensor network, aggregation schemes must be self-configuring and must adapt to the changes in deployment environment. For example, Figure 1.3 demonstrates a situation where a node should change its next-hop routing neighbor over time. Enabling such adaptation in a resource-constrained environment is challenging.

The common approach to coping with dynamics is to adapt the aggregation topology based on low level observations about network characteristics, without taking application semantics into account. Each node, based on long-term loss statistics, chooses a neighboring node to be its next hop for relaying messages. As loss rates change over time, the next hop node is changed to the one currently having good connectivity [IGE+03, MFHH02]. A node can collect loss statistics by snooping neighboring nodes' broadcasts (which works only when links are symmetric) or by explicitly asking neighbors for hints about its outgoing link quality (e.g., through acknowledgements). The adaptation decision can be completely local [MFHH02], or it can initiated by the base station [IGE+03] depending on the application's requirement. Local adaptation is scalable and quick, but global adaptation can exploit application-level semantics to tune the frequency of adaptation and the associated overhead.

We argue that application-aware adaptation is a valuable mechanism, since applications can exploit the intended data semantics, which are generally weak ones, and can make certain adaptation decisions better. To enable this, we identify a new component to adapt: the aggregation algorithm. Existing tree-based and our multi-path-based aggregation schemes present a tradeoff between the robustness against communication failure and the accuracy of computed answers. For example, tree-based aggregation is fragile against communication failure, but for most aggregates, it provides exact answers in the absence of failures. In contrast, multi-path-based aggregation, although highly robust against communication failures, generally provides approximate answers. Therefore, the best algorithm depends on the current loss rate; e.g., under a high loss rate, a multi-path-based algorithm may be preferable since its approximation error is smaller than the communication error of a tree-based algorithm. In a network where loss rates can change over time or where different parts of the network may have different loss rates, a desirable strategy is to combine the benefits of these existing approaches into one hybrid approach and to dynamically adapt it to the current loss rates. Applications using the aggregate results can provide feedback about the quality of end results, so that the aggregate scheme knows when and how to adapt.



This graph shows that Tributary-Delta, although a hybrid of tree- and multi-path-based schemes, can perform better than any of these schemes under all loss rates. The experimental setup and the full graph are provided in Chapter 5.

Figure 1.5: RMS error of a Count query under varying message loss rates and different aggregation schemes.

We achieve this through a novel aggregation scheme called *Tributary-Delta* that concurrently runs tree-based and multi-path-based aggregation in different parts of the network. As operating conditions change, it uses a combination of local repair and application-aware global adaptation of the aggregation topology and the aggregation algorithm. We show that our Tributary-Delta approach significantly outperforms both tree and multi-path schemes. An example result is shown in Figure 1.5 (full details in Chapter 5). As expected, the tree-based approach is more accurate than the multi-path-based approach at very low loss rates, because of its lower approximation error (0% versus 12%). However, at loss rates above 5%, tree is much worse than multi-path because of its high communication error. On the other hand, Tributary-Delta provides not just the best of both (e.g., from running either tree or multi-path in the entire network), but in fact provides a significant error reduction over the best, across a wide range of loss rates, thus demonstrating the synergies of using both in tandem.

1.4.2 Challenge#2: Robust Data Storage

A sensing system stores the raw data collected from different types of sensors, or the information derived from them, in its distributed database on the Internet. The database may be deployed on nodes under a single, loose administration like today's Akamai [Aka05] or PlanetLab [PACR02,Pla05], or multiple cooperative administrations like today's ISPs. In such large systems, failures of system components (e.g., nodes or links) are the norms, rather than the exceptions. The database needs to be highly available and durable in the face of such failures. The need arises partly because many sensing applications often archive important data (e.g., scientists must archive all the data collected from their sensing experiments), and partly because, as a result of systems advances, performance is no longer the only/primary limiting factor in the utility of distributed systems—availability has become at least as important as performance.

Designing highly available distributed database (or, general storage) systems has long been an active area in systems research. Standard availability techniques include replication, regeneration (automatic repair on replica failures), and dynamic load balancing. Replication helps in tolerating transient crash-failures, regeneration helps in adapting to long-term crash-failures, and dynamic load balancing avoids failures due to overloads caused by flash-crowd like events. Although each of these components has been well studied in the context of many existing systems, we face the following two key problems in using them within an Internet-scale sensing system:

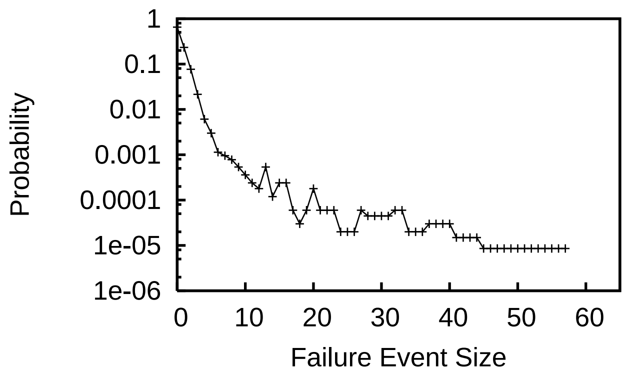
- Standard availability techniques used in existing distributed storage systems are less effective than one might hope in an Internet-scale system, often resulting in system designs that are far from optimal. We identify that correlated failure events, i.e., when a large number of nodes fail almost simultaneously, are the primary reason behind this. Addressing correlated failures in system design is difficult since the nature of failure correlation in the real world is poorly understood.

- Existing availability techniques have not yet been used in the context of Internet-scale sensing. Although these techniques, in general, are not very effective in tolerating correlated failures, sensing systems have many unique properties (due to their relaxed functional requirements, as described later) that, we believe, can significantly simplify the problems. However, such possibilities have not been explored yet; we need to develop suitable techniques that can exploit the properties and improve system availability.

We now discuss these two problems in detail.

Correlated Failures in Internet-scale Systems

Existing distributed databases, unlike the future sensing systems we envision, are not designed to scale to thousands of not-very-well-maintained nodes in the Internet. One might think that we could use previously proposed availability techniques in our target sensing system; however, our experience shows that, surprisingly, these techniques are less



A failure event *causes a number of machines to fail within a short period of time. The* failure event size *denotes the number of machines failed due to a particular failure event. The graph confirms that failures are often correlated and large events are not very rare.*

Figure 1.6: Failure correlation (from 1/2003 to 6/2004) in the PlanetLab testbed.

effective in the real world than one might hope, often resulting in system designs that are far from optimal. We identify that failure correlation is the primary reason behind this.

The traditional approach of designing and evaluating systems assumes that failures are independent [BTC+04, BDET00, Cat03, DLS+04, DW01, Yu04, YV04]. For example, most existing systems, in selecting replication parameters (e.g., number of replicas, quorum size) required to achieve a target availability, use mathematical analysis that assumes independent failures. However, for a large-scale sensing system deployed over the Internet, the assumption of failure independence will rarely be true. Node failures are typically correlated on the Internet, with multiple nodes in the system failing (nearly) simultaneously. The existence of correlated failures in the Internet is evidenced in Figure 1.6 that shows the distribution of simultaneously-failed nodes in PlanetLab over an 18-month period. The size of correlated failures can be quite large. For example, Akamai experienced large distributed denial-of-service (DDoS) attacks on its servers in May and June 2004 that resulted in many of its client sites being unavailable [Com04], and PlanetLab experienced four failure events during the first half of 2004 in which more than 35 nodes failed within a few minutes (these PlanetLab events are also included in Figure 1.6). Such large correlated failure events may have numerous causes, including system software bugs, DDoS attacks, virus/worm infections, node overload, and human errors.

The impact of failure correlation on availability of archived sensor data is dramatic. Previous studies have shown [BWWG02, YNY+04], and our work confirms, that unavailability under realistic (correlated) failure patterns is orders of magnitude worse than if failures were independent. Intuitively, if failures arrive independently, individual failures are spread across time, and hence a system gets enough time to repair its replicas between two consecutive failures. However, when failure arrivals are correlated, a large number of failures may cluster in time to destroy all the replicas before the system can react. Once all the replicas fail, the system cannot repair itself because the data to copy to new replicas is no longer available.

Poor understanding of the precise nature and impact of correlated failures is one of the biggest stumbling blocks in designing a highly available distributed sensor database. System designs today either assume independent failures or use simple failure models that may lead to a wrong design, failing to provide the target availability in practice. Since large-scale correlated failures are common in today's Internet, the design and evaluation of an Internet-scale sensing system must consider them. For example, in using replication, a sensing system must choose suitable replication parameters (e.g., number of replicas, quorum systems, etc.) required to achieve a target availability despite the correlated failures common in real-world. To this end, we try to systematically understand the following important questions: Is the impact of correlated failures on system availability trivial? Are the design principles traditionally being used in real distributed systems correct under correlated failures? What is the right approach to mitigating the negative effects of correlated failures? Can we model correlated failures in real world and use the model in choosing replication parameters? We study three real-world failure traces and use a combination of experimental and mathematical analysis to answer the above questions. We show that previously proposed approaches (details in Chapter 2), although plausible, are less effective than one might hope under real-world failure correlation, often resulting in system designs that are far from optimal. Our study also reveals the subtleties that cause this discrepancy between perception (the myth) and reality. These new findings lead to a number of design principles for tolerating correlated failures in distributed storage systems. They also provide valuable insights about how some unique properties of a typical sensing system can be exploited to improve its availability.

Exploiting Sensing Properties

Although existing availability techniques, such as replication, regeneration, and load balancing, fail to ensure high availability in the presence of correlated failures, we believe that sensing systems have many unique properties (e.g., hierarchically scoped queries, tolerance for occasional data inconsistency, absence of write-sharing, easily serializable timestamped writes, etc.) that can significantly simplify the problems. As an example, we here show how the weak data consistency requirement of sensing applications can be exploited to choose suitable quorum systems that are more effective that traditional ones in tolerating correlated failures.

Users in sensing applications often make snapshot queries involving data collected from multiple sensors at a certain time. However, sensors are often asynchronous or only loosely synchronized; e.g., some sensors may be slower than the others, or some sensors may be temporarily disconnected from the network. As a result, data from some sensors in the snapshot may be stale compared to that from others. Such an inconsistent snapshot is difficult to avoid in an asynchronous or loosely-synchronized interactive system. However, most sensing applications can tolerate such occasional inconsistency by using application specific knowledge (e.g., timestamps and temporal correlation of sensor data).

Our study reveals that this tolerance for occasional stale data can be exploited to mitigate the negative impact of correlated failures. More specifically, we show that a recently proposed quorum system (called Signed Quorum Systems (SQS) [Yu04]), which occasionally returns stale data, is more suitable for a sensing system than traditional quorum systems. SQS, for its smaller quorum size, can effectively tolerate most correlated failures that adversely affect traditional quorum systems. For example, SQS, with regeneration mechanisms and with parameters chosen based on analysis using our failure model, can reduce unavailability of a sensing system by an order of magnitude without any additional overhead.

Similarly, we show that sensing applications' hierarchically organized data and hierarchically scoped queries can be exploited to design an efficient load balancing algorithm that can cope with flash-crowds [NGS05]. We also show that absence of write-sharing and easily serializable timestamped writes in sensing applications enable a simple regeneration algorithm that can scale to a large number of nodes. We show the feasibility and usefulness of these techniques by implementing them in IrisNet.

1.4.3 Summary and the Thesis Statement

In summary, the challenges of robust sensing arise from the resource constraints and the highly failure-prone deployment environment of wireless sensors, and large-scale correlated failures of the Internet-connected storage nodes. Although the challenges are difficult to address in general, **sensing applications have unique domain-specific properties (e.g., broadcast communication medium, tolerance for approximate query answers and weak data consistency) that we can efficiently exploit to make the applications highly robust, without any significant resource over-**

heads.

1.5 Contribution and Structure of the Thesis

This thesis makes contributions in two major areas. The first area is conceptual and consists of the novel ideas and design principles generated by our work. The second area of contribution is a set of artifacts—a few major components of IrisNet and a number of sensing applications that we have implemented to validate some of our ideas.

1.5.1 Conceptual Contributions

In this thesis, we address the aforementioned challenges and propose solutions that result in significant improvement in end-to-end robustness of an Internet-scale, heterogeneous sensing system. Because there is a tremendous heterogeneity and diversity in the problems, there is no single panacea for all of them. We recognize this heterogeneity and develop a suite of algorithms, techniques, and design principles to combat these problems. The components of this solution suite include the following:

1. **Synopsis Diffusion:** *Synopsis Diffusion* is a general framework for computing aggregates over an arbitrary (multi-path) topology. It decouples aggregation algorithm and aggregation topology, providing the opportunity to optimize them independently. We provide the formal foundation of this framework, example algorithms, and evaluation results. Synopsis Diffusion is the first to show the feasibility of in-network aggregation over a multi-path topology.

2. **Adaptive aggregation:** We propose *Tributary-Delta*, an energy efficient scheme that combines the benefits of existing tree-based and our multi-path-based aggregation by simultaneously running them on different parts of the network and automatically balancing the proportion of these two components as operating conditions change. Tributary-Delta is the first to demonstrate application-aware adaptation of aggregation algorithms to cope with long-term dynamics of operating conditions. In developing Tributary-Delta, we also propose *Adaptive Rings*, a multi-path aggregation topology that can automatically adapt to link- or node-failures in an energy-efficient manner.

3. **Design principles to tolerate correlated failures:** Using a combination of experimental and mathematical analysis of several real-world failure traces, we debunk

a number of common myths about how to design highly available systems. Based on our analysis, we identify a set of design principles that can be used to build Internet-scale sensing systems capable of tolerating correlated failures. Although the design principles allow us to exploit unique properties of sensing systems, they can be generalized and used in a large class of distributed storage systems.

4. **Lessons from availability evaluation:** Aside from evaluating our own ideas, we have implemented and evaluated a number of existing techniques that were either not implemented (e.g., SQS) or not evaluated (e.g., the Paxos consensus algorithm [Lam98]) before in an Internet-scale system. These implementations demonstrate the behavior of these algorithms in practice; i.e., we show that a number of important optimizations are required for a large number of concurrent Paxos instances to converge on an Internet-scale system. Our experiments with IrisNet's storage component identify important differences between the methodologies for evaluating availability and performance. We show that, unlike performance, availability can be accurately evaluated with simple *simulation*, with a realistic *failure trace* collected over a reasonably *long period* of time.

The first two contributions aid in robust data collection from wireless sensor networks, while the last two contributions are in the area of reliable storage of sensor data in Internet-connected nodes. In combination, they provide the end-to-end robustness of a sensing system.

1.5.2 Artifacts

In the course of this thesis, we have developed the following artifacts in the context of IrisNet:

1. We have designed and implemented the basic architecture of IrisNet that demonstrates the viability of our ideas related to Internet-scale sensing. The source code of IrisNet is available at `http://www.intel-iris.net`.

2. We have developed a number of prototype applications on IrisNet that validate our ideas. One application, IrisLog [Int03a], has been publicly available since September 2003, letting users make live queries regarding resource usage of 450+ PlanetLab machines worldwide. IrisLog can be accessed at `http://www.intel-iris.net/irislog`. The source code of this application comes with the standard IrisNet distribution.

3. We have designed and implemented IrisNet's distributed storage component that can tolerate correlated crash failures of system components and provide high availability of archived sensor data. It uses SQS, which we have shown to be better in tolerating correlated failures. It automatically regenerates new replicas when existing ones fail. In choosing replication parameters, it explicitly considers failure correlation in real systems. We allow the system administrator to specify a target availability; we then use our empirically validated failure model to choose the replication parameters required to achieve the target availability. To avoid overload failures, we have developed and implemented an efficient load balancing mechanism that quickly spreads load across multiple nodes when a flash-crowd approaches. The source code comes with the standard IrisNet distribution.

4. We have developed a failure benchmark program that generates synthetic failure traces (i.e., information about when individual nodes in a system fail and recover) for certain classes of distributed systems. The benchmark uses realistic models capturing certain failure properties (e.g., failure correlation) that we have identified from our study. We believe that such a benchmark will be very useful in more accurately evaluating availability of distributed systems. The benchmark can be found at `http://www.intel-iris.net/benchmark`.

The first two artifacts are specific to Internet-scale sensing. However, the last two artifacts are more general and can be used by a large class of distributed systems.

1.6 Outline of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, we describe background materials in the area of sensing applications and robustness. In Chapter 3, we give a general overview of IrisNet, a general-purpose, shared infrastructure for developing new sensing applications. We have designed and implemented IrisNet to evaluate some of the ideas presented in this thesis.

Chapters 4 through 7 form the core of this thesis. In chapter 4, we present Synopsis Diffusion, a general framework to compute aggregates over arbitrary topologies. We present the formal foundation of this framework, describe several example algorithms, and provide evaluation results using an efficient multi-path topology called Rings.

In Chapter 5, we show how the aggregation scheme can be dynamically adapted to operating condition changes. We show the benefit of application-aware adaption through

Tributary-Delta, a novel framework to combine the benefits of tree-based and multi-path-based aggregation by running them concurrently in different parts of the network and by dynamically changing their proportion with operating conditions. Tributary-Delta uses Adaptive Rings, a novel and efficient adaptive multi-path topology that exploits a unique property of Synopsis Diffusion. We also present relevant evaluation results.

Starting from Chapter 6, we focus on robust storage of the sensor data (which might be collected by techniques discussed in Chapters 4 and 5). In Chapter 6, we study the impact of correlated failures on a distributed storage system. Based on failure traces collected from three large distributed systems, we show that many design principles used in existing systems are incorrect in practice where failures are often correlated. We also propose new design principles that can mitigate the adverse effects of correlated failures.

In Chapter 7, we describe the design and implementation of IrisNet's distributed storage component. The design is inspired by the lessons we learn in Chapter 6. Moreover, we use SQS for replication, the Paxos consensus algorithm for consistent replica regeneration, and a novel fragmentation technique for dynamic load balancing. Our evaluation shows the behavior of these techniques in an Internet-scale system.

Finally, in Chapter 8, we present a summary of our work and contributions. We conclude this chapter and this thesis with a discussion of general lessons learned about robustness of sensing applications, and outline some directions for future research.

Chapter 2

Background and Related Work

The purpose of this chapter is to provide the background and an overview of related work in the area of data collection and storage in sensor networks. We start Section 2.1 with a brief overview of wireless sensor networks, followed by a discussion on in-network aggregation and existing techniques to make it robust against transient and long-term failures. This will provide the background material to understand the constraints of wireless sensors and the challenges of robust data collection. We then, in Section 2.2, describe existing techniques to build a robust distributed storage system. Existing distributed storage systems do not target sensing applications, and our discussion here will show the limitations of these systems in addressing several characteristics (e.g., workload, scale, correlated failures coming from its Internet-scale deployment, etc.) of a sensing system. At the end of each section, we consider the general design space and show where in this space existing techniques and our solutions fall. Finally, we summarize the discussion of this chapter in Section 2.3.

2.1 Wireless Sensor Networks

This section focuses on robust data collection from wireless sensor networks. To provide the context, we start with a discussion on general properties of wireless sensors and their deployment environments. We then describe existing energy-efficient aggregation schemes and show how they are affected by different types of failures typical in real sensor deployments. Finally, we discuss several existing techniques proposed to make the aggregation schemes robust against failures.

A wireless sensor network consists of a collection of *sensors*, each of which is a small low-cost device equipped with low-power sensing components, a micro-controller, a small

Table 2.1: Hardware characteristics of different motes (taken from [LMG+04]).

Mote	`weC`	rene	`dot`	`mica`	`mica2`	mica2 dot	iMote	btNode
Processor	4 MHz				7 MHz	4 MHz	12 MHz	7 MHz
Flash (code, Kb)	8	8	16	128	128	128	512	128
RAM (kB)	0.5	0.5	1	4	4	4	64	4
Radio (kBaud)	10	10	10	40	40	40	460	460
Radio Type	RFM				ChipCon		Zeevo	Ericson
μ-controller	Atmel						ARM	Atmel
Expandable	no	yes	no	yes	yes	yes	yes	yes
Released	1999	2000	2001	2002	2003	2003	2003	2003

amount of memory, a low power radio, and a battery. Table 2.1 shows hardware characteristics of different generations of the *mote* sensors. Although a single mote has limited capability, when deployed in a large number, they have the ability to measure a given physical space in great detail. A sensor network generally has a base station with sufficient processing power, storage, energy, and connectivity. The base station acts as the interface between users and sensors; i.e., it can collect data from sensors for users to use. Wireless sensors have been used for habitat monitoring [BRY+04, HTB+05, SMP+04], structure monitoring [XRC+04], health [SGW01], education [SMP01], etc.

Experience on building applications has shown several common properties of wireless sensor networks. First, with a range of only a few hundred feet at most, sensors often use multi-hop communication; i.e., they relay data through neighboring nodes to the base station. Second, battery is generally the only source of energy, and it is not feasible to replace batteries in most sensor deployments. Therefore, it is necessary to minimize energy consumption in order to maximize sensors' lifetime. Third, although communication, processing, and sensing, all consume energy, communication is the single most expensive operation. For example, in the current generation of motes, transmitting 1 bit of data consumes almost as much energy as processing 1000 instructions. S. Madden [Mad03] has measured the energy used in various phases of processing during a simple data-collection scenario where a mote transmits one sample of (air-pressure, acceleration) readings every ten seconds and listens to its radio for one second per ten-second period to receive and forward results for a small group of neighboring sensors. The result shows that 97% of energy consumption in this scenario is related to communication, either from directly using the radio or as a result of the processor waiting for the radio to send data to or receive data from neighboring nodes.

There are many techniques to optimize energy usage in a sensor network. Here we mention a few that are relevant to our work. First, a small duty cycle ($< 2\%$) is used so that a sensor is awake (or, its radio is on) for a small fraction of time. Second, sensors use unreliable communication, since reliable communication (e.g., with acknowledgement and retransmission) is expensive. Third, the number of messages to communicate is reduced by additional processing whenever possible. For example, while relaying multiple messages from neighboring sensors, a sensor can process all of them and send only a single processed message. This technique is called *in-network processing* (or *in-network aggregation* when database-style aggregation is performed), and has been shown to be very effective in minimizing energy usage.

2.1.1 Data Processing in Sensor Networks

There are two commonly used models for processing data in sensor networks. The first one is the data streams model where all potentially useful sensor data is pushed from sensors to the base station that archives and processes the data [ABB+03, ZSC+03]. This model is useful when the application does not know a priori what in-network processing to perform, when in-network processing is not feasible, or when all sensor data need to be archived for later analysis. The second model views the sensor network as a database, where data is stored in individual sensors and queries are pushed all the way to the sensors. Since this model can use query-specific in-network aggregation, it consumes significantly less energy than the data streams model. This model is suitable for applications that require aggregate information from the sensors. In many applications, aggregation answers assume greater importance than individual sensor readings. Most applications deploy sensors to sense space, and therefore, a query like "average temperature in the northern part of the forest" (which requires aggregation over all the sensors in that region) is more natural than a query like "temperature of the sensor #101." Moreover, data reported by individual sensors may be noisy; aggregation helps removing, or reducing the effect of, outliers. Finally, in some cases, individual sensors provide too detailed information. For example, in a sensor network monitoring highway traffic flow by sensing road vibrations, the individual readings of sensors (necessarily) deployed every few feet provide far more detail information than needed to answer traffic flow queries.

2.1.2 In-network Aggregation

We here describe a number of previous research efforts on in-network aggregation.

Directed Diffusion

Directed diffusion [IGE+03] has been proposed as a data gathering protocol for sensor networks. Originally, it targeted monitoring of events that are typically sensed only by a few nodes. An example scenario is tracking animal herds in a given geographic region. Diffusion's communication paradigm is based on an information sink's broadcasting requests, or interests, for relevant data. After the interests have been flooded through the whole network, each node knows the "direction" (or, *gradients*) toward the sink node. Then nodes producing relevant information send, and intermediate nodes relay, the data toward the direction to the sink node. Data can be forwarded in a single path, or in multi-path. When all the nodes in the network want to send data to the sink, the data forwarding paths toward the sink form a directed acyclic graph. A node that is supposed to relay data from multiple neighboring nodes can opportunistically aggregate or filter data. For example, it can suppress duplicates or unnecessary data in order to reduce communication overhead.

Although the original proposal of directed diffusion did not focus on duplicate-sensitive aggregates (e.g., Sum), implementing such aggregates over directed diffusion would require each node to send data in exactly one path toward the sink (the base station), and hence the global communication would form a tree topology rooted at the base station. Intermediate nodes in the tree topology then aggregate data in-network as it propagates toward the sink. Such a tree topology is important to avoid double-counting of sensor data.

Tiny AGgregation (TAG)

TAG [MFHH02, MFHH03, MSFC02], implemented in the TinyDB system, provides a declarative interface, through an SQL-like syntax, to process aggregate queries on sensor networks. At a high level, it, like directed diffusion, aggregates data in-network over a tree topology. First, a tree that spans all the sensors is constructed, with the base station as the root of the tree. In its simplest form, the base station (querying node) broadcasts the query, and as it is flooded through the network, each node selects the first node it hears the query from as its parent in the topology. Second, as data is propagated along the tree toward the base station, each non-leaf node performs in-network aggregation over the messages received from its children and forwards only the (partial) aggregation result. For example, in counting the live sensors in the network, the in-network aggregation operation in a non-leaf node adds the values reported by the children nodes and sends the result to its parents. The nodes are loosely synchronized; partial results propagate level-by-level up the tree in distinct epochs, with each node awaiting messages from all its children before sending a new partial result to its parent. In this way, every node in the network sends exactly one fixed-size message; and therefore, this approach requires the optimal n messages in an n-node network. A few other systems, including Cougar [BGS01], use the same technique.

From a query processing perspective, TAG provides several advantages over directed diffusion. First, it provides standard aggregate functions that users can easily use through a well-defined language. In diffusion, such functions are viewed as application-specific components, and must be coded in a low level language. Second, because the functions and their semantics are well-defined, TAG can optimize query processing. Diffusion's user-defined operators are very difficult for a query optimizer to deal with, because it cannot understand their semantics, and thus cannot be sure when it may compose or re-order them with respect to other operators. Therefore, the user bears the burden of getting the placement and ordering of operators correct, significantly complicating the development of complex and correct programs. Finally, TAG's loose synchronization between nodes at different level of the tree topology provides a better opportunity for in-network aggregation, and hence TAG is more energy-efficient.

2.1.3 Robustness Against Transient Failures

Most of the existing sensor applications are deployed in harsh, or even hostile, environments. One of the biggest challenges in collecting or aggregating data in such environments is the high message loss rate. Studies of real deployments have shown that 50% of nodes experience greater than a 10% loss rate and 30% of nodes experience greater than a 30% loss rate, a significantly higher loss rate than that experienced by other wireless networks [ZG03]. As mentioned in Chapter 1, under such high loss rates, aggregation over a tree topology with unreliable communication results in high errors in aggregation answers, because loss of one message or failure of one node results in loss of a complete subtree in the aggregation topology.

We now describe several techniques to address this problem.

Reliable Communication

One approach to combatting transient failures is to use reliable communication such as RMST (Reliable Multi-segment Transport) [SH03] and PSFQ (Pump Slowly Fetch Quickly) [WCK04]). Both the protocols are characterized by hop-by-hop error recovery, repair requests via NACKs that are delivered at a rate faster than the source transmission rate, and in-network caching. They differ in the details of their implementation: PSFQ performs error recovery at the transport layer, while RMST does that at the MAC layer. These protocols can mask transient message losses, making individual links of the aggregation topology more reliable. For example, RMST has been used over directed diffusion for guaranteed delivery of data, even under high loss rates. However, reliable communication protocols use extra messages (e.g., acknowledgements, retransmissions) that have high overhead in terms of energy consumption, channel utilization, and latency [SH03].

Robust Topology

Another approach to robust aggregation is to use an aggregation topology more robust than a tree. For example, our Synopsis Diffusion framework, described in Chapter 4, achieves robustness by using an energy-efficient robust aggregation topology (e.g., multipath), instead of relying on expensive reliable communication. This has the advantage of being robust, without the overheads of reliable communication.

Gossip based aggregation [KDG03] also uses a robust aggregation topology. Each node starts with an initial value, its *mass*, depending on the target aggregate function. The whole aggregation process is loosely synchronized. In the beginning of each round, a node transfers a fraction (e.g., half) of its current mass to a randomly chosen node in the network. At the end of the round, each node combines its current mass with the mass it receives from other nodes in that round. In this way, the total mass in the network is always conserved. It can be shown that, after $O(\log(n))$ rounds, the mass of each node in the network converges to the result of the target aggregate function. Although highly robust, this algorithm has some drawbacks when used in sensor networks. First, to ensure mass conservation, it requires reliable communication; at least, the sender needs to know whether the message has been successfully delivered. Second, a node can not efficiently communicate with any random node in the network, it may need to relay message though other nodes. Third, it ensures only eventual convergence. Fourth, it is not energy-efficient; aside from requiring reliable communication, it needs $O(n\log(n))$ messages, each of which may need to be relayed through multiple nodes. Finally, techniques are known to compute only a small number of aggregates (e.g., Sum, Average, and Count); computing more complex aggregates is still an open issue.

A special case of the above mass conservation principle is the *value-splitting* technique used in TAG with the goal of improving robustness. The idea is to use a directed acyclic graph (DAG) instead of a tree, and have each node with accumulated value v send v/k to each of its k parents. For aggregates such as Count or Sum, this reduces the error resulting from a single message loss from v to v/k, but the expected aggregation error remains as bad as for the tree [MFHH02].

Model-based Aggregation

Model-based aggregation [MPD04] uses temporal correlation of data to correct errors due to transient losses in sensor networks. It uses a model of temporal variation to predict future data. The next observed sensor data is then compared to the predicted value to decide the likelihood of that observed data is erroneous. If the observed data is detected to be erroneous, it reports the predicted value, otherwise, the observed value is reported. The model is adjusted over time based on the observed values. The success of this technique depends on the feasibility of building a good model of the data. In many cases where representative sensor data are not available to build an initial model, or when sensors are supposed to report rare events, such techniques are unlikely to work. Deshpande *et al.* proposed techniques to model sensor data to improve the energy-efficiency of data acquisition [DGM+02]. This class of techniques are orthogonal to other previously described techniques, and can be used on top of them to further improve robustness.

2.1.4 Robustness Against Long-term Failures

A sensor network may experience long-term changes in link quality because of environmental changes or in node membership because of node failures, new deployments, etc. To ensure long term robustness of data aggregation, it is desirable to adapt the aggregation topology to cope with such changes.

In directed diffusion, the sink (i.e., the base station) repeatedly broadcasts exploratory message intended for path setup and repair. When sources detect such a message, they send exploratory messages back, possibly along multiple paths, toward the sink. After the sink starts receiving these messages, it reinforces one particular neighbor (e.g., with the best connectivity) in order to draw down sensor data. Similarly, intermediate nodes select good neighbors and reinforce relevant gradients. The sink can also use negative reinforcement to truncate old (worse) paths. An aggregation topology gradually evolves with a sequence of positive and negative reinforcements, and thus adapts to the changes in the environment.

TAG relies on a local repair scheme in which each node monitors the quality of the link with its current parent and a few neighbors (potential parents). The link quality can be measured by snooping neighbors' broadcasts (assuming symmetric links) or by explicitly asking neighbors to report back their inward link quality (e.g., through acknowledgement messages). When a node sees that the quality (loss rate) to its current parent is significantly worse than that of another potential parent, it "re-parents" itself to improve on the message loss probability.

Our Adaptive Rings topology, described in Chapter 5, is similar to TAG, in the sense that each node keeps statistics on link quality and independently takes decision to adapt the topology. However, it exploits a unique property of our Synopsis Diffusion framework that enables a node to efficiently measure its outgoing link quality just by snooping neighbors' broadcasts (instead of explicit messages from neighbors). Moreover, it adapts a multi-path topology, instead of a tree topology as used by TAG.

In contrast to the above schemes, our Tributary-Delta scheme, described in Chapter 5, adapts both the aggregation topology and the aggregation algorithm (tree vs. multi-path algorithm). Moreover, it uses a combination of local repair and global adaptation.

Topology adaptation has also been studied outside the context of aggregation in sensor networks. In ASCENT [CE04], nodes in a densely deployed network assess their connectivity and adapt their participation in a multi-hop network topology. For example, a node can decide to go into sleep mode if it detects that its absence will not impact message delivery because neighboring nodes can provide good connectivity. Similarly, a node can signal when it detects high message loss rate, requesting additional nodes to wake up and join the network in order to relay messages for it. Xu *et al.* [XHE00] proposed similar techniques to modify a topology in response to local measurements of density. Li *et al.* [LR00] presented a scheme where mobile nodes can modify their trajectory to transmit messages in the context of disconnected *ad hoc* networks. Ramanathan *et al.* [RRH00] proposed some distributed heuristics to adaptively adjust nodes' transmit powers in response to topological changes caused by mobile nodes. All these techniques modify the basic topology on which an aggregation topology is built. Therefore, they are orthogonal to our focus.

2.1.5 Discussion

From the robustness point of view, we can classify existing aggregation schemes using two orthogonal dimensions. The first dimension is whether a scheme uses reliable communication or not. While reliable communication makes an aggregation scheme highly robust, unreliable communication is energy-efficient and more suitable for wireless sensors. The other dimension is the aggregation topology used. A tree topology is simple and can avoid double-counting errors when computing duplicate-sensitive aggregates. However, a more

Table 2.2: Classification of aggregation schemes.

The table classifies aggregation schemes based on whether they use energy-efficient unreliable communication and whether they use a robust topology. Synopsis Diffusion (the shaded cell), described in Chapter 4, is both energy-efficient and robust.

	Unreliable communication	*Reliable communication*
Tree topology	**Energy-efficient**, Not robust E.g., TAG [MFHH02], Directed Diffusion [IGE+03]	Not energy-efficient, **Robust** E.g., Reliable Directed Diffusion [SH03]
More robust topology	**Energy-efficient**, **Robust** E.g., Synopsis Diffusion	Not energy-efficient, **Robust** E.g., Gossip [KDG03]

robust (e.g., multi-path or random) topology, if used correctly, can better mask node- and link-failures. Table 2.2 shows how different aggregation schemes fall within this two dimensional space. Our Synopsis Diffusion scheme, described in Chapter 4, falls under the best region in this space—it does not require the overhead of reliable communication, yet it enjoys the robustness and energy-efficiency of multi-path aggregation topology. We do not show the model-based aggregation scheme in this table since it is orthogonal to, and can be used in conjunction with, the other schemes.

Table 2.3 shows different properties of existing adaptive aggregation schemes. Although both TAG and directed diffusion adapt their aggregation topology (tree), they slightly differ in details of their implementation. TAG's adaptation is completely local—each node, based on low-level properties such as loss rates, independently decides when and how to adapt. Such a local technique is scalable and quick in responding to failures. TAG tries to minimize the depth of the tree; a node chooses a parent that has good connectivity and thus a bushy tree is generated. In contrast, in directed diffusion, the root uses application semantics to decide when to flood exploratory messages that initiate the adaptation process, although each node independently decides the best next hop depending on link quality. Globally deciding when to adapt has the advantage that the sensing application, depending on the user's requirements, can tune the frequency of adaptation and overheads associated with it. Our work in this thesis extends these ideas in two ways. First, our Adaptive Rings topology adapts a multi-path topology. Second, we combine the benefits of adaptive tree-based and adaptive multi-path-based aggregations in a hybrid scheme called Tributary-Delta. It uses the observation that adapting aggregation algorithms in response to network dynamics can provide additional benefits. Tributary-Delta, like directed diffusion, uses a combination of local repair based

Table 2.3: Classification of adaptive aggregation schemes.

Shaded rows show our schemes described in Chapter 5. Tributary-Delta adapts both aggregation topology and aggregation algorithm, and uses a combination of local and global adaptation decisions.

Scheme	Adapting Component	Adaptation Control	Optimization Goal
TAG	Topology (tree)	Local repair	Bushy, robust tree
Directed Diffusion	Topology (tree)	Local repair mechanism + Global repair decision	Robust tree
Adaptive Rings	Topology(multi-path)	Local repair	Robust multi-path
Tributary-Delta	Topology (tree/multi-path) + Algorithm	Local topology repair + Global decision for algorithm selection	Application-specified robustness bound

on low-level properties and global adaptation based on application semantics.

2.2 Robust Sensor Data Storage

Data collected from sensors in an Internet-scale sensing application is stored in a distributed read-write storage system. Distributed storage systems [BTC+04, BDET00, Cat03, DLS+04, DW01, HMD05, KBC+00, YV04] have long been an active area in systems research, due to their fundamental role in computer systems. However, these systems can not be directly used in a sensing system—e.g., a file system does not support rich queries on data and existing distributed database systems do not scale well. This motivates for a new storage architecture for sensing systems, etc.

This new storage architecture for a sensing system can use many existing robustness techniques such as data replication and regeneration (or, self-repairing). Redundancy helps in tolerating transient crash-failures, while regeneration helps in adapting the system to long-term crash-failures. However, the scale and the deployment environment of sensing systems raise a few challenges not addressed by existing techniques. First, failures seen by a large system deployed on the Internet are often correlated. This requires a new approach to determining replication parameters of sensor storage—existing techniques assume independent failure, and the parameters chosen by such independence assumptions often result in significantly suboptimal availability. Second, quorum systems used by existing read-write systems severely limit the availability of the systems, specially when they are deployed on the Internet. This motivates investigating the right choice of quorum systems for sensing systems.

In the rest of this section, we first discuss several standard robustness techniques (e.g., replication, quorum systems, regeneration) and their tradeoffs. We then focus on correlated failures and techniques used by a few existing systems to combat them. We also show limitations of existing approaches and briefly mention our approach to addressing the problems.

2.2.1 Replication and Erasure-coding of Read-only Data

In replication, data is copied to a number of nodes given by the *degree of replication*. Each of the nodes is called a *replica* and the set of nodes is called the *replica group*. The data can be accessed if any of the replicas is available. In erasure-coding [Pla97], used in OceanStore [KBC+00] and CFS [DKK+01], a data object is encoded into n *fragments*, out of which any m fragments can reconstruct the object. Replication can be viewed as a special case of erasure-coding with $m = 1$. Under the assumption of independent failures, it can be shown that erasure-coding (i.e., $m > 1$) provides significantly higher availability than replication for a given redundancy (m/n) [WK02]. However, erasure-coding requires additional overheads for keeping track of fragments and encoding/decoding them. Both replication and erasure-coding can be used with read-write data, but most read-write systems use replication.

2.2.2 Replication of Read-write Data

Sensing systems have read-write data. Replication of read-write data introduces the additional requirements of *consistency*—a read must be served with the "correct" data. What "correct" means depends on the application. Often, it is required that any read sees the latest data (*strict consistency*). However, for some applications, it is acceptable to read stale data with a very small probability (*probabilistic consistency*), or to make sure that all writes will eventually be applied to all the replicas, but at certain moments, different reads may see different values (*eventual consistency*).

Strict Quorum Systems

Quorum systems or *voting systems* determine the sets of replicas readers and writers need to access to ensure consistency. Suppose there are n replicas. Let a *quorum* denote the set of replicas readers or writers access. A strict quorum system defines a set of read-quorums and write-quorums such that intersection of any two read-quorum and write-quorum is nonempty. The intersected replicas ensure that the latest write by a writer is seen by any reader.

Popular examples of strict quorum systems include *read-one-write-all* where a read quorum consists of a single replica, while a write quorum consists of all the n replicas, and *majority quorum systems* where any read- or write-quorum consists of $\lceil (n+1)/2 \rceil$ replicas. If a reader or writer is indeed able to access all the replicas in a quorum, we say the replica group is *available*. Otherwise, the replica group is *unavailable* and the read or write fails. It is known that majority quorum systems achieve the best availability among all strict quorum systems [BGM87]. However, even majority quorum systems severely limit the availability of replicated systems deployed in wide-area networks where node- and link-failures are not rare.

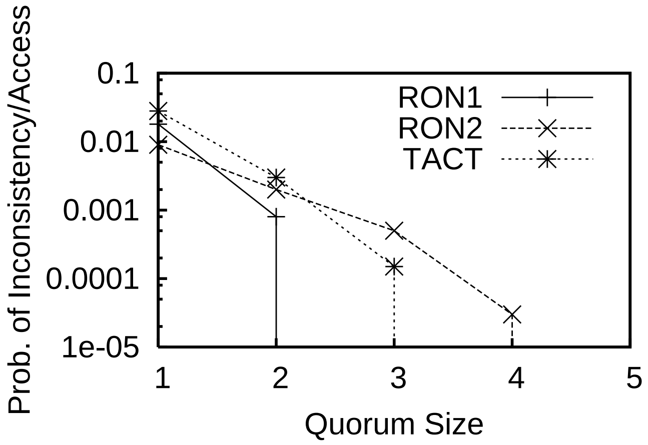
Weak Quorum Systems

The availability of a quorum system can be improved by requiring probabilistic consistency; i.e., permitting the read to miss the latest write with a small probability. Such occasional inconsistency allows using smaller quorums—the system remains available even if fewer replicas than needed by strict quorum systems are available.

Probabilistic quorum systems (PQS) [MRW97] use smaller quorums and provide probabilistic guarantee of quorum intersection. For example, in one PQS construction, quorums are all the sets of size $l\sqrt{n}$ and each quorum is accessed with equal probability. By enforcing an access strategy on quorums, it guarantees an intersection probability of at least $1 - e^{l^2}$.

Signed quorum systems (SQS) [Yu04] further reduce the quorum sizes to be $O(1)$. Let a *mismatch* on a replica be an event when either the reader can access the replica but the writer can not or vice-versa. Mismatch on a replica happens because of network failures; e.g., when the reader is partitioned from the replica and the writer. The intuition behind SQS is that strict quorum systems are overly pessimistic regarding the likelihood of mismatches in the current Internet. Even if the probability of mismatch on one replica may not be small, the probability that we have concurrent mismatches on *multiple* replicas, placed randomly on the Internet, is small (Figure 2.1). SQS use quorums of size m and access them in a way such that inconsistency arises only if m or more simultaneous mismatches happen. As shown in Figure 2.1, this probability can be made arbitrarily small by tuning the value of m (quorum size). A number of specific SQS constructions can be found in [Yu04].

In this thesis, we argue that weak quorum systems are more appropriate to use within



The graph shows the probability of an event that, for a given reader and a given writer, the reader fails to reach a quorum reachable by the writer, or vice versa. For example, for the TACT curve, the probability that the reader can access a quorum consisting of 3 replicas none of which is reachable from the writer, or vice versa, is close to 0.001. The y-axis can also be interpreted as the probability of c simultaneous mismatches, where c is given by the x-axis. These results are taken from [Yu03] and are based on RON [ABKM01] and TACT [YV01] traces.

Figure 2.1: Probability of an inconsistent access with SQS.

sensing systems.

2.2.3 Automatic Replica Regeneration

Availability of a sensing system can be further improved by automatically *regenerating* failed replicas to tolerate additional failures. Namely, when replicas fail, instead of waiting for them to recover, the system can recruit other nodes in the system as new replicas. The hope is that regeneration takes much less time than recovery (e.g., it can be done without human intervention). To regenerate, we need to shift the replica group to exclude the failed replicas and to add new replicas.

Regeneration is tricky in sensing systems, or in any read-write systems that require some form of consistency. For such systems, it is necessary to ensure that there will not be more than one replica group for a given data object. Multiple replica groups may arise if, for example, the original replica group gets partitioned and each partition, suspecting that the other has failed, regenerates into a new replica group. After this happen, writers may write to one replica group while the readers may read from the other one, resulting in data inconsistency.

The standard trick to avoid such replica divergence is to use a regeneration quorum system, independent of the data access quorum system, to ensure that replicas agree on the membership of the new replica group. Existing regeneration systems differ on the choice of such quorum systems. For example, RAMBO [LS02] uses a majority quorum system—replicas must coordinate (using the Paxos consensus protocol [Lam98]) with a majority of the existing replicas to start regeneration. In contrast, Om [YV04] uses a randomized consensus protocol and a witness model [Yu03] that achieves similar functionality as a quorum system. In the witness model, quorum intersection is not always guaranteed, but is extremely likely. In return, a quorum in the witness model can be as small as a single node.

Om improves over RAMBO in terms of regeneration overhead and availability. For example, RAMBO can not regenerate if the regeneration quorum is not available, while Om can regenerate as long as a single node is available. However, we believe that this improvement is not crucial in practice. In most cases, e.g., when a single replica fails, regeneration is not a time sensitive operation; it is sufficient to regenerate before all read-quorums fail.[1] If the regeneration quorum recovers before that time, RAMBO remains almost as available as Om. On the other hand, because Om uses the witness model for regeneration, there is certain probability of replica divergence. Once this happens, the system will remain inconsistent until the bad scenario (e.g., large-scale network partition) goes away. No guarantee is provided on how long such a scenario will last. Finally, because regeneration is rare (as rare as failures) compared to data access, improving the performance of regeneration has little impact on the overall system performance.

This thesis argues that an approach which is opposite to Om is more suitable for sensing systems. In IrisNet, we use a strict regeneration quorum system (so that a replica group never diverges) and a weak data quorum system (so that it improves system availability at the cost of occasional inconsistency). IrisNet's regeneration, described in Chapter 7, has the following additional advantages over Om. First, it is simpler (Om needs a complex randomized consensus protocol). Second, Om strives to optimize the performance of reads, which leads to the design of read-one/write-all, not a suitable design for write-intensive sensing systems. As a side effect, regeneration must be performed upon the failure of any replica in Om. Third, even though regeneration in Om is efficient, it still causes around 20 second service interruption for writes. On the other hand, regeneration in IrisNet can happen in parallel with normal accesses. Table 2.4 compares all the designs.

[1] We can permit all the write-quorums to fail before regeneration because they are not required by the regeneration protocols.

Table 2.4: Comparison of regeneration designs.

The shaded row shows IrisNet's design: it uses weak data access quorum systems to tolerate correlated failures and strict regeneration quorum systems to avoid replica divergence.

	Data access quorum	Regeneration quorum	Consensus protocol
RAMBO	Any traditional quorum (strict quorum)	Majority (strict quorum)	Paxos
Om	Read-one/write-all (strict quorum)	Witness model (weak quorum)	Randomized
IrisNet	SQS (weak quorum)	Majority (strict quorum)	Paxos

2.2.4 Correlated Failures

Because a sensing system is large-scale, is deployed over the Internet, and might be hosted by not-very-well-maintained machines, failures observed by the system will often be correlated, as confirmed by a few recent studies [CV03, YNY+04]. This is in contrast with the failure independence assumptions traditionally used by researchers in studying availability [BTC+04, BDET00, Cat03, DLS+04, DW01, Yu04, YV04]. Correlation may be spatial, when a large number of nodes in a certain geographic region fail; or temporal, when a large number of nodes in the whole system fail within a short period of time. Unless otherwise stated, we use the word "correlation" to denote temporal correlation. Failure correlation has also been observed previously in other systems, including disk failures [BWWG02, CEG+04], processor failures in multi-processor systems [OSM+04], and overlay network routing [CSK02].

Correlated failures have recently drawn the attention of researchers working on wide-area distributed systems [CV03, HMD05, JM02, WMK02, YNY+04], local-area and campus network environments [BDET00, TI92]. Our focus is mostly on wide-area storage systems. One simple approach to addressing correlated failures in such systems is over-provisioning: the best design is selected under an evaluation based on independent failures and then some over-provisioning is introduced with the hope of offsetting the negative effects of failure correlation. A few recent storage systems have explored more sophisticated techniques, as described below.

OceanStore

OceanStore [KBC+00] is a global persistent data store designed to scale to billions of users. It encodes data using erasure-coding and stores it in a distributed hash table. Its strategy in combatting correlated failure is to discover sets of nodes that fail with low correlation, and to store the erasure-coded fragments of a data object in one such set [WMK02]. It relies on node failure data (e.g., when a node goes up and down) collected from human sources and network measurements. This data is used to form a complete weighted graph $G(V, E)$ where a vertex $v \in V$ represents a node in the system and the weight of an undirected edge $(i, j) \in E$ represents the failure correlation of nodes i and j. The failure correlation of two nodes is measured by the similarity (i.e., mutual information [BK00]) between their timings of going up and down. The nodes of the graph are then clustered [SM00] such that the average correlation across clusters is minimized. Thus, two random nodes within the same cluster are highly correlated, while those from two different clusters are more independent in their failure properties.

This design is based on the observation made by Weatherspoon *et al.* [WMK02] and Chun *et al.* [CV03]: failure histories can be used to discover a relatively stable pattern of correlated failures (i.e., which set of nodes tend to fail together). Weatherspoon *et al.* showed that the clusters constructed from the first half of their trace are similar to those constructed from the last half. Given the stability of the clusters, they conjectured that correlated failure can be avoided just by a suitable replica placement algorithm that places the fragments of a single data object in nodes in different clusters. Thus, the fragments do not observe excessive failure correlation among themselves. In some sense, the problem of correlated failures goes away. Such *introspective* failure analysis and modelling can be done online, and thus the system can adapt itself as the failure pattern changes over time.

Phoenix Recovery Service

Phoenix [JBH+05] uses *informed replication* that has a similar design principle as OceanStore: place replicas on nodes with minimal correlation. It considers software diversity as the measure of failure independence, because nodes running the same software often share the same vulnerability and fail together due to Internet catastrophes such as worm attacks. Using host diversity characteristics derived from a measurement study of the hosts on the UCSD campus, the designers of Phoenix developed and evaluated heuristics for determining the number and the placement of replicas. Since typical Internet catastrophes affect a set of nodes running similar software, and Phoenix places replicas on nodes running different software, at least one replica is most likely to survive a catastrophe after which Phoenix can recover the system, by creating new replicas.

Unlike OceanStore, Phoenix does not consider hardware or network failures (which can cause correlation too). OceanStore's approach is more general—a perfect clustering should be able to capture the correlation introduced by similar software, as shown in [WMK02].

Glacier

Glacier [HMD05] takes a different approach than the above two systems. Instead of relying on a predictable failure pattern, it employs excessive redundancy to tolerate the effect of correlated failures. It encodes data using erasure-coding and stores the fragments in nodes over a distributed hash table. Because of the complexity in availability estimation introduced by failure correlation, it uses a very simple failure model that considers only the (single) maximum failure size. It aims to achieve a given availability target despite the correlated failure of up to a fraction f of all the nodes. Such simplification allows the system to use a closed-form formula of $availability = \sum_{k=m}^{n} \binom{n}{k}(1-f)^k f^{n-k}$, which is in fact the same as the formula under independent failures (with f being the failure probability). Using this formula, Glacier is then able to calculate the required system configuration parameters.

2.2.5 Discussion

The standard approach to improving availability of a storage system is to use redundancy (e.g., replication, erasure-coding) and regeneration mechanisms. However, one of the major challenges in designing a sensing system is to determine the parameters associated with these techniques, since real-world failures are often correlated and their nature is not well understood yet.

Existing techniques for mitigating the negative impact of correlated failures can be classified as follows:

- *Avoiding correlated failures:* This class of techniques are based on the conjecture that a replica group can be placed in a set of nodes that demonstrate independent failure properties over time. In this way, a single replica group does not experience correlated failures. OceanStore's introspection and Phoenix's informed replication fall under this category.

- *Tolerating correlated failures:* This class of techniques are based on the assumption that correlated failures can not be avoided based on failure history [HMD05]. Therefore, these techniques use enough redundancy to tolerate the negative effects of failure correlations. Over-provisioning and Glacier's simple-model-based design fall under this category.

Our approach to tolerating correlated failures in a sensing system falls under the second category. Our study of three large distributed systems shows that finding a stable failure pattern of nodes on the Internet is not trivial, and correlated failures can not be avoided by using existing prediction techniques. However, our study confirms that existing techniques of tolerating correlated failures have flaws. We show that over-provisioning may lead to a wrong design—the best design under independent failures may not be the best under correlated failures. We further show that a simple model may either end up provisioning more than necessary or may fail to meet the target availability. We address these shortcomings by i) introducing a better failure correlation model that we have developed from several real world systems we studied, and ii) using the model to carefully choose replication parameters. We elaborate our solutions in Chapter 6.

2.3 Summary

In this section, we have discussed and critiqued existing techniques for making different components of a sensing system robust. Our discussion shows that current solutions do not comprehensively solve the observed problems. In particular,

- Existing aggregation schemes for wireless sensor networks are either not robust (because of their reliance on unreliable communication over tree topology) or not energy-efficient (because of their use of heavy-weight reliable communication). Existing schemes adapt to long-term failures by local repair, without taking application's tolerance for approximate answers into account.

- Existing distributed read-write systems use strict quorum systems. Existing techniques for building robust distributed storage ignore correlated failures, or hope to be able to avoid them, or use simple failure models to choose replication parameters for tolerating the failures. Such techniques often fail to provide the target availability of a system deployed in large-scale over the Internet.

The materials described in this chapter set the stages for our work. In the next few chapters, we revisit the problems, and provide our solutions to them.

Chapter 3

IrisNet

In this chapter, we provide an overview of IrisNet, the first general-purpose shared infrastructure tailored for developing and deploying Internet-scale sensing applications. Note that building a general-purpose sensing infrastructure is not the focus of this thesis. However, since no such systems existed, we had to design and implement IrisNet to show the feasibility of Internet-scale sensing, to implement some of our techniques, and to understand the behavior of sensing applications and the effectiveness of our techniques in the real world.

Our design of IrisNet as a general-purpose infrastructure is motivated by the following two reasons. First, developing a new sensing application is extremely challenging; it requires mechanisms for processing live feeds from high bit-rate sensors, storing and indexing the useful information extracted from the widely-distributed large-scale collection of sensors, processing expressive queries over the data, balancing load among nodes, continuing to function despite component failures, etc. Today there are no effective tools for providing all these functionalities within a single application. A general-purpose infrastructure like IrisNet can take care of all these functionalities, letting application developers develop and deploy new sensing applications with a minimal set of high level description. Second, a generic infrastructure would allow us to explore solutions that address challenges common to a large class of sensing applications, rather than the ones specific to a single sensing application. We use IrisNet to implement some of our techniques and to evaluate them.

IrisNet is designed to address the unique demands of sensing applications, arising from the following prototypical data source and query characteristics:

- **Data source characteristics:** widely distributed, numerical and multimedia data, frequent data updates.

- **Query characteristics:** hierarchically-scoped queries, soft real-time responses, queries over current data and summarized historical trends, queries involving both dynamic sensor data and more static attribute data.

As an illustration of these characteristics, consider the Ocean Monitor application mentioned in Chapter 1. It requires cameras, motes, and other sensors deployed along the coastlines throughout the world. The queries are geographically scoped for oceanographic events around a certain coastline or near some location (location is a static attribute). Current up-to-date data is useful, but historical data is the most relevant (*e.g.,* all sandbar formation events last year) for most queries posed by the oceanographers.

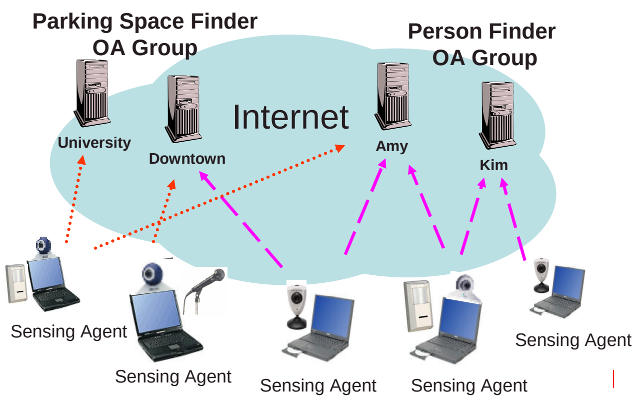
There are three key features of IrisNet that address the above demands and the challenges in developing and deploying new sensing applications:

Handling heterogenous data sources. IrisNet provides means for handling heterogeneous data sources such as high bit-rate video or audio sensors and resource-constrained wireless sensors. For scalability, IrisNet processes sensor feeds at or near their sources. Because the processing is done locally, network bandwidth consumption is dramatically reduced and the system scales to a large number of sensor nodes.

Distributed query processing. IrisNet provides a distributed database for each application, with efficient distributed query processing. As with the sensor feed processing, the goal is to push the query processing to the data, for increased parallelism and so that only the query answer (not the raw data) is transferred on the network. IrisNet provides query routing, caching, load balancing, and replication schemes tailored to the unique query characteristics discussed above.

Ease of application development. IrisNet provides application authors with a very high-level abstraction of the underlying system. IrisNet explores the extent to which a minimal application specification suffices. At its simplest, an application developer writes only a browser front end, application specific code, and an XML database schema for the application. IrisNet derives from these inputs all that is needed to run the application.

We believe that the first two features are fundamental to any successful sensing application infrastructure, and the third feature is crucial for its faster adoption by the community. We have built a working prototype of IrisNet, and studied its performance. A number of applications are being developed on IrisNet, even by developers beyond us. This chapter describes the overall architecture of IrisNet and the applications built on it.



IrisNet has a two-tier architecture. Sensing Agents (SAs) collect and process sensor data while Organizing Agents (OAs) on the Internet organize and store that data. Each sensing application has its own OA group. An SA can also collect data from a wireless sensor network.

Figure 3.1: IrisNet Architecture.

3.1 A Two-Tier Architecture

The IrisNet architecture is implemented using common off-the-shelf hardware and operating systems in combination with our custom user-level software. IrisNet has two different types of software modules: sensing agents (SAs) that collect and filter sensor data and, organizing agents (OAs) that store and organize data in a way users can query it. OAs run on PC-class hosts while SAs run on PDA-class or higher hosts. While both OAs and SAs must be connected to the Internet, SAs have the added constraint that the host must be directly connected to a sensing device.[1] While the total collection of SAs create a sensor feed infrastructure that is shared by all applications, individual groups of OAs create the distributed database and query processing infrastructure for specific applications (see Figure 3.1). In this section, we elaborate on this basic two-tier architecture (SAs and OAs), its benefits, and some of the challenges it creates.

3.2 The SA Architecture

Each SA host is directly connected to one or more sensors. The types of sensors can range from webcams and microphones connected to laptops to motion detectors and pressure gauges in motes. An SA running on the host provides a common runtime environment for applications to share the host's sensor feed(s). The total collection of SAs create a sensor

[1] We refer to the software module as the SA and OA, and refer to the physical machine as the SA host and OA host.

feed processing infrastructure that is shared by all applications. An SA may transmit data to many OAs for the same or different applications (Figure 3.1).

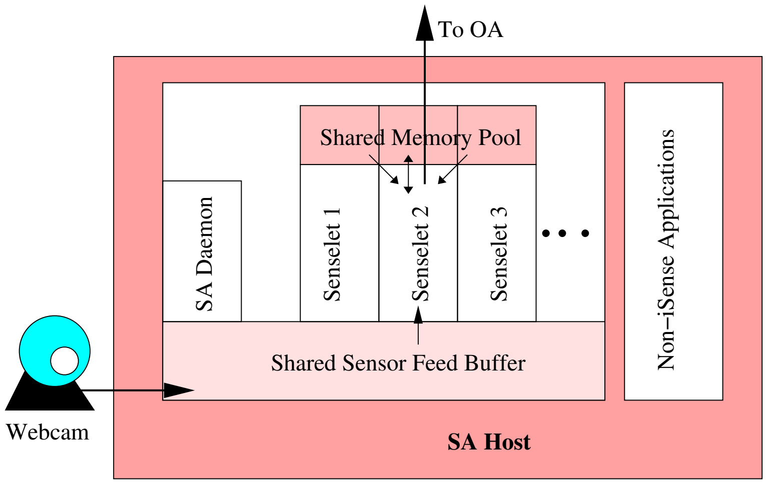
Because rich sensor feeds such as video streams and audio streams are potentially high-bit-rate, IrisNet faces a fundamental scaling challenge: how to make numerous distributed, rich sensor feeds available to users while minimizing the use of network bandwidth. Central to the IrisNet architecture is the aggressive filtering of raw data sources at, or near, the node where the data originate, to achieve drastic bandwidth savings in their transmission across the network. In this section, we describe in detail how IrisNet accomplishes this filtering efficiently.

3.2.1 Senselets: Application-specific Filtering

IrisNet exploits the observation that *application-specific filtering* of sensor feeds yields the greatest reduction in their data rates. For example, a Parking Space Finder (PSF) application may use a 10 frame-per-second (fps) video stream from a camera pointed at a parking lot as its input, but the application need only know which parking spaces are full and which are empty. Running filtering code tailored to the PSF application on the host to which the camera is attached will reduce the sensor feed from a stream of 10 fps video to a series of vectors of { `full, empty` } bits, one per parking space. Similarly, an SA running on the base station of a wireless sensor network can collect application-specific aggregate information from the network and send that, instead of raw data from individual sensors.

IrisNet performs application-specific filtering by using *senselets*, binary code fragments for extracting the useful information from the sensor feed (*e.g.,* existence of riptides from the camera overlooking ocean surface, average temperature reported by the motes along the coastline). The developer of a new IrisNet application writes the senselet(s) and uploads it to the SAs. The IrisNet execution environment on SA hosts provides sensor feed processing libraries with well-known APIs to be used by senselets. We expect typical senselets to be sequences and compositions of these well-known library calls, such that the bulk of the computation conducted by a senselet occurs inside the processing libraries. Note that we do not *require* that a senselet use these libraries; they are merely a convenience to developers, in that they represent a predictable development platform. Using well-known library calls plays a role in efficient sharing of CPU resources by distinct senselets, however, as discussed in Section 3.2.2.

Figure 3.2 shows the execution environment in an IrisNet SA host. An SA host receives one or more raw sensor feeds from directly attached sensors or from base stations of



Sensor data is placed in a shared buffer to which every senselet has access. A senselet can share its intermediate results with others through a shared memory pool. The SA daemon starts and terminates execution of individual senselets.

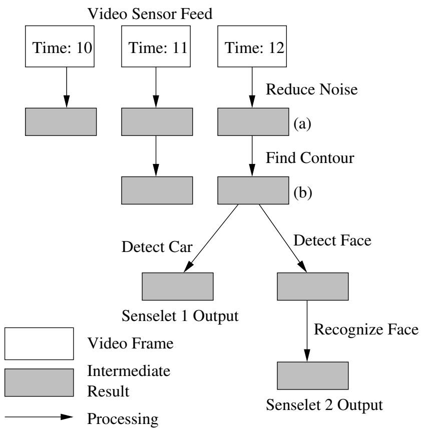
Figure 3.2: Execution environment in SA host.

wireless sensor clusters. One instance of the SA runs on each SA host as a root-privileged, user-level process. The SA is responsible for downloading new senselets, starting them, and terminating them when an application's deployment ends. Each senselet runs as a separate user-level process.

One circular shared-memory buffer for each locally attached sensor is mapped into the address spaces of all senselets. Raw sensed data are periodically written into this shared memory, so that all senselets may read them without incurring a memory-to-memory copy. We discuss the data sharing model for senselets in more detail in next section.

3.2.2 Cross-Senselet Sharing

One sensor feed may be of interest to multiple different IrisNet applications: *e.g.,* a video feed in a particular location may be used in one application to monitor parking spaces, and in another to track passersby in the same visual field. To make sensor feeds maximally available to users of heterogeneous applications, IrisNet must support *sharing* of sensor feeds among multiple senselets. We expect image processing primitives (*e.g.,* color-to-gray conversion, noise reduction, edge detection, *etc.*) to be reused heavily across senselets working on the same video stream. If multiple senselets perform very similar jobs (*e.g.,* tracking different objects), *most* of their processing would overlap. For example, many image processing algorithms for object detection and tracking use background subtraction. Multiple senselets using such algorithms need to continuously



A computation DAG shows the sensor data, intermediate results, and the relevant processing steps. The complete DAG is shown here for the video frame at time 12. A few intermediate results for previous frames are also shown.

Figure 3.3: Computation DAGs for two senselets.

maintain a statistical model of the *same* background [EDHD02].

Consider the two senselets whose computation graphs are shown in Figure 3.3. Senselet 1 finds images of cars in a video stream, while senselet 2 finds images of human faces in the same video stream. Note the bifurcation at time 12, step (b) between senselets 1 and 2; their first two image processing steps, "Reduce Noise" and "Find Contour," are identical, and computed over the same raw input video frame.

We wish to enable senselets like the pair shown in Figure 3.3 to cooperate with one another. In the figure, one senselet could share its intermediate results (marked as (a) and (b)) with the other, and thus eliminate the computation and storage of redundant results by the other.

Each senselet owns a shared-memory region where it has read and write permissions, and has read-only access to other senselets' shared-memory regions. These mappings are enforced by the SA. IrisNet stores senselets' intermediate results in shared memory at run time. This technique is quite similar in spirit to the memoization done by optimizing compilers, where the result of an expensive computation is stored in memory for re-use later, without repetition of the same computation. One key difference between results sharing in IrisNet and traditional memoization is that IrisNet shares intermediate results *between senselets*, whereas memoization shares intermediate results between exactly matching function calls within a *single running executable*. Two different senselet binaries may overlap in only a portion of their computation, and results should only be reused if they were computed using *identical* functions on identical input data. Moreover, senselets are soft real-time processes that act on time series, and may not be willing to use pre-computed results derived from stale raw data.

If a senselet's developer uses the libraries for sensor feed processing provided by the IrisNet infrastructure, most of that senselet's time will be spent within these libraries' functions. IrisNet uses *names* of sensor feed processing API calls to identify commonality in execution. Senselets name intermediate results using their *lineage*, which is a hashed encoding of the entire sequence of library function calls along the path from an original sensor feed to a result in the computation graph. This encoding preserves the ordering on non-commutative function calls. The libraries implement results sharing.[2] Each library function begins by looking for a previously computed intermediate result of the appropriate name in shared memory. If the result is found, and is timestamped within the requesting senselet's expressed tolerance for stale results (slack), it is returned from shared memory, and the redundant computation is avoided. Otherwise, the library performs the computation, and the result is stored under the appropriate name in shared memory for others to use. More details of this can be found in [NKG+04].

Note that the senselet writer need not explicitly refer to the shared store of intermediate results to reap the efficiency benefit of sharing. Rather, the sensor feed processing libraries used by senselets hide this complexity from the developer, and do the work transparently.

3.2.3 Incorporating Wireless Sensors

An SA can run in a base station of a mote network. In this way, IrisNet can tie together multiple mote networks and let an application use data from some or all of them. The SA in the base station has a slightly different set of functionalities than the SA attached to wired sensors. The SA performs the tasks necessary to manage the mote network, including asking the motes to sleep when no application requires their data, monitoring the motes [ZGE03] and providing feedbacks to administrators about when additional motes need to be deployed, retasking the motes by installing necessary software [LGC05, LMG+04] (for self-organizing, routing data, etc.), etc. It provides APIs for applications to perform standard tasks (e.g., topology formation, aggregation, raw data collection, etc.) within the mote network. For example, it can incorporate TinyDB [MFHH02] to query the motes and expose its APIs to applications. The SA also lets applications to transparently share valuable resources—by caching data collected for one application and

[2] IrisNet provides a parallel, non-shared API to the sensor feed processing libraries. If a senselet wishes not to share its results with others, it uses this API rather than the standard API, and allocates storage on the heap rather than in shared memory.

letting others to use that, by merging sensing tasks required by multiple applications, etc.

Senselets running on the SA encapsulate the code to process sensor data collected from the motes and the code that can be pushed into the mote network to reprogram it. The latter part is required if the standard APIs provided by the SA to program the motes is not sufficient. For example, the application may want to compute an aggregate that is not supported by TinyDB provided by the SA. In such case, the senselet provides the code, compiled to be run on motes, to implement the missing functionalities. The SA then installs the code within the motes, starts the task, and terminates it when dictated by the application.

In this thesis, we consider SAs collecting data from wireless sensors. More specifically, we focus on efficient and robust collection of aggregate data from wireless sensors. An application requiring efficient and robust data collection implements our techniques into its senselet that SAs push inside the wireless sensor networks; individual sensors then use our algorithm and push aggregate data to the SAs for the application to use.

3.3 The OA Architecture

OAs implement the data storage component of the generic sensing system architecture mentioned in Chapter 1. OAs are organized into groups, one group per application. A group of OAs creates the distributed database and query processing infrastructure for a specific application. Each OA participates in one sensing application (a single physical machine may host multiple OAs). Each OA has a local database for storing sensor-derived data; these local databases combine to constitute an overall application database. Using multiple OAs, *i.e.,* essentially using a distributed database, is necessary to support the high update rates that may result from sensor readings. Using separate OA groups allows each application developer to tailor the database schema to the application, and to facilitate application-workload-specific caching and load balancing.

IrisNet stores data in a distributed XML database. It provides application developers with a very high-level abstraction of the underlying system, in which the XML database is completely centralized. In this section, we describe how IrisNet takes an application's XML document and creates a *distributed* application database, which supports updates by the application's senselets and answers queries generated by the application's front end. Although distributed databases are a well-studied topic (e.g., [SAS+96a, GHOS96, PL91, AS93, KB91, ABGM90, CP92, OW02]), no work prior to ours shows how to dynamically distribute an XML document or how to perform query processing over a distributed XML document.

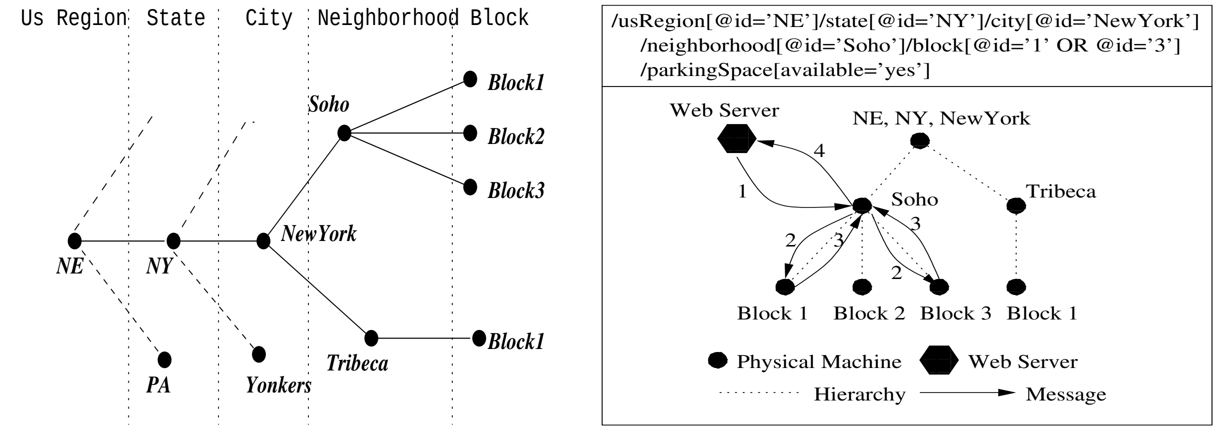
We here briefly describe the components of the OA architecture that are relevant to this thesis: distributed database, distributed query processing, and fault tolerance.

3.3.1 Distributing the Database

In order to adapt to query and update workloads, IrisNet can dynamically partition the sensor database among a collection of OAs. The logical hierarchy (e.g., Figure 3.4(a) for a PSF application) used by the application's queries is given by the application developer as a database schema. The schema includes special `id` attributes, whose value is a short name that makes sense to include in a user query (*e.g.,* New York). These attributes define potential split points for partitioning the database, *i.e.,* the minimum granularity for a partitioned unit. Unlike approaches with a fixed granularity (*e.g.,* a file block), the granularity can vary widely within the same application database. IrisNet permits an OA to own *any* subset of the nodes in the hierarchy (including non-contiguous subsets), as long as ownership transitions occur at split points and all nodes are owned by exactly one OA. (OAs can also cache/replicate data owned by other OAs, as discussed latter.) We envision that application developers will liberally include `id` attributes in their schema, to give IrisNet maximal flexibility in partitioning, while IrisNet will typically partition the document into far fewer actual partitions.

Global naming is achieved by requiring that (1) the `id` of a split node is unique among its siblings with the same parent (*e.g.,* there can be only one `city` whose `id` is `New York` among the `NY` state node's children), and (2) the parent of a non-root split node is also a split node. Thus, the sequence of node names and `id`s on the path from the root uniquely identifies a split node. An important feature of our design is that because the values of `id` attributes make sense in XML queries, we can extract any global name we need directly from the query itself! From the query in Figure 3.4(b), for example, we can extract the global name `city-NewYork.state-NY.usRegion-NE` for the New York node in Figure 3.4(a), by concatenating the name and `ID` pairs from the query in Figure 3.4(b). We are not aware of any previous global naming scheme that leverages XML query languages in this manner.

Each OA registers with DNS [MD88] each split node that it owns, where the registered name is the split node's global name appended with the name of the application and our domain name. DNS provides a simple way for any node to contact the owner of a particular split node of the database. It is the only mapping from the logical hierarchy to physical IP addresses in the system, enabling considerable flexibility in mapping nodes in the document to OAs, and OAs to physical machines. This permits the system to



(a) A logical hierarchy (b) Processing a query

Figure 3.4: Processing a IrisNet query over the logical hierarchy. (a) A logical hierarchy. (b) Top: An XPATH query. Bottom: A mapping of logical nodes to seven machines, and the messages sent to answer the query (numbers depict their relative order).

scale to as many machines as needed, each operating in parallel, in order to support large data volumes and high update frequencies. In IrisNet, an OA keeps track of both the computational load on its host machine and the frequency of queries (and of updates, if it is a leaf OA) directed to each split node it owns. An overloaded OA splits off part of the document it owns to a lightly loaded machine [NGS05]. Conversely, a very lightly loaded OA is collapsed into its parent. More details of IrisNet's load balancing will be discussed in Chapter 7.

3.3.2 Answering Queries

Due to IrisNet's dynamic partitioning, providing fast and correct answers to user queries is quite challenging. The goals are to route queries directly to the nodes of interest to the query, to take full advantage of the data stored at each OA visited, and to pass data between OAs only as needed. We show how IrisNet addresses each of these goals.

An IrisNet query is specified in XPATH [WWW99], a standard query language for XML data. An XPATH query selects data from a set of nodes in the hierarchy. In IrisNet, the query is routed directly to the lowest common ancestor (LCA) of the nodes potentially selected by the query. IrisNet uses a simple parser to scan the query for its maximal sequence of `id`-only predicates (*e.g.,* up to the New York node for the query in Figure 3.4(b)). This constitutes a non-branching path from the root of the hierarchy to the node, such that the query answer will come from the subtree rooted at that node. In the common case where the `id`-only chain ends at the place where the query no longer selects just a single path, the node is the desired LCA for that query. In Figure 3.4(b), for example, the Soho node is the LCA because the query selects from two of the node's children (Block 1 and Block 3). The simple parser constructs the DNS name for the node (as detailed above), performs a DNS lookup to get the IP address of the machine hosting the Soho node, and routes the query to the OA. We call this the *starting point* OA for the query. The key point is that for an arbitrary XPATH query posed anywhere in the Internet, IrisNet can determine where to route the query with just a DNS lookup: no global or per-application state at the web server is needed to produce the DNS name of the LCA. Also, the root of the hierarchy is not a bottleneck, because queries are routed directly to the LCA, which, for the typical hierarchically-scoped query, is far down in the hierarchy.

Upon receiving a query, the starting point OA queries its portion of the overall XML document and evaluates the result. However, for many queries, a single OA may not have enough of the document to respond to the query. The OA determines which part of a user's query can be answered from the local document (discussed below) and where to gather the missing parts (extracting the needed global names from the document). The OA looks up the IP addresses of the other OAs to contact and sends subqueries to them. These OAs may, in turn, perform a similar gathering task. Finally, the starting point OA collects the different responses and the combined result is sent back to the user. For the example in Figure 3.4(b), the Soho OA receives the query from the web server, sends subqueries to the Block 1 and Block 3 OAs, each of which returns a list of available parking spaces, to be combined at the Soho OA and returned to the user.[3]

While the above recursive process is straightforward at a high level, the actual details of getting it to work are rather complicated. The crux of the problem is to determine which part of an XPATH query answer can be extracted from the OA's local document (a fragment of the overall application document). Simply posing the query to the XML database will not work because the fragment may be missing nodes in the document that are part of the answer or fields in the document that are needed to correctly evaluate a predicate. In short, existing query processors are not designed to provide negative information about missing parts of an answer. Our solution to this problem is to tag the nodes in a fragment with status information that indicates various degrees of completeness and to maintain tagging/partitioning invariants (such as partitioning only at split points).

[3] This is a simple example in which the children of the Soho node reside on different machines, but in general, a subquery may be sent to any descendant in the hierarchy, depending on the query and how the document is fragmented.

Then, we convert the XPATH query into an XSLT program that walks the OA's XML document tree and handles the various tags appropriately. Complete details on the approach appear in [DNGS03].

IrisNet also lets application developers easily extend its existing database functionality. To process and aggregate sensor data in a new way (e.g., to fuse data from multiple sensors), beyond the ways supported by IrisNet's database engine, a developer needs to decompose the target task into three functions, with a semi-centralized view of execution in mind; IrisNet then automatically distributes the processing and lets users seamlessly incorporate the new functionality with the existing query interface. More details of this feature can be found in [CGN05b].

3.3.3 Caching and Data Consistency

Like many other distributed applications, it is obvious that there will be a great deal of locality in the user requests to a sensing application. For example, in a PSF application, there are likely to be many more queries about downtown parking than rural/suburban parking, as well as spikes in requests for near the stadium on game day. To take advantage of such patterns, OAs may cache data from any query gathering task that they perform. Subsequent queries may use this cached data, even if the new query is not an exact match for the original query. During the gathering task, IrisNet generates subqueries that fetch partitionable units; in this way, its query processing mechanisms (described above) work for arbitrary combinations of owned and cached data.

Due to delays in the network and the use of cached data, answers returned to users may not reflect the most recent data. A query may specify consistency criteria indicating its tolerance for stale data. For example, a PSF application can specify a decreasing tolerance for stale data as a user approaches her destination. We store timestamps along with the cached data, indicating when the data was created, so that an XPATH query specifying a tolerance is automatically routed to the data of appropriate freshness.

3.3.4 Fault Tolerance and Replication

While a hierarchical database organization enables the scalable processing of rich queries, it is more susceptible to failures. For example, the failures of nodes high up in the hierarchy can cause significant portions of the database to become unavailable. Other hierarchical systems, such as DNS [MD88], rely on replication to tolerate such failures. However, DNS replication is relatively simple since DNS records are rarely updated.

Unfortunately, sensor readings change frequently making replication more difficult in IrisNet.

IrisNet achieves fault tolerance through two mechanisms. First, queries are first sent to the LCA OA, and hence failure of the OAs containing nodes above the LCA node in the hierarchy does not affect the query processing. Second, IrisNet replicates nodes in the logical hierarchy on multiple OAs. Each node in the hierarchy has multiple primary replicas. When the DNS name for a node is resolved, it returns the addresses of all primary replicas. Typically, primary replicas are assigned to nodes that are optimal in placement. For example, a leaf node primary replica is likely to be near its associated SAs.

IrisNet also uses secondary replicas. Secondary replicas only maintain a weakly consistent copy of the data corresponding to the node. Secondary replicas are often placed far from the primary replicas to avoid simultaneous failures of these hosts. We store the location of these secondary replicas separately from the primary replicas by using an alternative trailing domain name. Querying this alternate name retrieves the address of all primary and secondary replicas for the node. One other important distinction is that the DNS records for the alternate name are given a TTL of 0 while those for the normal name are given a TTL of 10 minutes. The low TTL for the alternate DNS records ensures that an up-to-date list of replicas is retrieved during the critical periods when the secondary replicas are used.

When a host attempts to route a query to an OA, it may fail in a number of ways: the TCP connection setup may fail, the query may timeout, the contacted host may no longer be responsible for the node, etc. Upon any failure, the host re-issues the query to the next primary replica. If all primary replicas fail, the host performs a DNS lookup on the alternate node name and queries any newly discovered replicas in a similar fashion. The current IrisNet prototype does not spawn new replicas when a failure is detected.

More details about IrisNet's replication will be described in Chapter 6 and 7.

3.4 IrisNet Applications

A key goal in IrisNet is to greatly simplify the task of developing a sensing application. We now briefly describe how IrisNet lets application developers develop new applications, and provide examples of a few applications developed on IrisNet.

3.4.1 Developing an Application in IrisNet

A typical developer needs to write only the following:

- Application specific code: senselets and application-specific database aggregation functions, if not already supported by IrisNet's distributed database.

- A database schema that describes and organizes the collected sensor readings (*e.g.,* a schema that describes the characteristics of a parking spot, including its hierarchical, geographic location); and

- A user-friendly browser front end that converts user input into queries upon the application database.

Note that the developer is not burdened with the details of collecting the sensor readings, processing queries on the widely distributed data, scaling the application to the wide area, enforcing the appropriate privacy policies, protecting the applications from failures and attacks, etc.

3.4.2 Prototype Applications

In this section, we describe several applications built on IrisNet to show how different pieces described so far fit together. Two of these applications, the IrisLog and the Parking Space Finder, were developed by us, while the third one, the Ocean Monitor, was developed by a group of oceanographers from Oregon State University. We use IrisLog to evaluate some of our techniques in Chapter 7.

IrisLog

IrisLog [Int03a] is a distributed infrastructure monitor that demonstrates the scalability of IrisNet. IrisLog is deployed in PlanetLab [PACR02, Pla05], an open, globally distributed testbed and has been running since September 2003. Currently, IrisLog monitors over 450 PlanetLab hosts at 270 sites spanning 5 continents, and is the largest IrisNet deployment to date. Rather than using physical sensors, this application uses machine statistics, e.g., CPU load or network bandwidth consumption, and system logs as sensor inputs. It allows efficient querying of both individual and aggregate machine statistics and resource utilization across the PlanetLab infrastructure.

The database schema used in IrisLog employs a geographic XML hierarchy, a slice of which is shown in Figure 3.5. This hierarchy allows efficient processing of geographically

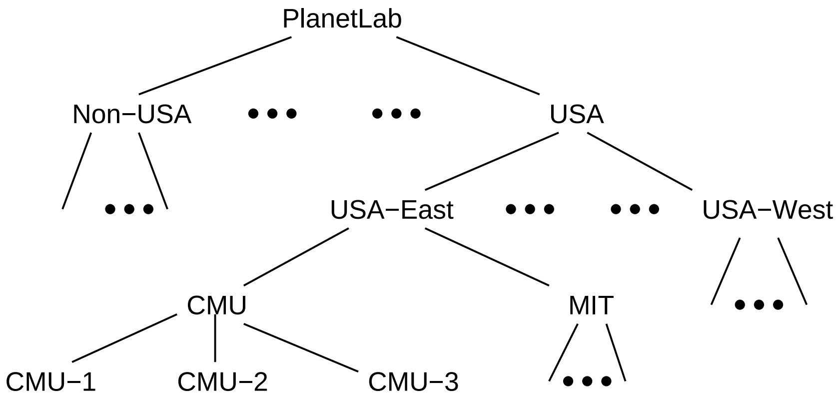


Figure 3.5: Hierarchy used in the IrisLog application.

scoped queries (e.g., find the least loaded CMU node). At each host machine, a set of monitoring tools is executed periodically to log machine and user statistics, which are used to update locally-hosted fragments of the distributed database corresponding to the machine. To support historical queries on the monitored data, the schema uses multi-resolution vectors to store each monitored metric. These vectors provide higher resolution samples of recent data than older data. Fragments corresponding to higher levels of the XML tree are automatically distributed among various machines, based on query load and performance, and replicated for fault tolerance. Compared to a system that streams log information from each host to a centralized monitoring station (e.g., Ganglia [MCC04]), IrisLog both distributes the processing load for handling user queries, and reduces total bandwidth for handling statistics updates.

Parking Space Finder (PSF)

The Parking Space Finder (PSF) is intended to provide the useful service of locating available parking spaces near a desired destination and directing the driver to such a space. The system utilizes a set of cameras connected to IrisNet SAs running senselets to detect the presence of cars in spaces and update the distributed database with this high-level semantic information. The database itself is organized according to a geographic hierarchy, and is logically divided by region, city, neighborhood, block, etc. Our current table-top deployment uses the logical hierarchy shown in Figure 3.4(a). This hierarchy fits well with the application, as any update from a given camera or query from a given driver is likely to touch only small subtrees of the database, improving the scalability of the distributed system.

The PSF front-end is a web-based interface that takes as input the desired destination and current location. It queries IrisNet for the closest spot to the destination that is not occupied, and that matches other user-specified parameters, e.g., whether covered, if a permit is required, maximum hourly rate, etc. The front-end then uses Yahoo!® Maps online service to generate driving directions to the available parking spot. We imagine that in the future, this front-end can be integrated into a car's navigation system, and would be able to get current location and destination directly from the system, and make use of the built-in mapping to generate driving directions.

The PSF is able to handle some real-world constraints on deployed camera systems. For example, a single camera may not be able to cover a particular parking lot. Our current PSF senselet is able to use feeds from multiple, oblique camera views and stitch them together to produce an image that covers the entire lot, before running car detection routines. The detector uses variance of pixel intensity in image regions to determine whether a parking space is occupied. A more sophisticated detector could employ machine learning techniques to acquire visual models of empty spaces, or employ techniques to directly determine the presence of cars.

Ocean Monitor

In collaboration with oceanographers of the Argus project [CIL03] at Oregon State University, we have developed a coastal imaging application on IrisNet. SAs connected to cameras deployed along the Oregon coastline run senselets to detect and monitor near-shore phenomena, such as riptides and the formation of sandbars.[4] The system can capture and store still and temporally smoothed images (Figure 3.6), essentially raw data, and also distilled, high-level information processed through senselets. Using IrisNet senselets, the application allows oceanographers to run their detection algorithms at the remote site. This permits a greater fidelity of observation (more images can be examined) than is possible through previous efforts that collected raw data over low-bandwidth modem and long-range radio links for centralized processing. Users can dynamically change parameters of the senselets, vary data sampling rates, and even install new processing algorithms to the remote camera sites, without interrupting service or making a trip to the coast.

One important type of oceanographic image-based sensor is the pixel stack. These detectors track time-varying intensities of series of small regions of interest (ROIs) in coastal images, and correlate these changes with various phenomena. An important as-

[4] Our current deployment uses only one live camera on the Oregon coast.

Figure 3.6: Images from the IrisNet Coastal Imaging prototype. On the left, raw video frames. On the right, temporally smoothed images revealing the sand bars.

pect of using these detectors is the correct association of regions in the images with real-world locations and coordinates. Extrinsic calibration techniques applied in IrisNet [CGN+05a], along with a few known ground truth points, can be used to accurately determine the correspondences between image regions and global coordinates. Furthermore, Ocean Monitor implements a technique to provide a composite overhead view of the coastline, by projecting multiple camera images onto global coordinates and stitching them together [CGN+05a]. Such a rectified, composite image is designed to assist in the instrumenting of pixel sensors, permitting the user to select locations based on a world or map coordinate frame, which the system can automatically convert to ROIs in the source images.

3.5 Summary

IrisNet is a shared software infrastructure tailored for easy development of Internet-scale sensing applications. It implements the generic functionalities required by such applications so that application developers can use IrisNet as a building block. It enables application-specific processing of sensor data, stores sensor-derived data near the sources, and supports a general purpose query processing on that distributed data. Initial experience of building applications on IrisNet shows the feasibility of Internet-scale sensing and the ease of application development on IrisNet. Our existing prototype applications all use wired sensors; building applications incorporating wireless sensors is part of our future work.

As mentioned before, this thesis focuses on robustness of data collection and storage.

In the next two chapters, we will focus on how a senselet running on a base station can reliably collect aggregate data from wireless sensors. The following two chapters will involve the storage layer of IrisNet and focus on its fault tolerance.

Chapter 4

Synopsis Diffusion

As mentioned in Chapters 1 and 3, a generic sensing system such as IrisNet consists of two main components: the data collection component and the data storage component. In this chapter and the next chapter, we consider the data collection component. More specifically, we focus on collecting aggregate data from wireless sensor networks in robust and efficient ways. We describe our solutions in two steps. First, we consider a relatively static network and propose a novel in-network aggregation framework called *Synopsis Diffusion* that provides highly-accurate estimations of useful aggregates in an energy-efficient way. Then, we consider the dynamics of a typical wireless sensor network and propose a novel adaptive aggregation framework called *Tributary-Delta*, which efficiently combines Synopsis Diffusion with existing tree-based schemes and dynamically adapts to the dynamics. We describe Synopsis Diffusion in this chapter; Tributary-Delta will be described in the next chapter.

The key strength of Synopsis Diffusion comes from its decoupling the aggregation algorithm from the underlying aggregation topology. This enables in-network aggregation over energy-efficient and highly robust multi-path topologies. Due to the robustness provided by multi-path routing [GGSE02], Synopsis Diffusion can estimate aggregate answers with high accuracy, even with high message loss rates and without reliable communication. Synopsis Diffusion is based on three key ideas:

- **Exploit the wireless broadcast medium.** Existing schemes of aggregation on a tree superimpose a point-to-point message exchange on an inherently broadcast medium; Synopsis Diffusion enables any and all listeners to take advantage of the messages they hear. In this way, aggregate data can propagate over multiple paths toward the base station. We show that Synopsis Diffusion using a broadcast-based topology (a Rings topology, as described in Section 4.2) is as energy efficient as tree-based aggregation, but dramatically more robust.

- **Use order- and duplicate-insensitive (ODI) synopses to summarize partial results.** With Synopsis Diffusion, each node sends its *partial result* along multiple paths in the aggregation topology, introducing the concern that readings will be double-counted and will reach the base station in different orders. Synopsis Diffusion addresses this by using *order- and duplicate-insensitive* (ODI) synopses, small-size digests of the partial results received at a node, such that any particular sensor reading is accounted for only once. In other words, the synopsis at a node is the same regardless of the number of times a given reading from a given sensor arrives at the node (either directly or indirectly via partial results).

- **Decouple aggregate computation from message routing.** With Synopsis Diffusion, unlike with existing tree-based aggregation schemes, the aggregation topology (i.e., *which* nodes send and receive messages *when*) can be decided independently of the aggregation algorithm. The goal for an aggregation topology is to maximize the number of distinct sensor nodes able to send their values to the base station, while minimizing the number of messages sent (i.e., the energy consumption). There is a continuum to this trade-off, and Synopsis Diffusion provides the flexibility to use any point on this continuum, to incorporate any desired level of redundant propagation of readings and partial results (for protection against possible failures), and to be fully adaptive to the node and link dynamics actually encountered during routing.

Previous sensor network research has shown that for aggregates that are inherently duplicate-insensitive (e.g., Max and Min), robust in-network aggregation using broadcast is both low-energy and highly accurate [ZGE03]. However, most aggregates are duplicate-sensitive (e.g., Sum, Count, Average), and the challenge is to devise duplicate-insensitive synopses for these duplicate-sensitive aggregates. To this end, we develop a formal definition of duplicate-insensitive synopses. This definition captures the overall goal of duplicate-insensitivity, but it is not immediately useful for designing synopses. Thus we derive simple properties that provably imply the more general definition, and show how these can be used to design (provably correct) synopses. We believe that Synopsis Diffusion is a powerful framework for the design of sensor network aggregation schemes. Instead of tightly coupling in-network aggregation and message routing, Synopsis Diffusion permits independent explorations of these areas.

The remainder of the chapter is organized as follows. Section 4.1 describes a naïve algorithm that conveys the basic idea of ODI aggregation. Section 4.2 presents the basic

Synopsis Diffusion scheme. Section 4.3 presents our formal framework and theorems for ODI synopses. Section 4.4 presents ODI synopses for additional aggregates. Section 4.5 discusses error bounds of approximate Synopsis Diffusion algorithms. We describe our experimental results and various trade-offs that Synopsis Diffusion enables in Section 4.6. We conclude the chapter with related work in Section 4.7 and a short summary of the chapter in Section 4.8.

4.1 A Naïve ODI Algorithm

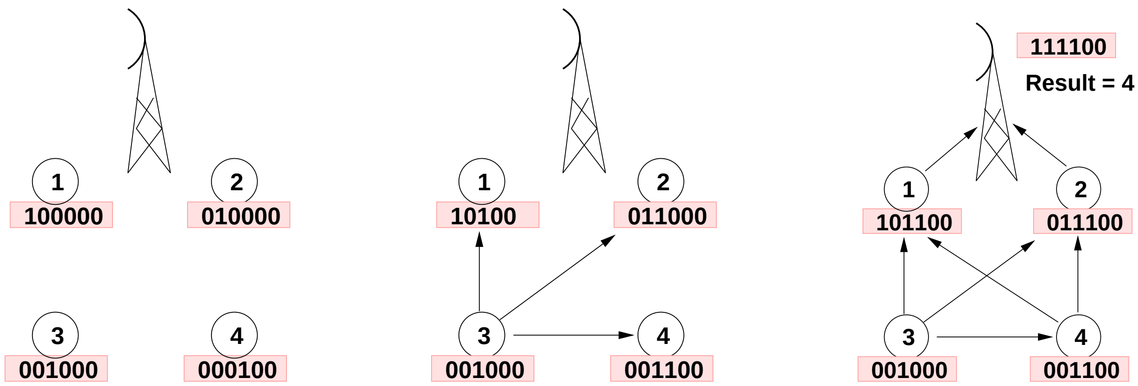
The basic requirement of an ODI algorithm is to ensure that the final result computed at the base station is independent of the underlying topology—i.e., whatever the topology is, in whichever ways sensor readings get combined inside the network, the end result must always be the same (e.g., same as the result provided by a tree topology). We will formally define ODI algorithms in Section 4.3.2.

We start with a naïve ODI algorithm that can compute a duplicate-sensitive aggregate, Count of live sensors in a network, in-network over an arbitrary topology. Note that existing tree-based schemes for Count, where each node adds its children's accumulated counts and sends the sum to its parent(s), are not ODI and they do not work with an arbitrary topology—the same reading may be counted more than once if the topology is not a tree (and hence each reading can be routed through multiple different paths). The following algorithm avoids this problem.

At the beginning of aggregation, a node with ID i uses the function Bit-Vector that generates a bit-vector with only the i'th bit set to 1 (Figure 4.1(a)). Assuming that each node has a unique ID, different nodes set different bits in the bit-vector. Then the nodes broadcast their bit-vectors. On receiving a bit-vector from its neighbor in the aggregation topology, a node combines it with its current bit-vector by using the Boolean OR function (Figure 4.1(b)). In this way, bit-vectors generated by different nodes get combined in-network as they propagate toward the base station. Finally, the base station uses the function Bit Count to count the number of 1 bits in the final bit vector and reports the count as the final answer (Figure 4.1(c)).

Intuitively, the above algorithm correctly computes Count because every node sets a unique bit in the bit-vector and the Boolean OR operation preserves all the bits so that the base station can count them all. Moreover, the algorithm is ODI since in-network aggregation is performed by the Boolean OR function which is ODI. Section 4.3 provides a formal definition of ODI and a framework for proving that an algorithm is ODI.

The above naïve algorithm shows several important aspects of the class of ODI algo-



(a) Generating bit-vectors by individual nodes

(b) Combining bit-vectors in intermediate nodes

(c) Counting bits at the base station

In (a), each Node converts its reading to be aggregated into a synopsis. In (b), each node broadcasts its own synopsis. A node receiving this synopsis merges it with its own synopsis and then broadcasts the merged synopsis. After a sequence of such broadcasts, the base station receives a (merged) synopsis that represents synopses of all the nodes in the network. In (c), the base station evaluates the synopsis into the final answer to the aggregate query.

Figure 4.1: A naïve ODI Count algorithm.

rithms we aim to devise. First, the target function (i.e., Count) has been implemented by a set of helper functions: Bit-Vector, Boolean OR, and Bit Count. These functions have been chosen carefully such that they produce the same result we expect from the target Count function and they have certain properties (discussed in Section 4.3) that make the aggregation process ODI. Second, the partial result exchanged between two nodes is represented as a synopsis [BBD+02, GM99], a small digest (e.g., bit-vector, histogram, sample, etc.) of the data. Finally, to be energy-efficient, the synopsis needs to be small. In this respect, the above naïve algorithm is not efficient—to allow every node a unique bit position, the size of the bit-vector needs to be at least n, where n is the number of nodes participating in the aggregation. Later, in Section 4.4, we will provide an approximate Count algorithm that requires a bit-vector of size only $O(\log(n))$.

4.2 Synopsis Diffusion

We now describe Synopsis Diffusion, a novel in-network aggregation framework that enables robust, highly-accurate estimations of duplicate-sensitive aggregates like Count. We first describe the general aggregation framework, and then illustrate its use by using a specific aggregation topology (called *Rings*). Although the description is based on adapting the TAG communication model and continuous query scheme [MFHH02], it is not dependent on the particular model or scheme.

The in-network aggregation of a Synopsis Diffusion algorithm is defined by three functions on the synopses:

- **Synopsis Generation:** A synopsis generation function $SG(\cdot)$ takes a sensor reading (including its metadata) and generates a synopsis representing that data.
- **Synopsis Fusion:** A synopsis fusion function $SF(\cdot,\cdot)$ takes two synopses and generates a new synopsis.
- **Synopsis Evaluation:** A synopsis evaluation function $SE(\cdot)$ translates a synopsis into the final answer.

The exact details of the functions $SG()$, $SF()$, and $SE()$ depend on the particular aggregate query to be answered. An example is given at the end of this section; additional examples are presented in Section 4.4.

A Synopsis Diffusion algorithm consists of two phases: a *distribution phase* in which the aggregate query is flooded through the network and an aggregation topology is constructed,[1] and an *aggregation phase* where the aggregate values are continually routed toward the base station. During the aggregation phase, each node periodically uses the function $SG()$ to convert sensor data to a local synopsis and the function $SF()$ to merge two synopses to create a new local synopsis. For example, whenever a node receives a synopsis from a neighbor, it may update its local synopsis by applying $SF()$ to its current local synopsis and the received synopsis. Finally, the base station uses the function $SE()$ to translate its local synopsis to the final answer. The continuous query defines the desired period between successive answers, as well as the overall duration of the query [MFHH03, YG03]. One-time queries can also be supported as a special, simplified case.

An important metric when discussing the quality of query answers in the presence of failures is the fraction of sensor nodes contributing to the final answer, called the *percent contributing*. With Synopsis Diffusion, a sensor node contributes to the final answer if there is at least one failure-free "propagation path" from it to the base station. A *propagation path* is a hop-by-hop sequence of successfully transmitted messages from the sensor node to the base station. Note that it does not require that the sensor's reading

[1] This topology may be adapted at any time as network conditions change.

actually be transmitted in the message, because with in-network aggregation, the reading will typically be folded into a partial result at each node on the path.

Although the Synopsis Diffusion framework is independent of the underlying topology, to be more specific, we describe next an example overlay topology, called *Rings*, which organizes the nodes into a set of rings around the base station.

4.2.1 Synopsis Diffusion on a Rings Overlay

During the query distribution phase, nodes form a set of rings around the base station q as follows: q is in ring R_0, and a node is in ring R_i if it receives the query first from a node in ring R_{i-1} (thus a node is in ring R_i if it is i hops away from q). The subsequent query aggregation period is divided into *epochs* and one aggregate answer is provided at each epoch. As in TAG, nodes in the Rings topology are loosely synchronized [MFHH02]. An aggregation is collected over an epoch. Each ring is allotted specific time slots within the epoch and all the nodes in that ring use these specific slots for receiving messages, sensing/processing, and transmitting messages. The duration of the allotted slot in an epoch is determined a priori based on the density of deployment (so that even if the sensors perform carrier sensing, all the sensors get enough time to transmit their messages once). The slots are assigned such that the receiving slot of ring i contains the transmission slot of ring $(i + 1)$ (receiving slot is slightly bigger than transmission slot to overcome limitations in the accuracy of time synchronization between nodes of adjacent rings). Figure 4.2 shows the process. Such a loose synchronization allows nodes to turn off their radio and processor for most of the epoch. It also enables multiple aggregation rounds to be pipelined (e.g., ring 5 can start a new epoch while ring 1 is still processing the previous epoch).

We now describe the query aggregation phase in greater detail, using the example Rings topology in Figure 4.3 for illustration. In this example, node q is in R_0, there are five nodes in R_1 (including one node that fails during the aggregation phase), and there are four nodes in R_2. At the beginning of each epoch, each node in the outermost ring (R_2 in the figure) generates its local synopsis $s = SG(r)$, where r is the sensor reading relevant to the query answer, and *broadcasts* it. A node in ring R_i wakes up at its allotted time, generates its local synopsis $s := SG(\cdot)$, and receives synopses from all nodes within transmission range in ring R_{i+1}[2]. Upon receiving a synopsis s', it updates its local synopsis as $s := SF(s, s')$. At the end of its allotted time (by when it is done with

[2]Note that there is no one-to-one (or even static) relationship between the nodes in ring R_i and those in ring R_{i+1} — a node in ring R_i fuses all the synopses it overhears from the nodes in ring R_{i+1}.

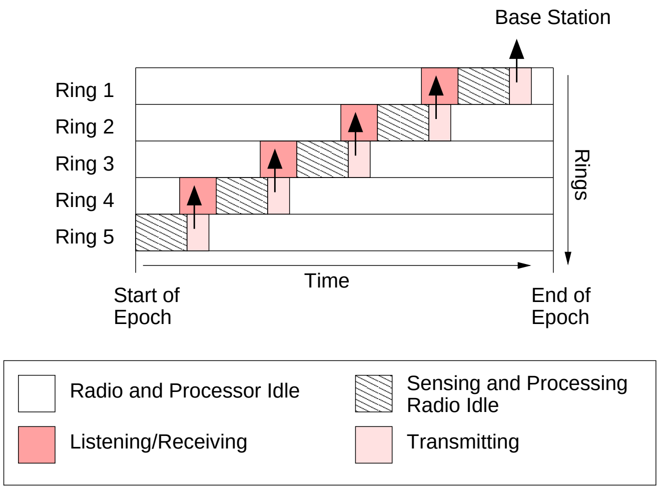


Figure 4.2: Partial aggregates flowing toward the base station during an epoch.

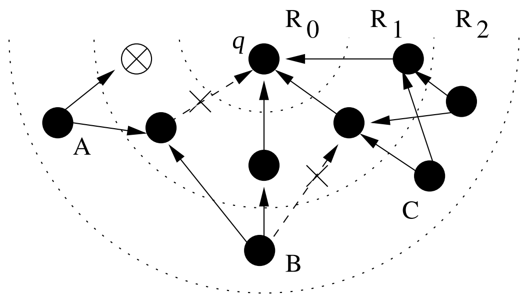


Figure 4.3: Synopsis Diffusion over the Rings topology. Crossed arrows and circles represent failed links and nodes.

fusing the synopses from its neighbors in ring R_{i+1}) within the epoch, the node broadcasts its updated synopsis s. Thus, the fused synopses propagate level-by-level toward the base station q, which at the end of the epoch returns $SE(s)$ as the answer to the aggregate query.

Figure 4.3 shows that even though there are link and node failures, nodes B and C have at least one failure-free propagation path to the base station q. Thus, their sensed values are accounted for in the answer produced this epoch. In contrast, all of the propagation paths from node A failed, so its value is not accounted for.

Because the underlying wireless communication is broadcast, each node transmits exactly once; therefore, *Rings generates the same optimal number of messages as tree-based schemes* (e.g., [MFHH02, MFHH03, MSFC02, ZGE03]). However, because synopses propagate from the sensor nodes to the base station along multiple paths, *Rings is much more robust*. (This added robustness is quantified in Section 4.6.)

4.2.2 More Robust Rings Topologies

Since for the nodes in ring 1 of a Rings topology, there is exactly one node receiving the transmission (the base station), ring 1's transmissions are more susceptible to random transmission losses than other rings' transmissions. To cope with this, we suggest (1) using multiple base stations (in ring 0) who form a mesh and combine the aggregated value at the end of each epoch or (2) making nodes in ring 1 transmit multiple times if the base station has not received a synopsis. The latter approach, although slightly more power consuming, uses the traditional model of having a single base station. We call this scheme Rings2 and will evaluate it in Section 4.6.

Note that Rings2 requires the base station to send acknowledgement messages to the nodes in ring 1 so that the nodes know whether retransmissions are required. In Chapter 5, we will show that Synopsis Diffusion provides an efficient mechanism called *implicit acknowledgement* that can be used in place of explicit acknowledgements. With this mechanism, the base station can simply broadcast the final synopsis at the end of each epoch; all the nodes in ring 1 get the required acknowledgements simply by over-hearing that final synopsis.

4.3 Formal Framework and Theorems

In this section, we present the first formal foundation for duplicate-insensitive aggregation. We define a Synopsis Diffusion algorithm to be "ODI-correct" if and only if its $SG()$ and $SF()$ functions are order- and duplicate-insensitive. Intuitively, these two properties ensure that the final result is independent of the underlying routing topology—the computed aggregate is the same irrespective of the order in which the sensor readings are combined and the number of times they are included during the multi-path routing. We formalize these two requirements later in this section. Next, we prove two key results for ODI synopses. Our first result shows how a Synopsis Diffusion algorithm can be proved to be ODI-correct by proving a few surprisingly simple properties. The second result gives a general framework for analyzing the approximation errors produced by an ODI-correct algorithm. These results are extremely useful for validating Synopsis Diffusion algorithms, and as a guide for designing new Synopsis Diffusion algorithms. We begin by defining, in the next section, some of the key terms used in our formal framework.

4.3.1 Definitions

A *sensor reading* r is a tuple consisting of both one or more sensor measurements and any meta-data associated with the measurements (e.g., timestamp, sensor id, and location). Because of the meta-data, sensor readings are assumed to be unique (e.g., there is only one reading corresponding to a given sensor id and timestamp).

We define a *synopsis label* function $SL()$, which computes the label of a synopsis. The *label* of a synopsis s is defined as the set consisting of all sensor readings contributing to s. More formally, $SL()$ is defined inductively, as follows. There are two cases for $SL(s)$, depending on whether the synopsis s results from an application of $SF()$ or an application of $SG()$:

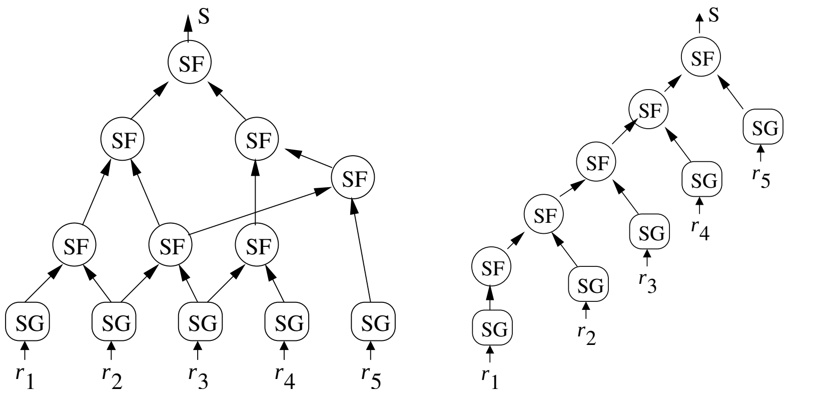
$$SL(s) \quad = \quad \begin{cases} SL(s_1) \uplus SL(s_2) & \text{if } s = SF(s_1, s_2) \\ \{r\} & \text{if } s = SG(r) \end{cases}$$

The operator $\uplus$ takes two multi-sets and returns the multi-set consisting of all the elements in both multi-sets, including any duplicates. For example, $\{a, b, c, c\} \uplus \{b, c, d\} = \{a, b, b, c, c, c, d\}$. Note that $SL()$ is determined by the sensor readings and the applications of $SG()$ and $SF()$—it is independent of the particulars of $SG()$ and $SF()$. Note also that a synopsis label is a virtual concept, used only for reasoning about the correctness of $SG()$ and $SF()$ functions: $SL()$ is *not* executed by the sensor network.

The notion of what constitutes a "duplicate" may vary from query to query, e.g., a query computing the number of sensors with temperature above 50°F considers two readings from the same sensor as duplicates, whereas a query for the number of distinct temperature readings considers any two readings with the same temperature as duplicates. For a given query q, we define a *projection operator*

$$\Pi_q : \textit{multi-set of sensor readings} \mapsto \textit{set of values}$$

that converts a multi-set of sensor readings (tuples) to its corresponding set of subtuples (called "values") by selecting some set of the attributes in a tuple (the same set for all tuples), discarding all other attributes from each tuple, and then removing any duplicates in the resulting multi-set of subtuples. The set of selected attributes must be such that two readings are considered duplicates for the query q if and only if their values are the same. For example, for a query computing the number of distinct temperature readings, the value for a sensor reading is its temperature measurement. For a query computing the average temperature, the value of a sensor reading is its (temperature measurement,



(a) Aggregation DAG (b) Canonical left-deep tree

Figure 4.4: Equivalent graphs under ODI-correctness.

sensor id) pair.

4.3.2 ODI-Correctness

Although we have given an intuitive definition of an ODI algorithm before, we now formally define what it means to be ODI. Let $\mathcal{R}$ be the universe of valid sensor readings. Consider a $SG()$ function, a $SF()$ function, and a projection operator Π_q; these define a universe, $\mathcal{S}$, of valid synopses over the readings in $\mathcal{R}$. We assume that $SF()$ is a deterministic function of its inputs. The formal definition of the properties we seek is:

- A Synopsis Diffusion algorithm is **ODI-correct** if $SF()$ and $SG()$ are *order- and duplicate-insensitive* functions, i.e., they satisfy: $\forall s \in \mathcal{S} : s = SG^*(V)$, where $V = \Pi_q(SL(s)) = \{v_1, \ldots, v_k\}$ and $SG^*()$ is defined inductively as

$$SG^*(V) = \begin{cases} SF(SG^*(V - \{v_k\}), SG(r_k)) & \text{if } |V| = k > 1 \\ SG(r_1) & \text{if } |V| = 1 \end{cases}$$

  where $\Pi_q(\{r_i\}) = v_i$.

Figure 4.4 helps illustrate ODI-correctness. We can represent the $SG()$ and $SF()$ functions performed to compute a single aggregation result using an *aggregation DAG*, as shown in Figure 4.4(a). There is a node for each of the different instantiations of the functions $SG()$ (which form the leaf nodes) and $SF()$ (which form the non-leaf nodes). There is an edge $e : f_1 \rightarrow f_2$ iff the output of the function f_1 is an input to the function f_2. Thus, all internal nodes have two incoming edges and 0 or more outgoing edges. Corresponding to the aggregation DAG, ODI-correctness defines a canonical left-deep tree (Figure 4.4(b)). The leaf nodes are the functions $SG()$ on the *distinct* values (in this example, all readings result in distinct values), and the non-leaf nodes are the functions $SF()$. A Synopsis Diffusion algorithm is ODI-correct if for *any* aggregation DAG, the resulting synopsis is *identical* to the synopsis s produced by the canonical left-deep tree.

More simply, regardless of how $SG()$ and $SF()$ are applied (i.e., regardless of the redundancy arising from multi-path routing), the resulting synopsis is the same as when each distinct value is accounted for only once in s. We chose a left-deep tree for our canonical representation because it lends itself to an important connection with traditional data streams (as discussed in Section 4.5).

A Simple Test for ODI-Correctness

We believe that ODI-correctness captures the overall goal of order- and duplicate-insensitivity. However, it is not immediately useful for designing Synopsis Diffusion algorithms because verifying correctness using this definition would entail considering the *unbounded* number of ways that $SG()$ and $SF()$ can be applied to a set of sensor readings and comparing each against the synopsis produced by the canonical tree.

Thus, a very important contribution of our work is in deriving the following simple test for ODI-correctness. There are four properties to check to complete the test.

- Property P1: $SG()$ **preserves duplicates:** $\forall r_1, r_2 \in \mathcal{R} : \Pi_q(\{r_1\}) = \Pi_q(\{r_2\})$ implies $SG(r_1) = SG(r_2)$. That is, if two readings are considered duplicates (by Π_q) then the same synopsis is generated.

- Property P2: $SF()$ **is commutative:** $\forall s_1, s_2 \in \mathcal{S} : SF(s_1, s_2) = SF(s_2, s_1)$.

- Property P3: $SF()$ **is associative:** $\forall s_1, s_2, s_3 \in \mathcal{S} : SF(s_1, SF(s_2, s_3)) = SF(SF(s_1, s_2), s_3)$.

- Property P4: $SF()$ **is same-synopsis idempotent:** $\forall s \in \mathcal{S} : SF(s, s) = s$.

Note that property P4 is much weaker than the duplicate-insensitivity property required for ODI-correctness. It only refers to what happens when $SF()$ is applied to the *exact same synopsis* for both its arguments. It says nothing about what happens when $SF()$ is applied to differing arguments that come from overlapping sets of sensor readings.[3]

[3] For example, consider the $SF()$ function that takes two numbers x and y and returns their average. This satisfies property P4, because the average of x and x equals x. However, the function cannot be used to compute a duplicate-insensitive average of all the sensor readings. For example, if the readings are 2, 4, and 36, we have SF(SF(2,4),SF(2,36)) = 11 but SF(SF(2,36),SF(4,36)) = 19.5 (and the exact average is 14).

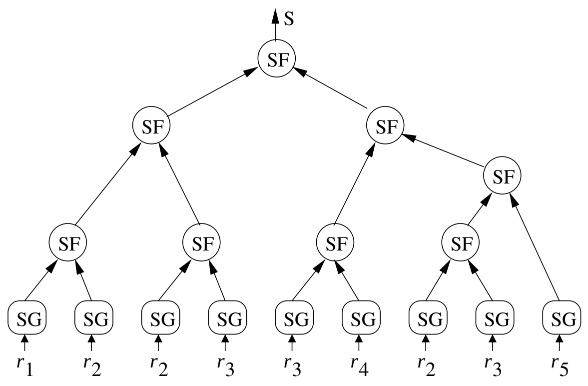


Figure 4.5: Graph used in the proof of Theorem 4.1.

Given the simplicity of properties P1–P4, it is surprising that they characterize ODI-correctness. The next theorem shows that indeed this is the case.

Theorem 4.1 *Properties P1–P4 are necessary and sufficient properties for ODI-correctness.*

Proof of Theorem 4.1. (sketch) Consider an arbitrary execution of Synopsis Diffusion, producing a synopsis s. Let G be the aggregation DAG corresponding to this execution (Figure 4.4(a)), and let u be the node in G that outputs s. In this proof, we will perform a series of transformations to G that, by properties P1–P4, will not change the output of u, and yet will result in the canonical left-deep tree (Figure 4.4(b)).

First, let G_1 (Figure 4.5) be the tree rooted at u corresponding to G, resulting from replacing each node in G with outdegree $k > 1$ with k nodes of outdegree 1, replicating the entire subgraph under the original node for each of the k nodes. This may create many duplicate SF and SG nodes. Also, any node in G without a path to x is discarded (it did not affect the computation of s). G_1 corresponds to a valid execution because SF is deterministic (so applying it in independent nodes results in the same output, given the same inputs), and likewise $SG(r) = SG(r)$ is a special case of property P1. Note that there is exactly one leaf in G_1 for each tuple in the synopsis label $SL(s)$.

Second, by properties P2 and P3, we can reorganize G_1 into an equivalent tree G_2 where the leaves of G_2 are sorted by $\Pi_q(\{r\})$ values: leaf $SG(r_i)$ precedes leaf $SG(r_j)$ only if $\Pi_q(\{r_i\}) \leq \Pi_q(\{r_j\})$.

Third, for each pair of adjacent leaves $SG(r_i), SG(r_j)$ such that $\Pi_q(\{r_i\}) = \Pi_q(\{r_j\})$, we can reorganize G_2 (by applying P2 and P3) such that they are the two inputs to an SF node. By property P1, both inputs are the same synopsis s', so by property P4, this SF node outputs s'. Replace the three nodes (the SF node and its two leaf children) with one of the leaf nodes (say the left one). Repeat until all adjacent leaf nodes are such that $\Pi_q(\{r_i\}) < \Pi_q(\{r_j\})$. Call this G_3. Note that there is exactly one leaf in G_3 for each value in $\Pi_q(SL(s))$.

Finally, reorganize the tree G_3 using P2 and P3 into a left-deep tree G_4 (Figure 4.4(b)); this is precisely the canonical binary tree. In particular, there is exactly one leaf node in G_4 for each value in $V = \Pi_q(SL(s))$, and the left-deep tree corresponds to the definition of $SG^*(V)$. Since performing the SG and SF functions as indicated by G_4 produces the original output s (i.e., the transformations have not changed the output), the algorithm is ODI-correct.

It is not difficult to show that each of the properties is necessary by considering aggregation DAGs with at most four sensor readings. For example, if property P4 were not true, then $SF(SG(r_1), SG(r_1))$ would not produce the synopsis that the canonical tree $SG(r_1)$ does. □

We now illustrate how these properties can be used to prove the ODI-correctness of a Synopsis Diffusion algorithm by revisiting the Count algorithm that estimates the number of sensor nodes in the network.

Claim 4.1 *The Count algorithm in Section 4.1 is ODI-correct.*

Proof. Consider a projection operator Π_q that maps a set of sensor readings to the corresponding sensor IDs. In the Count algorithm, $SF(s, s')$ is the Boolean OR of the bit vectors s and s'. Since Boolean OR is commutative and associative, so is $SF()$. Next, observe that $\Pi_q(\{r_1\}) = \Pi_q(\{r_2\})$ if and only if r_1 and r_2 have the same sensor ID and hence are the same reading. Thus $SG(r_1) = SG(r_2)$. Finally, $SF(s, s)$ is the Boolean OR of the bit vector s with itself, which equals s. Therefore, properties P1–P4 hold, so by Theorem 4.1, the algorithm is ODI-correct. □

Note that the $SE()$ function did not factor into the considerations of ODI-correctness. ODI-correctness only shows that $SE()$ will see the same synopsis as the left-deep tree. The accuracy of the approximate answer, on the other hand, depends on the accuracy of applying $SE()$ to this synopsis. Clever algorithms are still required to get provably good approximations, although the task has been simplified to being able to show (1) the ODI-correctness of $SG()$ and $SF()$, and (2) the accuracy of $SE()$ when applied to synopses from left-deep trees.

4.4 Examples: Duplicate-Sensitive Aggregates

With Synopsis Diffusion, aggregation can be done over arbitrary message routing topologies. *The main challenge of a Synopsis Diffusion algorithm is to support duplicate-*

sensitive aggregates correctly for all possible multi-path propagation schemes. As shown in Section 4.3, to achieve this, we require the target aggregate function (e.g., Count) to be mapped to a set of *order*- and *duplicate-insensitive* (ODI) synopsis generation and fusion functions. Intuitively, such a set of functions ensure that a partial result at a node u is determined by the set of readings from sensor nodes with propagation paths to u, independent of the overlap in these paths and any overlap with redundant paths. No matter in what combination the fusion functions are applied, the result is the same. Thus, a sensor reading is accounted for (exactly once) in the aggregate if there is a propagation path from the sensor node to the base station, and it is never accounted for more than once. We illustrate such functions using the following examples of ODI-correct Synopsis Diffusion algorithms that show the generality of the framework.

4.4.1 Approximate Count

This algorithm counts the approximate total number of sensor nodes in the network. (It can be readily adapted to other counting problems.) The approximation algorithm we present here is adapted from Flajolet and Martin's algorithm (FM) [FM85] for counting distinct elements in a multi-set. It is a well-known algorithm for duplicate-insensitive Approximate Count [BGGMM04, CLKB04, PSP03]. The algorithm uses the following *coin tossing experiment CT(x)*: toss a fair coin until either the first heads occurs or x coin tosses have occurred with no heads, and return the number of coin tosses. Note that $CT()$ simulates the behavior of the exponential hash function that is used in FM:

$$\text{for } i = 1, \ldots, x-1 : CT(x) = i \text{ with probability } 2^{-i} \tag{4.1}$$

The different components of the Synopsis Diffusion algorithm for Count are as follows.

- *Synopsis:* The synopsis is a bit vector of length $k > \log(n)$, where n is an upper bound on the number of sensor nodes in the network.[4]
- *$SG()$:* Output a bit vector s of length k with only the $CT(k)$'th bit set.
- *$SF(s, s')$:* Output the bit-wise Boolean OR of the bit vectors s and s'.
- *$SE(s)$:* If i is the index of the lowest-order bit in s that is still 0, output $2^{i-1}/0.77351$ [FM85].

The function $SE()$ is based on the following intuition. Consider the final synopsis s to which $SE()$ is applied, and the set, V, of sensor nodes contributing to s (i.e., V

[4] The upper bound can be approximated by the total number of sensor nodes deployed initially, or by the size of the sensor-id space.

is the set of nodes with failure-free propagation paths to the base station). From the experiment $CT()$, the probability that the j'th bit of s is set by a given node in V is 2^{-j}. If no node sets the j'th bit, then there are probably less than 2^j nodes in V, and if we have at least 2^j nodes in V, then we expect the jth bit to be set. Thus the number of sensor nodes in V is proportional to 2^{i-1}, where i is the index of the lowest-order bit that is not set by any node in V. Thus $SE(s)$ is proportional to the number of sensor nodes in V. In Section 4.5, we prove that the approximation error guarantees of [FM85] hold for this algorithm. The accuracy of the algorithm can be improved by having each synopsis maintain multiple independent bit-vectors and then taking the average of the indices within $SE()$ [FM85].

4.4.2 Approximate Count Distinct

This algorithm approximately counts the number of distinct items in the network. Flajolet and Martin's algorithm for approximate distinct counting can be readily converted to a Synopsis Diffusion algorithm, as follows. Suppose the elements to count are drawn from the set V. We use a random hash function $H_e : V \mapsto [0, r], r > \log(|V|)$, with an exponential distribution which ensures that half the elements in V are hashed to 0, a quarter to 1, one-eighth to 2, etc.

- *Synopsis:* The synopsis consists of a bit-vector of length $(r + 1)$.
- $SG()$**:** At node u, output a bit vector s of length r with only the $H_e(val_u)$'th bit set, where val_u is the value.
- $SF(s, s')$ *and* $SE(s)$*:* Same as the Approximate Count algorithm.

The intuition as to why this works is the same as for the above Approximate Count algorithm.

4.4.3 Approximate Sum

An approximate sum of (nonnegative integer) sensor readings can be computed using a simple generalization of the approximate Count algorithm: If the sensor node v has the value val_v to contribute to the final answer, it pretends to be a collection of val_v distinct nodes. Specifically, for $SG()$ each node v outputs a bit vector of length k' (where k' is sufficiently large to hold the maximum sum) with the following bits set: for each of val_v times, perform $CT(k')$ and set the returned bit. $SF()$ and $SE()$ are the same as in the Approximate Count algorithm.

However, running $CT()$ for val_v times, as in the above algorithm, may consume a large amount of energy when val_v is large. Instead, we give below an alternative algorithm that avoids this overhead. This algorithm is adapted from a variant of FM that instead of returning $2^{i-1}/0.77351$, where i is the index of the lowest-order 0-bit, returns 2^j, where j is the index of the *highest-order 1-bit* [AMS99]. Because the algorithm keeps track of only the maximum bit set, the synopsis can be smaller. The algorithm assumes that a node can quickly generate a random number between 0 and 1.

- **Synopsis:** Assume that the values we wish to add are integers in the range $[0..X]$. Because the sum can be bounded by nX, where n is an upper bound on the number of nodes in the network, the synopsis is an integer in the range $[1.. \log(nX)]$, i.e., its size is $\log\log(nX)$ bits.

- $SG()$**:** For node v, select a random number x_v in $[0, 1]$ and output $\lceil -\log_2(1 - x_v^{1/val_v})\rceil$.

- $SF(s, s')$**:** Output $\max(s, s')$.

- $SE(s)$**:** Output 2^{s-1}.

The intuition behind the $SG()$ function is as follows. The goal is to mimic the process where for each of val_v times, $CT(k')$ is done and the returned bit is set. The probability that the ith bit will be the maximum bit set after m trials ($m = val_v$ in this case) *equals* the probability that all m trials return the ith bit or less *minus* the probability that all m trials return the (i–1)th bit or less, i.e., $(1-2^{-i})^m - (1-2^{-(i-1)})^m$. To select a maximum bit set according to this probability distribution, $SG()$ selects a random number x and finds the smallest integer $i \geq 1$ such that $x \leq (1 - 2^{-i})^m$. Solving for i, we seek the smallest integer i such that $i \geq -\log_2(1 - x^{1/m})$, namely, $\lceil -\log_2(1 - x^{1/m})\rceil$.

As with Approximate Count, the variance of the approximation can be decreased by maintaining multiple independent synopses and having $SE()$ output $2^{\bar{s}}$, where $\bar{s}$ is the average the indices of the highest-order 1-bits [AMS99].

Considine *et al.* [CLKB04] independently present an energy-efficient Approximate Count algorithm based on the original FM. Their algorithm is somewhat more accurate than ours for the same amount of energy, so we use their algorithm in our Approximate Sum experiments in Section 4.6.

The Approximate Sum algorithm can be further extended to compute Average, Standard Deviation, and Second Moment [CLKB04].

4.4.4 Uniform Sample of sensor readings

Suppose each node u has a value val_u. This algorithm computes a uniform sample of a given size K of the values occurring in all the nodes in the network. Note that traditional sampling algorithms over tree [KRA+03] would not produce a uniform sample in the presence of multi-path routing because they are duplicate-sensitive. However, our ODI synopses produce a uniform sample of the contributing nodes (i.e., the nodes with failure-free propagation paths, regardless of whether they are selected for the sample). The components of the algorithm are as follows:

- *Synopsis:* A sample of size K of tuples. (Initially, it will have fewer than K tuples, until there are at least K nodes contributing to the synopsis.)
- *$SG()$:* At node u, output the tuple (val_u, r_u, id_u), where id_u is the sensor id for node u, and r_u is a uniform random number within the range $[0, 1]$.
- *$SF(s, s')$:* From all the tuples in $s \cup s'$, output the K tuples (val_i, r_i, id_i) with the K largest r_i values. If there are less than K tuples in $s \cup s'$, output them all.
- *$SE(s)$:* Output the set of values val_i in s.

Because the $SG()$ function labels each value with a uniform random number and thus places it in a random position in the global ordering of all the values in the network, selecting the K largest positions results in a uniform sample of the values from contributing nodes. The (duplicate-removing) union operation in SF ensures that the synopsis accounts for a given node's value at most once.[5]

4.4.5 Aggregates computed from Uniform Samples

Many useful holistic aggregates, for which there are no efficient and exact in-network aggregation algorithms, can be approximated from a uniform sample computed using the previous algorithm. For example, given the sensor values $x_1, x_2, \ldots, x_n$, the k-th Statistical Moment $\mu_k = \frac{1}{n}\sum_{i=1}^{n} x_i^k$ (e.g., μ_1 is the Mean) and the k-th percentile value for $0 < k < 100$ (e.g., $k = 50$ is the Median) can be approximated with ϵ additive error[6] and with probability $1 - \delta$ by using a sample of size $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ [BYKS01]. Thus, our random sampling algorithm provides an efficient way to estimate these holistic aggregates.

[5] Note that in practice, id_u need not be included in the synopsis, because equality in the random id r_u can effectively detect duplicates.

[6] For k-th percentile aggregates, the error is with respect to the rank of the value not its magnitude.

4.4.6 Most Popular Items

The goal of this Top-K Frequent Item algorithm is to return the K values that occur the most frequently among all the sensor readings. (When $K = 1$, this is the Mode.) It uses the $CT()$ function used in the Count algorithm described in Section 4.2.

- *Synopsis:* A set of the K most popular items (estimated).
- *$SG()$:* At node u, output the (value, weight) pair $(val_u, CT(k))$, where $k > log(n)$ and n is an upper bound on the total number of items.
- *$SF(s, s')$:* For each distinct value v in $s \cup s'$, discard all but the pair (v, weight) with maximum weight for that value. Then output the K pairs with maximum weight. If there are less than K pairs, output them all.
- *$SE(s)$:* Output the set of values in s.

Essentially, the algorithm determines the frequency of an item by running an ODI-correct Count algorithm for each value. The counting is done using the Alon *et al.* variant [AMS99] of Flajolet-Martin's algorithm (FM), which estimates the number of distinct values by keeping track of the highest-order bit that is set to 1. Maintaining the Count for *every* item would result in very large synopses being exchanged. Instead, our algorithm keeps track of the items generating the highest-order bits (and thus, probabilistically, occurring the most frequently in the network). To reduce the number of false positives and false negatives, each synopsis can contain multiple independent sets of popular items and the function $SE()$ can choose the K items that appear in the most number of sets.

This algorithm can be adapted to approximately answer the iceberg query to find all items occurring above a certain threshold T as follows: $SF()$ retains all the values with $weight \geq log(T)$.

The above algorithm essentially uses sampling to find popular items. It works well with a skewed frequency distribution, i.e., when the popular items appear significantly more frequently than the unpopular items. For situations where items may have non-skewed frequency distribution, we have presented in [MNG05] an ODI algorithm that can efficiently compute items whose frequencies exceed a given threshold.

4.4.7 Union Counting over a Sliding Window

Gibbons *et al.* [GT01] proposed algorithms for union counting over a sliding window; i.e., counting the number of *recent* 1s in the position-wise union of multiple data streams (generated at multiple nodes). One of their algorithms uses a randomized data structure called Randomized Wave that yields an (ϵ, δ)-approximation for union counting in any sliding window up to a prespecified maximum window size. A Randomized Wave consists of multiple *levels* such that level l maintains at most c/ϵ^2 1s with their timestamps, where c is a constant. In the model considered by Gibbons *et al.*, all the nodes independently generate Randomized Waves based on their local data streams. The Waves are then sent to a referee who answers the query. This algorithm can be extended to the Synopsis Diffusion framework as follows.

- *Synopsis:* A Randomized Wave data structure.
- *SG():* Output the Randomized Wave generated by the algorithm in [GT01].
- *SF(s, s'):* For each level l, output the c/ϵ^2 items in $s_l \cup s'_l$ with their earliest timestamps, where s_l denotes the l'th level of the Wave s.
- *SE(s):* Output the result given by the query answering algorithm in [GT01] (referee uses only one final Wave s).

4.4.8 Count-Min Sketch Generation

Cormode *et al.* [CM05] has recently proposed a sublinear space data structure, called Count-Min Sketch, for summarizing data streams. The sketch is a two dimensional count array A, with each row r having a hash function h_r. On arrival of an item i with count c, for each row j of A, $A[j, h_j(i)]$ is incremented by c. Cormode *et al.* has shown that this sketch can be used to answer many queries including point, range, and inner product queries and finding quantiles, frequent items, etc., with good time and space complexity.

Although the Count-Min Sketch has been proposed in the context of a single stream, by replacing the content of each cell with an aforementioned ODI Sum synopsis, it can be extended to be used in the Synopsis Diffusion framework.

- *Synopsis:* A two dimensional array of Sum synopsis.
- *SG():* Generate the Count-Min Sketch as in [CM05]. Then replace each cell of the sketch with the corresponding Sum synopsis.
- *SF(s, s'):* For each cell (x, y), output $SF_{Sum}(s_{x,y}, s'_{x,y})$, where SF_{Sum} is the fusion function for Sum synopses and $s_{x,y}$ is the cell (x, y) of the synopsis s.

- *SE(s):* For each cell (x, y), output $SE_{Sum}(s_{x,y})$, where SE_{Sum} is the evaluation function for Sum synopses. On the resulting two dimensional count array, apply the specific estimation algorithm in [CM05].

Note that the $SG()$ and the $SF()$ functions used by the algorithms presented in this section follow the formal framework in Section 4.3, ensuring that the algorithms are ODI-correct.

4.5 Error Bounds of Approximate Answers

As shown in Section 4.4, many Synopsis Diffusion algorithms provide only approximate answers to certain queries. In fact, there are two distinct sources of errors in the final answers computed by a Synopsis Diffusion algorithm A. The first one is the *communication error*, which is defined as the fraction of sensor readings not accounted for in A's answer in a given epoch (i.e., 1 minus the *percent contributing*). This error is introduced by the underlying routing scheme; it occurs when some of the sensors have no failure-free propagation paths to the base station. The second source of error is the *approximation error*, which is defined as the relative error of the answer computed by A with respect to the answer computed by a corresponding exact algorithm using all the readings accounted for in A's final answer. This error is introduced by the $SG()$, $SF()$, and $SE()$ functions.

We argue that with a sufficiently robust routing scheme, the communication error can be made negligible. We illustrate this using a simple analysis. Suppose the underlying multi-path routing constructs a directed acyclic graph (DAG) G rooted at the base station. We consider a regular DAG of height h where each node at level $i, 1 \leq i \leq h$, has k neighbors at level $(i-1)$ to transmit its synopses toward the base station. For simplicity, assume that level i has d^i nodes, where d is some constant. Also assume for this analysis that message losses occur independently at random with probability p. Then the number of sensor readings N that can reach the base station is given by $N \geq \sum_{i=0}^{h}(1-p^k)^i d^i = \frac{d^{h+1}(1-p^k)^{h+1}-1}{d(1-p^k)-1}$. Thus, the overall communication error is upper bounded by approximately $1-(1-p^k)^h$. To make it more concrete, assume that $p = 0.1, h = 10$. Then, with $k = 1$ (i.e., a tree topology), the error is around 0.65, while it is less than 0.1 and 0.01 for $k = 2$ and $k = 3$, respectively. Hence, by increasing the number of neighbors to transmit synopses toward the base station (i.e., increasing the redundancy of the underlying message routing), through denser sensor deployment if necessary, the communication error can be made insignificant.

Thus, with a robust routing topology, the main source of error in the result computed by a Synopsis Diffusion algorithm is the approximation error. Next, we summarize a generic framework to analyze this approximation error.

Traditionally, the error properties of approximation algorithms are analyzed in a *centralized model* where the algorithms are applied at a central place (e.g., the base station) where all the values are first collected. For example, data stream algorithms [BBD+02] use this model. However, Synopsis Diffusion presents a *distributed model* where the $SG()$ and $SF()$ functions are applied in the distributed set of sensors. The following theorem shows the equivalence of these two models for an ODI-correct Synopsis Diffusion algorithm.

Theorem 4.2 *The answer computed by an ODI-correct Synopsis Diffusion algorithm is the same as that computed by first collecting the values that can reach the base station through at least one failure-free propagation path and then applying the* $SG(), SF()$, *and* $SE()$ *functions on them.*

Proof. (sketch) Consider an arbitrary instance of Synopsis Diffusion aggregation. By ODI-correctness, the corresponding aggregation DAG (Figure 4.4(a)) can be reduced to a canonical left-deep tree (Figure 4.4(b)). This left-deep tree can be viewed as processing a data stream of sensor readings at a centralized place: to each new stream value, we first apply SG and then apply SF with the current stream synopsis. □

Hence, the final result computed by a Synopsis Diffusion algorithm has the following semantics: (1) the final answer includes all the values that can reach the base station through at least one failure-free propagation path, and (2) the result is the same as that found by applying the function SE on the output of a centralized data stream algorithm using SG and SF as indicated above.

Theorem 4.2 shows that *any approximation error guarantees provided for the well-studied centralized data stream scenario immediately apply to a Synopsis Diffusion algorithm, as long as the data stream synopsis is ODI-correct*. Thus, we can effectively leverage existing data stream error analysis, as illustrated in the following claim.

Claim 4.2 *The Count algorithm in Section 4.4 has the same approximation error guarantees as Flajolet-Martin's (FM) distinct count algorithm [FM85].*

Proof: Consider an instance of the Approximate Count algorithm and a corresponding canonical left-deep tree where the leaf nodes from left to right match the order in which values occur in a particular stream the FM algorithm is applied on. By Theorem 4.2, this is equivalent to applying the $SG()$ and $SF()$ functions on all the data at a central place and then applying the $SE()$ function on the final synopsis. The successive applications of $SG()$ (i.e., generating a bit vector) and $SF()$ (taking Boolean OR of the bit vectors) are equivalent to starting with an empty bit vector V, generating the indices, and setting the corresponding bits in V. This is precisely the same way the FM algorithm generates the synopsis. Finally, both the algorithms use the same function to evaluate the resulting synopsis. Hence, they generate the same answer.□

4.6 Evaluation

In this section, we evaluate the Synopsis Diffusion scheme and compare it with the existing schemes.

4.6.1 Methodology

Aggregation Schemes

We evaluate five different aggregation schemes:

TAG The standard tree-based TAG scheme (Section 2.1.2 in Chapter 2).

TAG2 The TAG scheme with value-splitting among two parents (Section 2.1.3 in Chapter 2).

Rings The Synopsis Diffusion (SD) algorithm over the Rings topology (Section 4.2.1).

Rings2 SD with the nodes in ring 1 transmitting twice (Section 4.2.1).

Flood SD over the flood topology that relies on each node attempting to flood its synopsis to all other nodes in the network. This flood is similar to the route discovery used in many ad hoc wireless network routing protocols (e.g., [JM96]). Like the route discovery, the objective of the flood is to discover all possible paths between the sensor node and the base station. Each node starts with a synopsis of its own sensor reading. In each round, a node broadcasts its current synopsis to all neighbors and receives messages from all its neighbors. At the end of the round, a node merges all of its received synopsis with its own current synopsis to generate its new synopsis. To ensure that all nodes contribute to the synopsis at the base station, the Flood algorithm has $D + 1$ rounds, where D is the diameter of the network.

Unless otherwise stated, we compute the Average of the sensor data. We assume uniform data, i.e., the reading of each sensor is its unique ID.

Topology

To evaluate the performance of the above schemes, we implemented them within the TAG simulator used in [MFHH02]. In our simulations unless otherwise noted, we collect a Sum aggregate on a deployment of 600 sensors placed randomly in a $20ft \times 20ft$ area. We place the base station at the center of the grid. Sensors report their node-ids, which are assigned sequentially from 1 to 600, as their sensor readings. In each simulation, we collect results over 500 epochs – we collect a single aggregate value each epoch. We begin data collection only after the underlying aggregation topology, for both Synopsis Diffusion and TAG, are stable.

Message size

We use 48-bytes messages as used by the TinyDB systems. Each Sum synopsis bit-vector uses 32 bits. However, in transmitting multiple bit-vectors, we reduce the size of the synopsis by interleaving the bit-vectors and applying run-length encoding [PGF02]. In our experiments for computing Sum, we use 20 32-bit synopses that when compressed, takes around 14 bytes on average. Two sets of Sum synopses (or one set of Average synopses that computes both the Sum and the Count) fit in a single TinyDB packet along with headers and extra room to handle the variation in the compression ratio.

Transmission model

The TAG simulator supports a realistic message loss model based on the wireless network interfaces in the Berkeley MICA motes. This realistic loss model, described in [MFHH02], assigns loss probability of links based on the distance between the transmitter and receiver as follows: the loss probabilities are 0.05, 0.24, 0.4, 0.57, 0.92, and 0.983 within the range 1, 2, 3, 4, 5, and 6 ft respectively, and 1.0 outside the range of 6 ft. Note that these ranges are relative to the deployment area.

Accuracy

To quantify the performance of the schemes, we use the relative root mean square error (RMS)—defined as $\frac{1}{V}\sqrt{\sum_{t=1}^{T}(V_t - V)^2/T}$, where V is the actual value and V_t is the aggregate computed at time t. The closer this value is to zero the closer the aggregate is to the actual value.

Power consumption

To estimate the overhead of the schemes, our goal is to identify the total battery power consumed by the scheme since energy is the critical resource in this environment. There are two main sources of power consumption on the sensor hardware: computation and communication. To enable our code to execute on actual sensor hardware, we have implemented the Synopsis Diffusion algorithm for computing Sum and some other aggregates within the TinyOS and the TinyDB environment. By analyzing the binary code compiled by TinyOS and using the data-sheet of the mote hardware [Atm03], we found that our code uses at most a few hundred additional CPU cycles in comparison to the TAG implementation. This difference was insignificant in both the overall power budget as well as in the relative communication power consumption of the different schemes. Therefore, we choose to simply use the network communication power consumption to compare the performance of the different schemes. We model the communication power consumption according to the real measurement numbers reported in [MFHH03].

4.6.2 Realistic Experiments

In this section, we explore some of the high-level performance differences among our schemes. We use a realistic experimental setup – with a random node placement and a realistic network loss model – to illustrate these differences. To show the effect of aggregation function and the distribution of data, we use the following two scenarios. First, we compute Average of uniformly distributed sensor data mentioned before. Second, we assume skewed data, i.e., the reading of each sensor is inversely proportional to the square of its distance from the base station[7], and compute the Average at the base station.

Figure 4.6 shows how the Average aggregates reported to the base station change over time with different aggregation schemes, for the skewed data set. Both TAG and TAG2 show significantly higher error and variance than the Synopsis Diffusion schemes. This is because, because of losses, only a small fraction of nodes can send their data to the base station. This is shown in the second column of Table 4.1. The third and the forth columns of the table show the average RMS errors of the computed aggregates when the sensor data is uniform (column 3) and when is it skewed (column 4) as in Figure 4.6. At a high level, it shows that both TAG and TAG2 incur large RMS error because only a small fraction of the nodes report to the base station. Since both TAG and TAG2 provide similar average RMS errors, we report only the performance of TAG in the rest

[7]This reading distribution could be the result of intensity readings near any radiation source (*e.g.*, a light bulb)

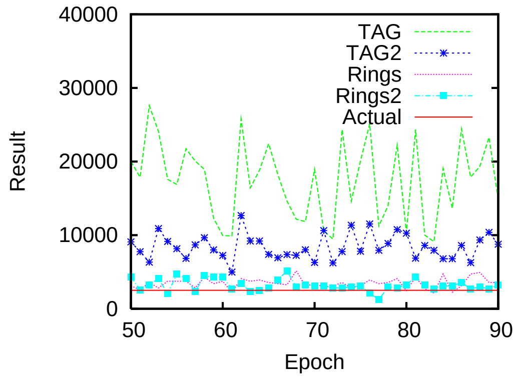


Figure 4.6: The Average aggregate computed in different epochs with different aggregation schemes, for the skewed data set.

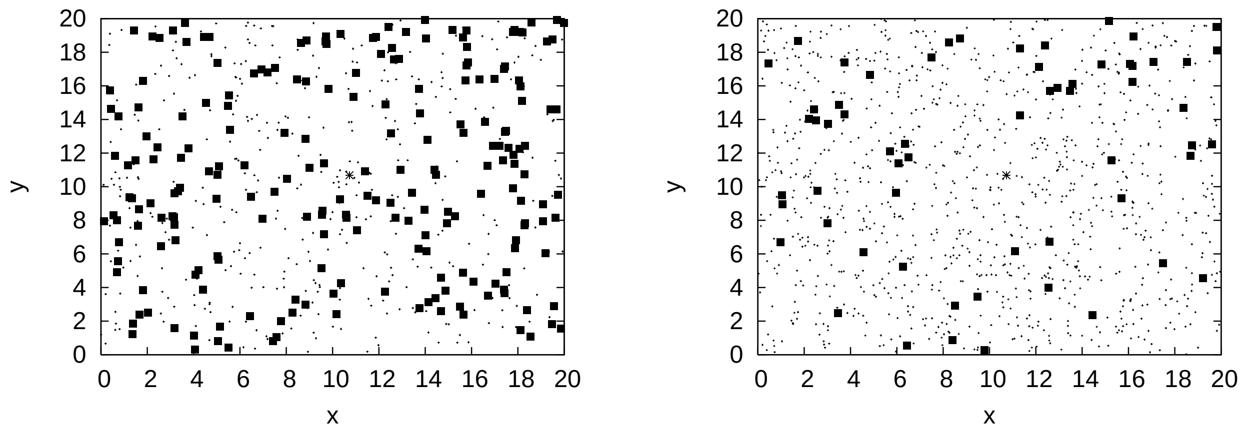
Table 4.1: Robustness of different aggregation schemes computing Average.

Scheme	Average % nodes in a final answer	RMS Error		Energy per node (mJ)
		Uniform Data	Skewed Data	
TAG	< 15%	0.87	0.99	41.8
TAG2	N/A	0.85	0.98	41.8
Rings	65%	0.33	0.19	41.8
Rings2	95%	0.15	0.16	42.1
Flood	≈ 100%	0.13	0.13	780.2

of the experiments. Rings, which is as energy efficient as TAG and TAG2 (shown in column 5 of Table 4.1 and explained in detail in Section 4.6.5), is much more robust than these two. It also shows that the performance of Rings2 is better than Rings and is very close to Flood under this realistic setup. Note that, the errors in the Synopsis Diffusion schemes come mostly from the approximation algorithm.

4.6.3 Effectiveness of Rings2

Figure 4.7 shows the effectiveness of ring-1-nodes' transmitting data twice, with a snapshot (from the base station's point of view) of a single epoch. The solid squares in the figure are the nodes that have failed to send their data to the base station, for all their transmission paths to the base station have failed. With Rings, around 35% nodes fail to contribute to the final answer (Figure 4.7(a)), where as Rings2 reduces that number to around 5% (Figure 4.7(b)). This implies that the robustness bottleneck of around 30% of the nodes is the last hop to the base station, and improving the robustness of this last



(a) Rings (b) Rings2

The star at the center denotes the base station. The solid squares indicate the nodes not included in the final answer. The small dots represent the nodes included in the answer computed in that particular epoch. Rings2 improves the fraction of small dots (i.e., nodes contributing to the final answer).

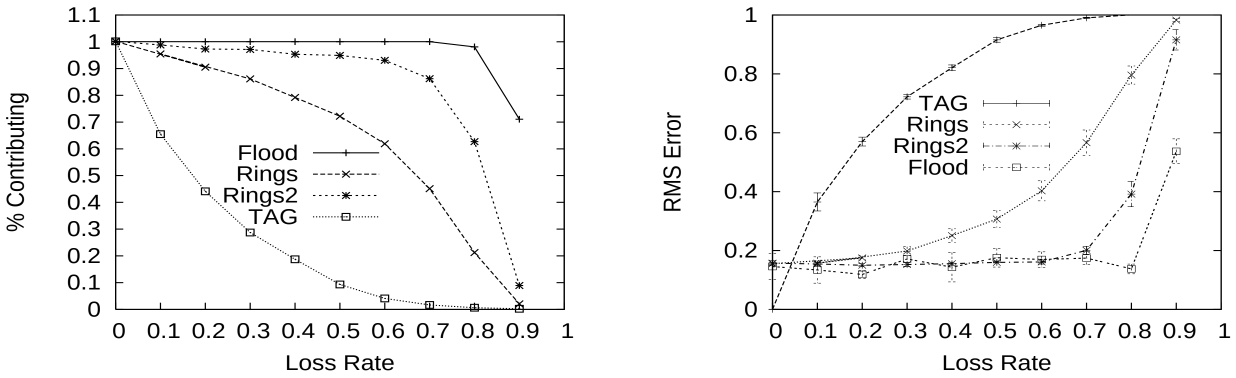
Figure 4.7: Random placement of the sensors in a 20×20 grid and their activity during an epoch.

hop by an extra retransmission is extremely effective in improving robustness.

4.6.4 Effect of Communication Losses

In this set of experiments, we use a simpler loss model in which each packet is dropped with a fixed probability. Figure 4.8(b) shows the impact of changing this loss probability on the accuracy of the different schemes. Even with loss rates as low as 10%, the RMS error for TAG is 0.36, whereas the RMS errors for Rings, Rings2, and Flood are only around 0.15. More importantly, Rings2 perform as well as Flood even when the loss rate is as high as 60%.[8] We also note that the performance of TAG degrades much more quickly with increasing loss rate than any of the Synopsis Diffusion schemes. From Figure 4.8(a), we can see that this degradation is directly related to the fact that the readings of fewer and fewer nodes are incorporated into the reported aggregate. In addition, we can see that the impact of excluding sensor nodes dominates the impact of any approximation errors.

[8]At high loss rate, Flood fails to provide 100% contributing nodes since we allow the flood to run for a limited number of epochs.



(a) % nodes included (b) RMS Error

Figure 4.8: Impact of packet loss on aggregation schemes. The bars in the second graph show the 95% confidence intervals (omitted in the first graph because the intervals are very small).

4.6.5 Effect of Deployment Densities

The placement of sensors can influence the loss rates observed as well as the topology used to aggregate sensor readings. Here, we consider two different variations in the distribution of sensors: the density of sensors and the shape of the sensor deployment region.

To evaluate the impact of sensor density, we vary the number of total sensors while keeping the region (size and shape) in which the sensors are deployed constant. This makes the connectivity graph of the sensors more sparse. In addition, we employ the realistic packet loss model described earlier.

Figure 4.9 shows the impact of changes in density on the accuracy of TAG, Rings, Rings2 and Flood. As the network becomes more sparse, the aggregation schemes are forced to use longer, more error-prone links. This has little impact on Flood, which has a high degree of redundancy in its data collection. Rings and Rings2, having limited redundancy compared to Flood, performs worse with very low sensor density. However, in reasonably dense networks, Rings2 performs as well as Flood due to the large amount of redundancy it can take advantage of. Sparse networks surprisingly also have little impact on TAG. TAG prefers to construct short trees since deep trees combined with packet losses result in very poor performance. As a result, the average parent-child link distance does not change significantly with density. This results in a similar percentage of sensors readings being omitted from the aggregate and, therefore, similar error performance regardless of density.

The added redundancy of Flood and Rings2 comes at a cost in terms of overhead. Figure 4.10 plots the impact of density on our overhead metric, communication power

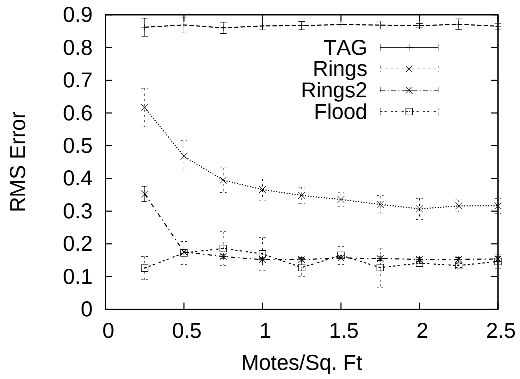


Figure 4.9: The impact of sensor density on accuracy. The bars show the 95% confidence intervals

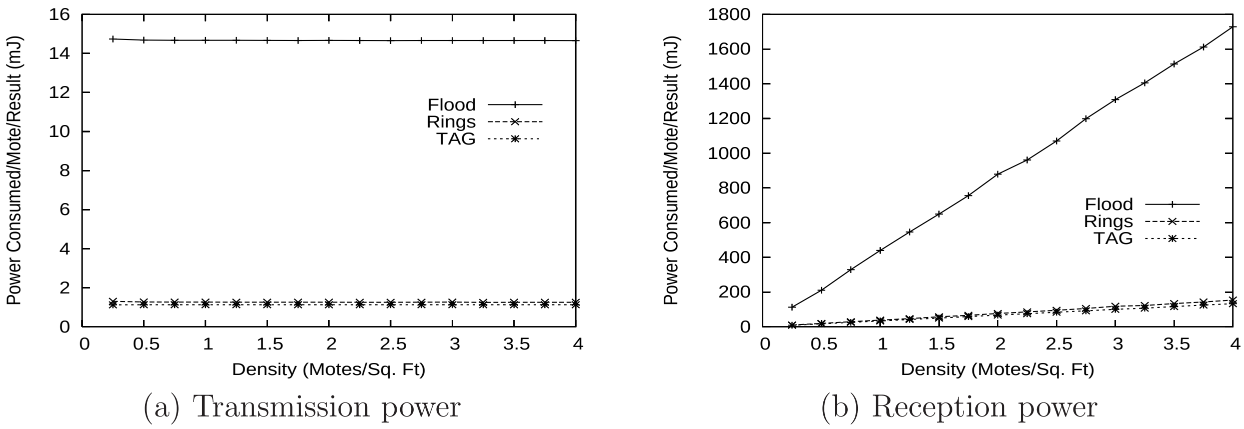


Figure 4.10: Impact of sensor density on power consumption.

consumption. Since the nodes in TAG and Rings remain awake for receiving messages for roughly the same amount of time [MFHH02], and roughly the same number of transmissions occur in both schemes, the nodes' network interfaces in both schemes receive approximately the same number of messages. So, both TAG and Rings have the optimal overhead for transmission power. Rings2 consumes slightly more transmission energy due to the use of redundant transmissions in ring 1 (see Section 4.2.2). Note that, however, the Rings and Rings2 scheme force each node to process all of the received packets, in contrast to a TAG node processing a smaller subset of these messages per epoch. Fortunately, the cost of processing a message is far less than receiving the message. Finally, as expected, Flood has the highest overhead for transmission and reception of the schemes.

In addition to density, the rough shape of a sensor deployment can also affect the performance of the different aggregation schemes. To evaluate this effect, we varied the width of the rectangular deployment area while keeping the height (=20) and the sensor density (2 per square area) constant. Our results show that while the performance of TAG degrades as the diameter of the network increases (i.e., height of the tree increases), that of Rings degrades only slightly.

4.6.6 Effect of Synopsis Size

Synopsis Diffusion provides the opportunity to select a desired approximation accuracy based on the affordable energy overhead (as determined by the message size). For example, in the Approximate Sum algorithm in Section 4.4, a larger synopsis enables additional independent bit-vectors to be used, reducing the approximation error.

To see how the relative error of Synopsis Diffusion changes with the size of the synopsis, we increase the number of bit-vectors in the Sum synopsis (and hence the total number of bits in the compressed synopsis). Figure 4.11 shows the average of the relative errors of the final answer for realistic loss rate and for no loss rate. The x-axis of the graph shows the number of bits of the compressed bit-vectors (we increased the number of bit-vectors by four and reported the length of the compressed synopsis, thus the use of 20 bit-vectors in our other simulations corresponds to the use of around 100 bits). The graph shows that both the average approximation error and the confidence interval can be decreased by using more bits (i.e., more bit-vectors) in the synopsis.

4.6.7 Beyond Sum

We here consider aggregates more complex than the ones considered so far in this section.

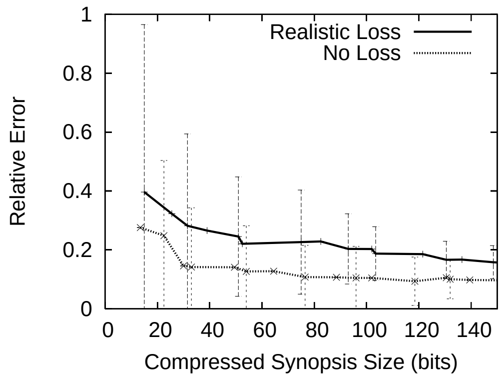


Figure 4.11: Effect of synopsis size in computing Sum. The bars represent 95% **confidence intervals.**

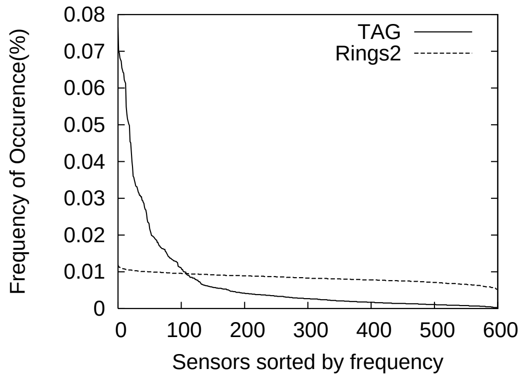


Figure 4.12: Computing uniform sample.

Uniform Sampling

Figure 4.12 compares the sampling algorithm described in Section 4.4 running over Rings2 with the random sampling algorithm known as RanSub [KRA+03] running over TAG. The algorithms compute a sample of size 5, and the graph shows the histograms of the node ids included in 10,000 samples. Note that, RanSub must be run over a tree topology (since its synopsis is not ODI), and it provides a uniform sample when there is no message loss. However, with a realistic loss model, RanSub with TAG provides a distribution far from uniform, while the Synopsis Diffusion algorithm, using Rings2, closely approximates a uniform distribution.

Top-k Values

We have also simulated the Synopsis Diffusion algorithm to find 5 most frequent values in the network where the value of a sensor is the integer part of its distance from the base station (this creates a slightly skewed distribution of the popularity of the data).

We use 10 synopses from which $SE()$ estimates the 5 most popular items. We quantify the accuracy of our estimation $\{x_1, \ldots, x_k\}$ by using the metric *relative rank-error* (RRE) $= \frac{1}{k}\sum_{i=1}^{k}(|i - r_i|)$, where r_i is the actual rank of x_i in the descending order of frequency of all the unique items. With realistic loss model and random placement of the sensors, our algorithm provides very small (≈ 0.6) relative rank-error.

4.6.8 Discussion

Our results have quantified a number of advantages that Synopsis Diffusion provides over tree-based aggregation schemes. First, we have shown how Synopsis Diffusion reduces total errors (due to the combination of estimation error and missing sensor readings) in lossy environments. Second, we have shown that Synopsis Diffusion can achieve these gains without a significant increase in power consumption.

4.7 Related Work

This section describes the related work beyond those mentioned in Chapter 2.

4.7.1 Aggregation over Multi-path

Concurrent to our work, two independent works proposed the idea of computing aggregates over multi-path aggregation topology. The Considine *et al.* paper [CLKB04] is the most closely related work to ours. They independently proposed using duplicate-insensitive sketches for robust aggregation in sensor networks. They presented a very nice technique for extending the Flajolet-Martin distinct counting algorithm to perform duplicate-insensitive sums and averages in a manner suitable for resource-constrained sensor nodes. Moreover, they considered a broadcast-based ring topology of sensor nodes, and showed by simulation the accuracy improvements of rings over the TAG scheme. Our work extends this work in a number of important ways: (1) we present the first formal definition of duplicate-insensitive synopses; (2) we prove powerful theorems characterizing ODI synopses and their error guarantees—their paper has no analogous results; (3) we present solutions for a wider range of aggregates; (4) we consider techniques for adaptive rings that reduce message loss (described in the next chapter); and (5) our simulation results use a more realistic communication loss model, and consider scenarios not addressed in their paper such as mobile sensors and adaptive topologies described in the next chapter.

Bawa *et al.* [BGGMM04] have independently proposed duplicate-insensitive approaches for estimating certain aggregates in peer-to-peer networks. However, the work mainly focuses on the different semantics of the computed aggregates and the required topology and algorithms to achieve that. They do not address the formal requirements of the algorithms; they do not provide the example algorithms we provide; and, they use a peer-to-peer network for evaluation and do not consider many of the sensor-relevant issues addressed by our work.

4.7.2 Query Processing over Data Streams

The use of synopsis in Synopsis Diffusion is related to that in data stream algorithms. There has been a flurry of recent work in the data stream community devising clever synopses to answer aggregate queries on data streams (see [BBD+02, Mut03] for surveys, and [CGMR05, ZKOS05] for some more recent work). There the goals are to estimate an aggregate with one pass through the data and only limited memory. Thus at any point in scanning the data stream, the limited memory data stream synopsis is a small digest suitable for producing a highly-accurate estimate of the stream to that point. The traditional data streams model is i) centralized , i.e., the synopsis is generated at a single place, and ii) order-, but not duplicate-, insensitive. Thus, the model is not adequate for the ODI synopses required for Synopsis Diffusion.

The same synopsis used in traditional data streams can sometimes be used for Synopsis Diffusion. However, Synopsis Diffusion introduces two complications beyond traditional data streams. First, the data is not presented as a sequential stream to a single party. Instead, the data is spread among multiple parties and the aggregation must occur in-network. Specifically, the Synopsis Fusion function merges two synopses, not just a current synopsis with a next stream value. More related then is work on distributed streams algorithms [GT01, GT02]. In the distributed streams model, there are multiple parties, each observing a stream and having limited memory, and the goal is to estimate aggregates over the union of these streams by exchanging synopses at query time. This requires the merging of multiple synopses (ala Synopsis Fusion). Second, Synopsis Diffusion requires duplicate-insensitive synopses. None of the previous work on sequential or distributed data streams was concerned with duplicate sensitivity. (The exception is for aggregates that are by definition duplicate-insensitive, such as Count Distinct.) Only recently, Tao *et al.* [TKC+04] have used duplicate-insensitive counting in mobile environments.

Figure 4.13 diagrams a path for developing new Synopsis Diffusion algorithms, where

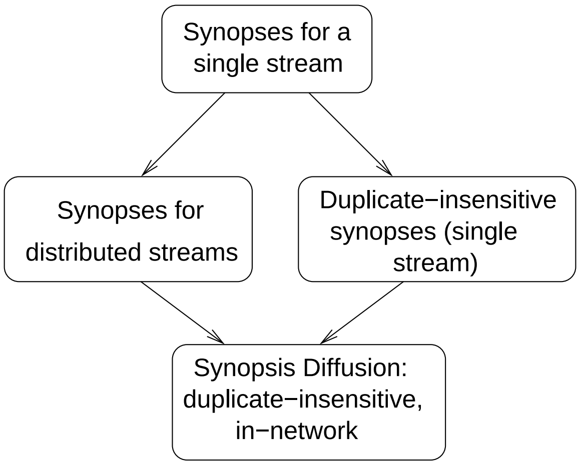


Figure 4.13: Hierarchy among synopsis problems. Each edge is directed from an easier problem to a harder problem.

each edge is directed from an easier problem to a harder problem.

4.8 Summary

In this chapter, we have presented Synopsis Diffusion, a novel framework for robust aggregation in wireless sensor networks. We have presented the formal foundation of this framework, example algorithms, and evaluation results. Our evaluation shows that Synopsis Diffusion can be significantly more robust (upto 5 times) than, but almost as energy efficient as, existing tree-based aggregation schemes.

By using ODI synopsis, Synopsis Diffusion decouples aggregation algorithm from aggregation topology. This provides the unique opportunity to independently optimize these two components. Moreover, it exposes several design tradeoffs related to aggregation algorithm and topology and lets applications choose the appropriate parameters. For example, since accuracy of most Synopsis Diffusion algorithms can be improved by using larger synopsis, applications can tune synopsis size according to target robustness and available energy. Similarly, since there is often tradeoffs between energy-efficiency and robustness of topologies (e.g., more robust topologies such as Flood and Rings2 consume more energy), the application can choose the "right" topology according to its requirements. Moreover, as operating conditions change, the application can adapt aggregation algorithm and topology independently, to ensure the intended quality of service. For example, when the loss rate is low, an energy-efficient tree topology can be used. However, when the loss rate becomes excessively high, sensors can spend additional energy to employ a more robust topology such as Flood—the topology independence of

Synopsis Diffusion will seamlessly guarantee the correctness of the aggregation process. In the next chapter, we will discuss two such adaptation techniques.

Chapter 5

Adaptive Aggregation Schemes

In the last chapter, we proposed Synopsis Diffusion for robust data aggregation in a relatively static, but failure prone, sensor network. However, the operating condition of a long running sensor network is likely to change over time. New nodes may join the network because of redeployments. Over time, nodes can fail—they may run out of energy, overheat in the sun, be carried away by wind or rain, or crash due to software bugs. A node's communication range (and, thus, the topology) can change dramatically due to the vagaries of RF propagation, a result of its strong environmental dependence. To ensure robustness against these changes, it is desirable that an aggregation scheme automatically adapts, ensuring that the live nodes in the sensor network can contribute to the final aggregate answer computed at the base station.

In this chapter, we address this by designing an aggregation scheme that can adapt to the dynamics of the operating environment. Existing adaptive aggregation schemes use low-level observations of network characteristics to repair the tree topology, without taking application semantics into account. We combine the benefit of such local repairs with application-aware adaptation so that the application, based on the quality of the answers computed at the base station, can decide how and when to adapt the aggregation scheme. We argue that such application-aware adaptation is valuable, since the application can exploit the intended data semantics, which is generally weak (e.g., aggregation over 90% of the sensors may be sufficient for the application), and can make certain adaptation decisions better (e.g., if the error in final results is tolerable, it can adapt less often and conserve energy). We achieve this by adapting both the aggregation topology and the aggregation algorithm (existing schemes adapt only the topology). We show that simultaneously running Synopsis Diffusion and existing tree-based schemes in different parts of the network, and adapting the topologies and their proportion based on application's feedback provide significant robustness benefit. We call our scheme *Tributary-Delta*.

Table 5.1: Comparison of the Tributary-Delta scheme with other in-network aggregation schemes.
We assume that all the schemes use unreliable communication. The total energy consumption is given by its two components: number of messages and message size. The total error is given by the sum of the communication error *produced by message losses within the network and the* approximation error *coming from the aggregation algorithm (independent of message loss). Because the message size and approximation error depend on the aggregate, these metrics are shown for two representative aggregates: Count and Frequent Items.*

	Energy Components			**Error Components**			**Latency**
	Message count	**Message size**		**Communication error**	**Approximation error**		
Aggregate:	*any*	*Count*	*Freq.Items*	*any*	*Count*	*Freq.Items*	*any*
Tree (e.g., TAG)	minimal	small	medium	very large	none	small	minimal
Multi-path (e.g., Syn. Diffusion)	minimal	small	large	very small	small	small	minimal
Tributary-Delta (This chapter)	minimal	small	medium	very small	very small	small	minimal

In the rest of the chapter, we first motivate for and provide an overview of the Tributary-Delta scheme. It requires an adaptive multi-path topology that we develop in Section 5.2. We describe the details of Tributary-Delta in Section 5.3 and evaluate it in Section 5.4.

5.1 Tributary-Delta: Motivation and Overview

The motivation for adapting aggregation algorithm comes from the fact that even though multi-path-based aggregation (i.e., Synopsis Diffusion) is generally more robust than tree-based aggregation, the former may not be the best choice under certain common situations. For example,

- For many aggregates, the known energy-efficient multi-path algorithms provide only *approximate* answers (with accuracy guarantees), while tree-based aggregation schemes provide the exact answer. Therefore, when communication loss is insignificant (e.g., when the sensors are deployed in controlled environments), tree-based scheme provides more accurate answers.

- For some aggregates (e.g., the Frequent Item algorithm in [MNG05]), the message

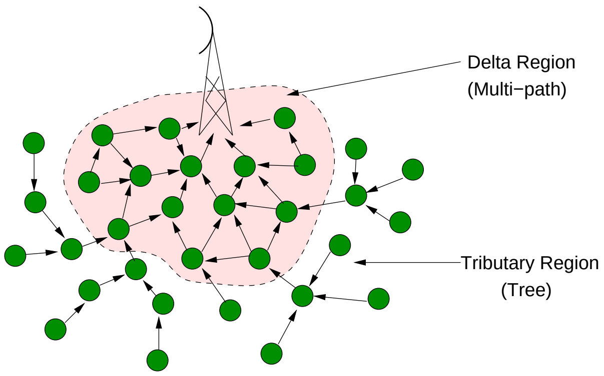


Figure 5.1: A Tributary-Delta Topology.

size in multi-path-based scheme is longer than that in tree-based scheme, thereby consuming more energy.

The first two rows of Table 5.1 provide a qualitative comparison of tree-based and multi-path-based aggregation schemes. For multi-path, we consider the Rings2 topology described in Section 4.2.2. As the table shows, the tree-based scheme suffers from very high communication error while the multi-path-based scheme can have larger message sizes and approximation errors. These tradeoffs lead to the following questions.

- Can we combine the benefits of these two existing techniques into one hybrid aggregation scheme?

- Can we dynamically adapt the hybrid topology to the changes in current loss rates, energy budget, etc.?

We address these two questions with a novel in-network aggregation scheme called *Tributary-Delta*. In regions of the network with low loss rates, trees are used for their low or zero approximation error and their short message size. In regions of the network with higher loss rates or when transmitting partial results accumulated from many sensor readings, multi-path is used for its robustness. Thus, both tree and multi-path may be used simultaneously in different parts of the network. The last row of Table 5.1 summarizes the benefits of Tributary-Delta. We call our scheme *Tributary-Delta* because of the visual analogy to a river flowing to a gulf: when far from the gulf, the merging of river *tributaries* forms a tree-like shape, whereas near the gulf, the river branches out into a multi-path *delta* in order to reach the gulf despite the increased obstacles (see Figure 5.1).

Enabling Tributary-Delta requires four components: adaptive tree topology (e.g., self-repairing tree in TAG), adaptive multi-path topology, tree-based aggregation algorithm (e.g., TAG), and multi-path-based aggregation algorithm (e.g., Synopsis Diffusion). Given that no adaptive multi-path aggregation topology exists, we first develop Adaptive Rings, an efficient adaptive multi-path topology, in the next section.

5.2 Adaptive Rings

Adaptive Rings is an adaptive multi-path aggregation topology. It enhances the Rings2 topology described in the last chapter with the following two additional mechanisms: i) each node efficiently measures the loss rate between itself and its parents, and ii) if the loss rate is high, it chooses a new set of neighbors. For example, a node n with a parent n' monitors if n' fails to forward most of n's transmissions. This may occur if the message transmitted by n is lost before it is received by n', or if n' is dead. In either case, n selects a new parent n''. We describe the required mechanisms in detail in the rest of this section.

5.2.1 Measuring Link Quality with Implicit Acknowledgements

A standard approach to measure link quality is to use acknowledgement or negative acknowledgement messages—a receiver uses these control messages to notify the sender whether the transmission has been successful. However, this introduces overheads in the form of additional energy consumption, longer aggregation latency, and higher channel utilization.

In forwarding data, *ad hoc* wireless networks often address this challenge by using implicit acknowledgements [JM96]. A node n sending to n' snoops the subsequent broadcast from n' to see if n's message was indeed forwarded (and, therefore, was previously received) by n'. However, this does not work with common (tree-based) in-network aggregation schemes. E.g., consider computing Sum with the TAG scheme. If n sends the value x to n', and later overhears n' transmitting some value $z \geq x$, there can be two possibilities: either n' has heard from n and has included x in z, or n' has not heard from n[1] and z is the sum of the values n' has heard from its other children. Thus, n has no way to determine whether transmission through n' is reliable. Fortunately, ODI synopses used by Synopsis Diffusion have a unique property that we can exploit to efficiently implement

[1]Because wireless communication can be asymmetric, n may hear from n' even if n' does not hear from n.

implicit acknowledgements for in-network aggregation.

The Algebraic Structure of an ODI Synopsis

We begin with the following corollary of Theorem 4.1 in Chapter 4.

Corollary 5.1 *Consider an ODI-correct Synopsis Diffusion algorithm with functions $SG()$ and $SF()$. The set $\mathcal{S}$ of synopses generated by $SG()$ together with the binary function $SF()$ forms a* semi-lattice *structure.*

A semi-lattice [DP02] is an algebraic structure with the property that for every two elements in the structure there is an element that is their least upper bound. The function $SF()$ is essentially the *join operator* in lattice terminology; i.e., there is a partial order $\succeq$ on the elements such that:

$$\text{if } z = SF(x, y) \text{ then } z \succeq x \text{ and } z \succeq y \tag{5.1}$$

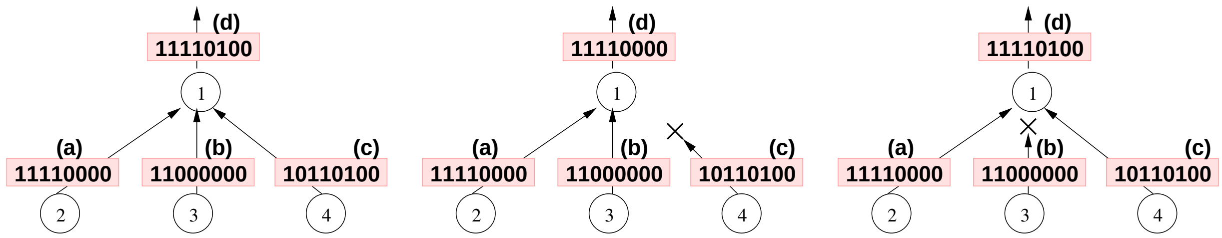
Moreover,

$$\text{if } z = SF(x, y) \text{ then } SF(x, z) = z \text{ and } SF(y, z) = z \tag{5.2}$$

An example of a semi-lattice is the fixed size bit-vectors used in the Count algorithm with the Boolean OR function. The top of the lattice is the all 1's bit-vector, the bottom is the all 0's bit-vector, and for any two bit-vectors x and y, if x OR $y = z$, then x OR z $= z$ and y OR $z = z$. Corollary 5.1 follows immediately from Theorem 4.1 because it is well known that a commutative, associative, idempotent binary function on a set forms a semi-lattice [DP02].

Implication of the Semi-lattice Structure

If a node n running a Synopsis Diffusion algorithm transmits the synopsis x and later overhears some parent node n' transmitting a synopsis z such that $SF(x, z) = z$, it can infer that its synopsis has been *effectively* included into the synopsis z of that parent. For example, in Figure 5.2(a), node 3 can infer that its synopsis is effectively included in the synopsis sent by node 4, since `(c)10110100 OR (d)11110100 = (d)11110100`. Otherwise, it can infer that its message to that parent has been lost. For example, in Figure 5.2(b), node 3 can infer that its synopsis has not reached node 4, since `(c)10110100 OR (d)11110000 = 11110100` $\neq$ `(d)11110000`. Thus, overhearing a synopsis $z = SF(x, z)$ acts as an *implicit acknowledgement* for the node n.



(a) Nodes 2-4 successfully transmits to node 4

(b) Node 4's transmission is lost

(c) Node 3's transmission is lost

In (b), node 4 can infer that its message is lost just by overhearing node 1's transmission.

Figure 5.2: Implicit Acknowledgements provided by an ODI synopsis.

Note that we say that $z = SF(x, z)$ implies x is *effectively* included in z because the condition does not precisely imply that the transmission from n has been received by n'; rather it implies that even if the transmission is lost, the loss has no effect on the final output because it has been compensated by inclusion of the synopses from other children of n'. For example, in Figure 5.2(c), even though the synopsis from node 3 is lost, the loss remains undetected to it by the above Equation 5.2. However, as mentioned, the loss has no effect on the final output, since even if it were received by node 1, the final synopsis `(d)` would not have changed.

Note that, implicit acknowledgement provides an estimation of the outgoing loss rate of a node. This is more effective that the scheme where nodes measure loss rate by counting on incoming messages, since links are often asymmetric and incoming loss rate may be significantly different from outgoing loss rates.

5.2.2 Multi-path Topology Adaptation

We now explain how to use implicit acknowledgement and to modify the Rings2 topology described in Chapter 4 to construct an Adaptive Rings topology. The basic idea is to let each node decide whether it has sufficiently good connectivity with its current parents, and if not, assign itself to a different ring so that it gets a new set of parents. To be assigned to a new ring i, a node needs to update its ring number and wake up during the time slot assigned for the i'th ring in the epoch; since the nodes in ring $(i-1)$ wake up during the adjacent time slot, they readily become its new parents.

The Adaptive Rings topology adapts the ring assignments of the nodes as follows. A node x in the ring i uses implicit acknowledgements to keep track of n_{i-1}, the number of times the transmissions from any node in ring $i-1$ have effectively included x's synopses in last k (an application defined parameter) epochs. When n_{i-1} is below some threshold, x tries to assign itself to a new ring. To do that, it computes n_j, the number of times it overhears the transmissions of any nodes in a nearby ring j for the last k epochs. Since nodes in different rings transmit at different time slots of an epoch, x can compute n_j by listening during the appropriate time slot. The node x in ring i then uses the following heuristics:

1. Assign itself to ring $i+1$ with probability p if (i) $n_i > n_{i-1}$, and (ii) $n_{i+1} > n_{i-1}$ and $n_{i+2} > n_i$.
2. Assign itself to ring $i-1$ with probability p if (i) $n_{i-2} > n_{i-1}$ and (ii) $n_{i-1} > n_{i+1}$ and $n_{i-2} > n_i$.

Intuitively, the heuristics try to assign x to a ring so that it can have a good number of nodes from the neighboring ring to forward its synopses toward the base station at ring 0. For example, consider the first heuristic above. Condition (1) ensures that x will now have parents with better connectivity after switching rings, and condition (2) hints that higher rings have smaller loss rates than lower rings and hence switching to a higher ring is probably good. Although, condition (2) makes the switching decision conservative, our experience shows that it is effective in avoiding repeated switching between rings. The probabilistic nature of the heuristics avoids synchronous ring transition of the nodes and provides better stability of the topology. In our evaluation in Section 5.4.1, we use $k = 10$ and $p = 0.5$.

ODI synopses play two key roles in this adaptation. First, implicit acknowledgement provides n_{i-1}, an estimation of the quality of the existing links and second, it ensures that double counting a value during the adaptation does not hurt.

5.3 Tributary-Delta Details

Running Synopsis Diffusion over Adaptive Rings provides an adaptive multi-path aggregation scheme. We now show how this can be combined with an adaptive tree-based scheme (e.g., TAG over a repairing tree) in order to exploit the tradeoffs mentioned in Table 5.1.

To enable simultaneous use of the tree and multi-path aggregation schemes, we must resolve several challenges. For example, how do the sensor nodes decide whether to use the tree or the multi-path aggregation scheme? How do nodes using different schemes communicate with each other? How do nodes convert partial results when transitioning between schemes? We identify and address these and other challenges in this section, through a careful system design and algorithmic study. We also discuss how a large number of aggregates can be computed within the Tributary-Delta framework.

Although the general framework encompasses optimizing many possible metrics based on the criteria in Table 5.1, we here focus on the following setting. Users provide target thresholds on the communication error (e.g., at least 90% of the nodes should be accounted for in the answer).[2] Our goal is to achieve these thresholds while incurring minimal latency, using a minimal number of messages, and minimizing the message size.

5.3.1 The General Framework

In our Tributary-Delta aggregation scheme, we leverage the synergies between the existing energy-efficient schemes, by combining the efficiency and accuracy of (small) trees under low loss rates with the robustness of multi-path schemes. Specifically, part of the network runs a multi-path scheme while at the same time the rest of the network runs tree schemes. In the extreme, all nodes might either run a multi-path or a tree scheme. We dynamically adjust the use of trees and multi-path, based on current message loss rates. In this section we provide an overview of our Tributary-Delta scheme.

We begin by defining a directed graph G representing the aggregation topology during a Tributary-Delta aggregation. The sensors and the base station form the set of vertices of G, and there is a directed edge for each successful transmission. Each vertex is labelled either $\mathcal{M}$ (for multi-path) or $\mathcal{T}$ (for tree) depending on whether it runs a multi-path aggregation algorithm or a tree aggregation algorithm. An edge is assigned the same label as that of its source vertex. Note that both the set of edges and the labels of individual vertices and edges may change over time. Figure 5.1 depicts an example graph G. Figure 5.3 depicts a portion of another example graph, where T1–T5 are $\mathcal{T}$ vertices and M1–M4 are $\mathcal{M}$ vertices.

There are many ways to construct an aggregation topology with both $\mathcal{M}$ and $\mathcal{T}$ vertices. The basic correctness criteria is that no two $\mathcal{M}$ vertices with partial results representing an overlapping set of sensors are connected to a $\mathcal{T}$ vertex. This is necessary, since otherwise the corresponding $\mathcal{T}$ vertex, whose local aggregation algorithm is duplicate-sensitive, may double-count the same sensor data and provide an incorrect answer. Formally, for every maximal subgraph G' consisting of $\mathcal{M}$ vertices but not the base station, there is exactly one vertex $m \in G'$ directly connected to a $\mathcal{T}$ vertex in $G - G'$

[2] Users can also provide a bound on the approximation error (e.g., the answer should be within 10% of the actual answer of the query applied to the "accounted for" nodes); such bounds can be achieved trivially by adjusting the synopsis size.

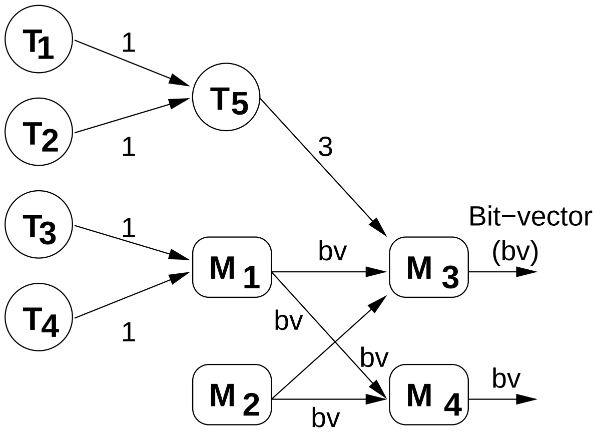


Figure 5.3: Combining Tree and Multi-path algorithms for computing Count in the Tributary-Delta framework. *Bit vector (bv)* is the multi-path synopsis for the Count aggregate (described in Chapter 4).

and every vertex $v \in G'$ has a path to m. Ensuring this requires electing a suitable leader (m) within G'. This general construction, although achievable, thwarts our objectives: it restricts the amount of available redundancy an $\mathcal{M}$ vertex can exploit and the leader election process complicates the aggregation process.

We therefore restrict ourselves to a simpler model where a sensor receiving a partial result from an $\mathcal{M}$ vertex uses a multi-path aggregation scheme. This ensures that a partial result from an $\mathcal{M}$ vertex never reaches a $\mathcal{T}$ vertex downstream toward the base station, and therefore a $\mathcal{T}$ node never gets the chance to double-count sensor data.

In terms of the graph, this correctness condition can be formulated as either an **Edge Correctness** property (Property 5.1) or a **Path Correctness** property (Property 5.2). The two properties are equivalent—both formulations are useful depending on the context.

Property 5.1 Edge Correctness*: An $\mathcal{M}$ edge can never be incident on a $\mathcal{T}$ vertex, i.e., an $\mathcal{M}$ edge is always between two $\mathcal{M}$ vertices.*

Property 5.2 Path Correctness*: In any directed path in G, a $\mathcal{T}$ edge can never appear after an $\mathcal{M}$ edge.*

An implication of *path correctness* is that the $\mathcal{M}$ vertices will form a subgraph (a multi-path "delta") that includes the base station, which is fed by trees of $\mathcal{T}$ vertices ("tributaries"), as depicted in Figure 5.1. Let the *delta region* of G be the set of $\mathcal{M}$ vertices. Coincidentally, any graph G satisfying *path correctness* is also desirable for high accuracy—partial results near the base station account for larger numbers of sensor readings than partial results near the leaves of G, and hence the additional robustness provided by the delta region significantly improves answer accuracy.

Our Tributary-Delta scheme requires multi-path algorithms that can operate on (approximate or exact) partial results from both tree and multi-path schemes. For example, M3 in Figure 5.3 receives inputs from both a $\mathcal{T}$ vertex and two $\mathcal{M}$ vertices. We address this algorithmic challenge in Section 5.3.4.

Dynamic Adaptation

Our goal is to dynamically adapt where in the sensor network we use trees versus where we use multi-path, based on current message loss rates in various regions of the network. However, an implication of *edge correctness* is that individual vertices cannot switch between the two modes independently. We say an $\mathcal{M}$ vertex is *switchable* (to a $\mathcal{T}$ vertex) if all its incoming edges are $\mathcal{T}$ edges or it has no incoming edges. Similarly, a $\mathcal{T}$ vertex is *switchable* if its parent is an $\mathcal{M}$ vertex or it has no parent. In Figure 5.3, vertices T3, T4, T5, M1, and M2 are switchable. Based on these two definitions, we make the following observation.

Observation 5.1 **All** *children of a switchable $\mathcal{M}$ vertex are switchable $\mathcal{T}$ vertices.*

Note that a delta region uniquely defines the set of switchable $\mathcal{M}$ and $\mathcal{T}$ vertices in G. The next lemma implies that by considering only the *switchable* $\mathcal{T}$ and $\mathcal{M}$ vertices, it is always possible to expand (or shrink) the delta region if desired. Let G' be the connected component of G that includes the base station. Then expanding (shrinking) the delta region only makes sense if there is a $\mathcal{T}$ vertex (an $\mathcal{M}$ vertex, respectively) in G'. A simple induction proof yields the following result:

Lemma 5.1 *If the set of $\mathcal{T}$ vertices in G' is not empty, at least one of them is switchable. If the set of $\mathcal{M}$ vertices in G' is not empty, at least one of them is switchable.*

Proof. If the base station is a $\mathcal{T}$ vertex, it is switchable by definition. Otherwise, either there is a $\mathcal{T}$ vertex in the next level (which would be a switchable vertex) or *all* nodes in next level are $\mathcal{M}$. In this latter case, the argument proceeds inductively.

Similarly, if at least one leaf vertex (vertices with no incoming edges) is an $\mathcal{M}$, that vertex is switchable. Otherwise, all leaves are $\mathcal{T}$ vertices and we proceed inductively by considering vertices whose children are all leaves.□

In the next section, we study strategies for adapting the tributary and delta regions to changing network conditions.

5.3.2 Adapting to Network Conditions

We first discuss a number of practical issues that arise in designing our adaptation strategies.

Adaptation Decision

Recall from Section 5.3.1 that the only possible ways to adapt to changing network conditions are (1) to shrink the delta region by switching switchable $\mathcal{M}$ vertices (multi-path nodes) to $\mathcal{T}$ vertices (tree nodes) or (2) to expand the delta region by switching switchable $\mathcal{T}$ vertices (tree nodes) to $\mathcal{M}$ vertices (multi-path nodes). However, because of the different types of errors introduced by the tree and multi-path schemes (recall Table 5.1), it is unclear how switching one or more nodes impacts the answer accuracy. Therefore, we require users to specify a threshold on the minimum percentage of nodes that should contribute to the aggregate answer. It then becomes natural for the base station to be involved in the decision process: depending on the percentage of nodes contributing to the current result, the base station decides whether to shrink or expand the delta region for future results. Because there is only minimal communication error in multi-path schemes (recall Figure 1.5), increasing the delta region always increases the percentage of contributing nodes. Similarly, decreasing the delta region always decreases the percentage of contributing nodes. The system seeks to match the target percentage of contributing nodes, in order to take advantage of the smaller approximation error in tree aggregation. Because this design does not rely on the specifics of any one query, the resulting delta region is effective for a variety of concurrently running queries. Designs specialized to *particular* queries are part of our future work.

Synchronization

A key concern in switching individual nodes from tree aggregation to multi-path aggregation (and vice-versa) is how to ensure that nodes that *should* be communicating after the switch are indeed sending and receiving during the same epoch. When a node switches from $\mathcal{M}$ to $\mathcal{T}$, it needs to change its sending epoch to match its new parent's listening epoch and change its new children's sending epoch to match its listening epoch, etc. Conversely, when a node switches from $\mathcal{T}$ to $\mathcal{M}$, it needs to change its sending epoch to match the listening epoch of its neighboring nodes in the next level and change its children's sending epoch to match its listening epoch, etc. This re-synchronization overhead could arise, for example, if TAG were to be used together with rings for multi-path, and it would be a large deterrent to switching between tree and multi-path schemes. To ensure that no such re-synchronization is necessary, we make a simplifying design choice: a node in level i when switching from $\mathcal{M}$ to $\mathcal{T}$ must choose its tree parent from one of its neighbors in level $i-1$. Similarly, when the node switches from $\mathcal{T}$ to $\mathcal{M}$, it transmits to all its neighbors in level $i-1$, including its parent. In other words, all tree links should be a subset of the links in the ring. This ensures that the switched node can retain its current epoch, since the new parent in level $i-1$ is already synchronized to receive data from the node in level i. Trees constructed with this restriction may have inferior link quality; however, this is mitigated with Tributary-Delta because (1) we use multi-path to overcome poor link quality and (2) the above tree construction algorithm produces bushy trees that are effective in reducing total communication errors (further optimizations for producing bushy trees can be found in [MNG05]).

Oscillation

The base station's desire to match the accuracy of the final result to the user-defined threshold may lead to a repeated expansion and shrinking sequence of the delta region. This can happen when expansion of the delta region improves the accuracy to significantly above the threshold so that the base station decides to shrink the delta region in order to reduce the overhead of multi-path aggregation; but shrinking the delta region reduces the accuracy to below the threshold, and hence it needs to be expanded again. This situation can be common when a large number of nodes simultaneously switch their states. Such an oscillation can unnecessarily increase the adaptation overhead. There are two ways to prevent such oscillations. First, the delta region can be expanded or shrunk at a small granularity, e.g., one node at a time. After a small adjustment of the delta region boundary, the base station can wait to see the result of the adjustment before further adjustment. Second, the base station can use heuristics to damp down the oscillation. For example, if it experiences a repeated sequence of expansion and shrinking, it can simply stop the repetition or can gradually reduce the frequency of adjustments.

5.3.3 Adaptation Strategies

In this section, we present two alternative strategies to shrink and expand the delta region. In both strategies, we augment the messages being sent between nodes with an (approximate) Count of the number of nodes contributing to the partial result being sent. Assuming that the base station knows the number of sensors in the network, it can compute the percentage of nodes contributing to the current result.

Strategy `TD-Coarse`

In the first strategy, `TD-Coarse`, if the percentage of contributing nodes is below the user-specified threshold, the base station expands the delta region by broadcasting a *switching message* asking *all* the current switchable $\mathcal{T}$ nodes to switch to $\mathcal{M}$ nodes. This effectively widens the delta region by one level. Similarly, if the percentage of contributing nodes is well above the threshold, it shrinks the delta region by one level by asking *all* current switchable $\mathcal{M}$ nodes to switch to $\mathcal{T}$ nodes. The coarse-grained control of `TD-Coarse` is well-suited to quickly adapting the size of the delta region to network-wide fluctuations. However, it can not adapt well to different conditions in different parts of the network; for this, we introduce the following more fine-grained strategy.

Strategy `TD`

In the second strategy, `TD`, we use the existence of the parent-child relationship among switchable $\mathcal{M}$ nodes and switchable $\mathcal{T}$ nodes (Observation 5.1), as follows. Each switchable $\mathcal{M}$ node includes in its outgoing messages an additional field that contains the number of nodes in its subtree that did *not* contribute.[3] As the multi-path aggregation is done, the maximum, *max*, and the minimum, *min*, of such numbers are maintained. If the percentage of contributing nodes is below the user-specified threshold, the base station expands the delta region by sending a switching message asking to switch from $\mathcal{T}$ to $\mathcal{M}$ all children of switchable $\mathcal{M}$ nodes belonging to subtrees that have *max* nodes not contributing. In this way, subtrees with the greatest robustness problems are targeted for increased use of multi-path. Shrinking is done by switching each switchable $\mathcal{M}$ node whose subtree has only *min* nodes not contributing. The fine-grained control of `TD` facilitates adapting to non-uniform network conditions, at a cost of higher convergence time and additional switching message overhead because the base station needs to send one message every time it switches a small number of nodes. Note that there are many possible heuristics to improve the adaptivity of `TD`, such as using $max/2$ instead of *max* or maintaining the top-k values instead of just the top-1 value (*max*). Exploration of optimal heuristics is part of our future work.

5.3.4 Computing Aggregates over Tributary-Delta

To compute an aggregate in our Tributary-Delta framework, we need a corresponding tree algorithm, a multi-path algorithm, and a *conversion function* that takes a partial

[3]Note that there is no double-counting here because it follows from the *path correctness* property that the node is the root of a unique subtree.

result generated by the tree algorithm and outputs a synopsis that can be used by the multi-path algorithm. For example, in Figure 5.3, the node M3 receives two multi-path partial results (denoted as bv) and one tree partial result (3). The conversion function needs to transform the tree result to a synopsis so that M3 can use its synopsis fusion function to combine it.

The synopsis generated by the conversion function must be valid over the inputs contributing to the tree result. Fortunately, all the multi-path algorithms described in Chapter 4 have simple conversion functions. For example, the conversion function for the Count aggregate should take the output of the tree scheme–a subtree count c–and generate a (Sum) synopsis that the multi-path scheme equates with the value c. Intuitively, this enables a node running a multi-path algorithm to become oblivious to whether an input synopsis is from a multi-path node or the result of a conversion function applied to a tree result.

Many aggregates (e.g., Count, Sum, Min, Max, Average, Uniform sample, etc.) with known efficient multi-path (described in the previous chapter) and tree algorithms have simple conversion functions,[4] and hence can be efficiently computed in our Tributary-Delta framework. Moreover, the Uniform sample algorithm can be used to compute various other aggregates (e.g., Quantiles, Statistical moments) using the framework. Finally, in [MNG05], we have presented an efficient Tributary-Delta algorithm to identify frequent items in a sensor network.

5.4 Evaluation

In this section, we evaluate Adaptive Rings and Tributary-Delta, implemented within the simulator we used to evaluate Synopsis Diffusion in Chapter 4. We use the same experimental setup (e.g., 600 sensors in a 20×20 area, realistic message level loss rates, etc.) we used in Chapter 4. Unless noted otherwise, in each simulation run, we use Sum (of sensor ids ranging from 1 to 600) as the aggregate collecting an aggregate value every epoch for 100 epochs. We first evaluate Adaptive Rings and then present experimental results showing the effectiveness of our Tributary-Delta scheme.

[4] Since the multi-path algorithms for computing Min, Max, and Uniform sample have *no* approximation error, the tree algorithms can be the same as their multi-path counterparts. For these aggregates then, the "identity function" suffices as the conversion function. For Count and Sum aggregates, the Sum synopsis generation function in Chapter 4 can be used as the conversion function. Lastly, a separate conversion function for Average is not required since Sum and Count can be used to compute Average.

5.4.1 Evaluation of Adaptive Rings

In this evaluation, we compare Adaptive Rings with the schemes TAG, Rings, Rings2, and Flood described in Chapter 4. We seek to understand the effectiveness of Adaptive Rings in coping with node- and link-failures. Note that link-failures can arise because of transient losses, node failures, node mobility, or asymmetric communication. We want to study the effect of all such failures by answering the following questions:

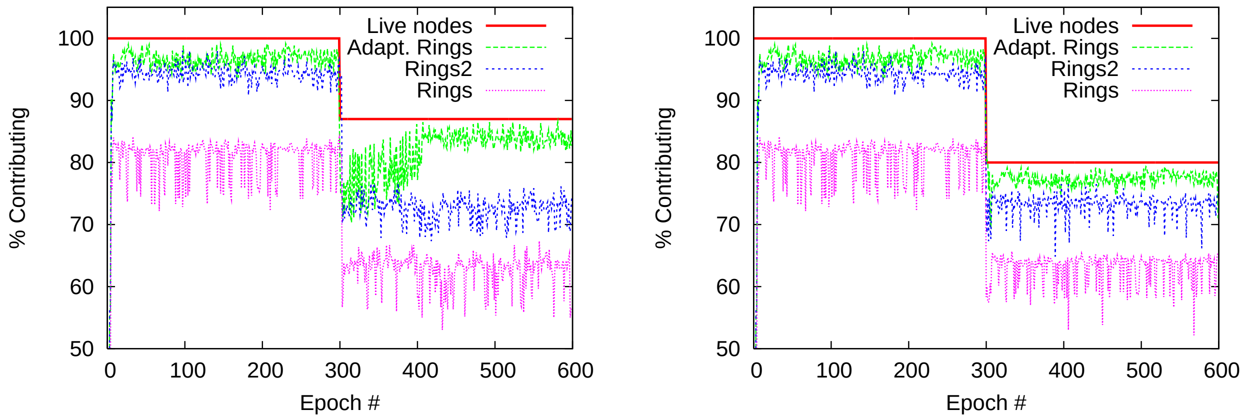
- How well can Adaptive Rings react to node- or link-failures?
- How effective is it in the face of node mobility?
- How effective is it's use of implicit acknowledgement in identifying and avoiding asymmetric links.

Effect of Failures

To show the effectiveness of topology adaptation in the Adaptive Rings scheme, we simulate two scenarios. Figure 5.4(a) shows the first scenario where at time $t = 300$, we kill all the sensors within the rectangular region given by the endpoints $\{(3, 6), (3, 15), (9, 15), (9, 6)\}$ of the 20×20 grid, with the bottom-left corner given by the point $(0, 0)$. This causes a loss of 14% of the total nodes and emulates a geographically correlated set of node failures that might happen, for example, due to a natural disaster.

As the graph shows, Adaptive Rings performs better (with a higher percentage of nodes and lower variance) than the other schemes even when there is no drastic network dynamics (i.e., $t < 300$). Rings2 perform better than Rings showing the effectiveness of the nodes in ring 1 sending twice. Immediately after $t = 300$, all the schemes suffer because the dead sensors break all paths from a significant portion ($\approx 8\%$) of the live sensors to the base station. However, Adaptive Rings gradually adapts its routing around the dead sensors and, thus, lets almost all the live sensors communicate again to the base station. In contrast, in Rings2, 8% of the nodes who could contribute to the computed aggregate before $t = 300$ fail to do so after $t = 300$. The convergence time of Adaptive Rings after $t = 300$ depends on how often the adaptation is done and how long link histories are considered in selecting the ring number for a node. This result shows the contributions of both the ring adaptation and ring 1's retransmissions to the robustness of the Adaptive Rings scheme.

In the second scenario, we kill 20% randomly selected sensors. As Figure 5.4(b) shows, Adaptive Rings effectively copes with the broken multi-path routes and lets almost



(a) Correlated Failures (b) Sparse Failures

In this experiment, we kill a significant number of nodes at time = 300. The Adaptive Rings topology can repair the broken topology and let the live nodes contribute to the final answers. The lines in the graphs are in the order of the corresponding legends.

Figure 5.4: The effectiveness of the Adaptive Rings scheme to cope with node failures.

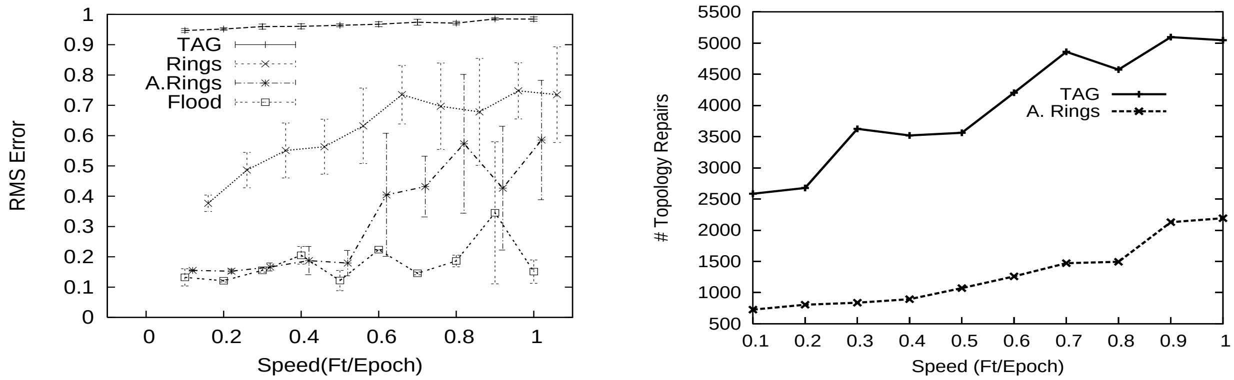
all the live sensors report to the base station. However, both the Rings and Rings2 schemes fail to do so.

Effect of Mobile Sensors

Sensors may be mobile for a number of reasons. They may be deployed on mobile objects (e.g., Robots) [DRS$^+$05], or they may be moved passively by the environment (e.g., by wind or water currents). Mobility can cause a number of challenges, including: 1) the same sensor transmitting its readings from multiple locations (creating duplicate messages), and 2) sensor movement changing the connectivity of the network. Due to Synopsis Diffusion's resilience to losses, duplicate messages and connectivity changes, it is able to handle mobility much more easily than schemes like TAG.

The results to quantify the impact of mobility on these schemes are shown in Figure 5.5(a). We assign the same velocity to all sensors in our simulation – we do vary this velocity between simulations. In addition, each sensor picks a random direction of motion at each time step. Each node checks for possible adaptation on every 4th epoch.

TAG relies on the continued existence of the links that form the aggregation tree, and hence it must repair the aggregation tree whenever sensor mobility removes one of these key links. In TAG, whenever a node is disconnected from its parent, it connects to the



(a) RMS Error (b) Adaptation Overhead

In this experiment, we let individual sensors move. Adaptive Rings can readjust the topology and avoid double-counting, thus providing small RMS errors. The bars in the first graph show 95% *confidence intervals.*

Figure 5.5: The impact of random motion on accuracy and topology adaptation overhead.

next node that it hears from. In addition, to prevent loops, the disconnected node also disconnects from all its children. This essentially forces the entire disconnected subtree to be recreated. As a result, TAG performance degrades with higher rates of mobility as seen in Figure 5.5(a).

The resilience of Synopsis Diffusion to connectivity changes depends closely on the type of propagation used. For example, Flood (described in Section 4.6.1) uses no history of past connectivity to collect results. As a result, changes in connectivity should have little effect on the behavior of the system or its performance. Note that Figure 5.5(a) does indicate some performance degradation. We suspect that this is a result of the diameter of the network changing as a result of mobility – preventing the flood from completing.

The performance of the Adaptive Rings scheme with mobility depends on a number of factors including frequency of the adaptation and the size of the history of the link quality.

The Rings propagation relies on past measurements of hop count distance between a sensor and the base station to construct the propagation schedule. If a sensor moves to a different distance from the base station, the ring topology must be repaired. As expected, Figure 5.5(b) shows that the frequency of these repairs is much lower than the frequency of repairs to the TAG topology. In addition, each repair operation is simple since a node simply assigns itself a ring number one greater than it hears from any of

Table 5.2: Effect of asymmetric links.
In this experiment, we model the links with a realistic asymmetry distribution observed in [ZG03] and a realistic loss rate mentioned in Section 4.6. We here report the average percentage of nodes contributing to the final answer. As shown, asymmetry affects all the schemes, however, Adaptive Rings can adapt itself to avoid asymmetric links. Therefore, the performance of Adaptive Rings degrades only slightly.

Scheme	Percentage of contributing nodes		
	Symmetric links	Asymmetric links	Change
TAG	15%	12%	15%
Rings	65%	60%	10%
Rings2	90%	83%	9%
Adaptive Rings	95%	94%	3%

its reliable neighbors. These factors allow Rings to maintain good performance despite hi-speed mobility.

Effect of Asymmetric Links

Asymmetric links are very common in real sensor network deployments. To see the effect, we model asymmetric links in our simulation, using a realistic asymmetry distribution observed in [ZG03], with a realistic loss rate mentioned in Section 4.6. Table 5.2 shows the results. Due to asymmetric links, the performance of TAG and Rings gets significantly worse (around 15% worse for TAG and 10% worse for Rings) than the case when all links are symmetric. The problem comes from the fact that even if a node x hears from a node y and based on that x selects y as its parent in the aggregation tree, without expensive per sender explicit acknowledgement, there is no guarantee that the transmission of x actually reaches y. However, the implicit acknowledgement of Synopsis Diffusion provides a solution for this problem: the transmission of y tells x whether x's transmission has effectively reached y, and thus the topology can be adapted accordingly. Thus, the performance of Adaptive Rings degrades only slightly ($< 3\%$).

5.4.2 Evaluation of Tributary-Delta

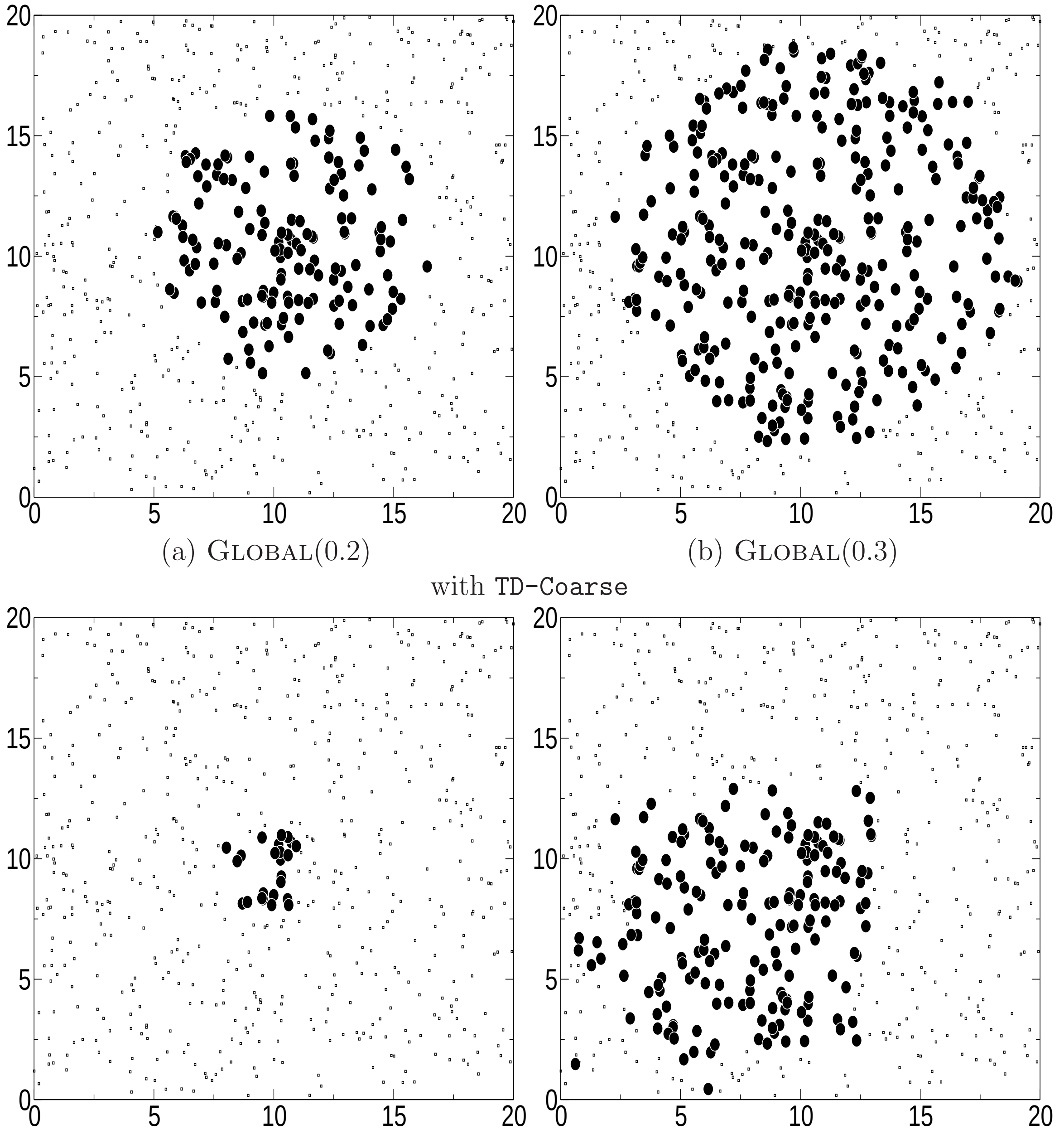
In this section we evaluate our two proposed Tributary-Delta schemes, `TD-Coarse` and `TD`, in varying network conditions, using the pure tree-based scheme `TAG` (TAG with a self-repairing tree) and the pure multi-path-based scheme `SD` (Synopsis Diffusion over Adaptive Rings) as baselines. We first show the different ways in which `TD-Coarse` and `TD` adapt to changes in network conditions. Then we use a simple aggregate (Sum) to report the error reductions due to our proposed schemes over the baseline schemes. Finally, we consider a more complex aggregate (Frequent Items). Recall from Section 5.3 that the adaptivity decisions in our proposed schemes are guided by a threshold on the percentage of nodes contributing to the aggregate. We use 90% as the threshold for our evaluation.

We use LabData, a scenario reconstructing a real deployment, and Synthetic, a synthetic scenario with several failure models, for our experiments. Using actual sensor locations and knowledge of communication loss rates among sensors, LabData simulates a deployment of 54 sensors recording light conditions in the Intel Research Berkeley laboratory [Int05]. The dataset contains around 2.3 million sensor readings. The Synthetic scenario is a deployment of 600 sensors placed randomly in a 20×20 area, with a base station at location $(10, 10)$. We study two failure models for Synthetic: Global(p), in which *all* nodes have a message loss rate of p, and Regional(p_1, p_2), in which all nodes within the rectangular region $\{(0, 0), (10, 10)\}$ of the 20×20 deployment area experience a message loss rate of p_1 while other nodes have a message loss rate of p_2.

Adaptivity of `TD-Coarse` and `TD`

To demonstrate the different ways in which our two strategies adapt to changes in network conditions, we study the Synthetic scenario under the two failure models. First, we apply the Global(p) failure model with increasing values of p. Figure 5.6(a) and Figure 5.6(b) show snapshots of the `TD-Coarse` scheme when the loss rates are 0.2 and 0.3 respectively. As expected, the delta region expands as the loss rate p increases—depicted by an increase in the number of larger dots from Figure 5.6(a) to Figure 5.6(b). The snapshots for the `TD` scheme are similar, except that the delta region increases gradually, instead of expanding by all switchable nodes at a time.

Second, we apply the Regional(p_1, p_2) failure model with increasing p_1 and a fixed $p_2 = 0.05$. In `TD-Coarse`, because the delta region expands uniformly around the base station, all nodes near the base station are switched to multi-path, even those experiencing small message loss. In `TD`, this problem does not arise because the delta region expands only in the direction of the failure region. Figure 5.6(c) and Figure 5.6(d) capture pictorially the response of `TD` to such localized failures. Even at a high loss rate, in `TD`, the delta region mostly consists of nodes actually experiencing high loss rate.



(a) Global(0.2) (b) Global(0.3)

with TD-Coarse

(c) Regional(0.3,0.05) (d) Regional(0.8,0.05)

with TD

Each dot depicts a sensor located at the given coordinates in the deployment area. The larger dots comprise the delta region and smaller dots comprise the tributary region. The base station is at $(10, 10)$.

Figure 5.6: Evolution of the `TD-Coarse` and `TD` topologies for varying loss rates.

Evaluation Using a Simple Aggregate

In this section we evaluate the error reduction of our two proposed schemes, `TD-Coarse` and `TD`, in varying network conditions, using the `TAG` and `SD` schemes as baselines. We restrict ourselves to simple aggregates like Count and Sum for which the partial results can fit in a single TinyDB packet for both `TAG` and `SD`. To ensure that all schemes use comparable energy levels, we disallow retransmissions (as in the original TinyDB implementation).

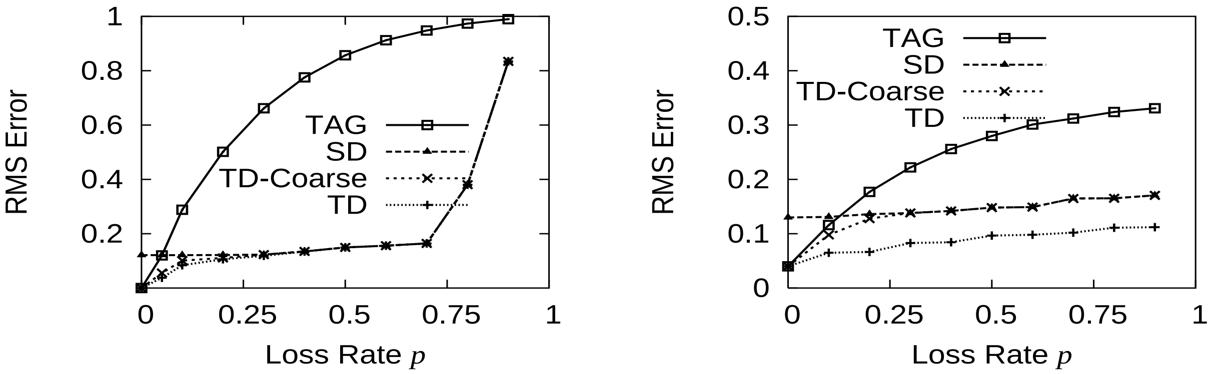
We measure the error as the relative root mean square (RMS) error—defined as $\frac{1}{V}\sqrt{\sum_{t=1}^{T}(V_t - V)^2/T}$, where V is the actual value and V_t is the aggregate computed at time t. The closer this value is to zero the closer the aggregate is to the actual value.

Real scenario. We find the RMS error in evaluating the Sum aggregate on LabData to be 0.5 for `TAG` and 0.12 for `SD`. Both `TD` and `TD-Coarse` are able to reduce the error to 0.1 by running Synopsis Diffusion over most of the nodes.

Synthetic scenarios. For the remainder of this section, we use Synthetic scenarios. Figure 5.7(a), the complete graph for Figure 1.5, presents the RMS error of different schemes under the Global(p) failure model. At *all* loss rates in both cases, the error for either `TD-Coarse` or `TD` is no worse than the minimum of `TAG` or `SD`. In particular, the error is reduced significantly at low loss rates ($0 \leq p \leq 0.05$), when some tree nodes can *directly* provide exact aggregates to the base station. This effect is more pronounced in Figure 5.7(b) with the `TD` strategy under a Regional($p, 0.05$) failure model. `TD` uses multi-path aggregation only in the failure region and so exact aggregation over a significant portion of nodes can be carried out using tree aggregation.

Next, to evaluate how well our Tributary-Delta schemes adapt to dynamic scenarios, starting with the Global(0) failure model, we first introduce Regional($0.3, 0$) at time $t = 100$. Then at $t = 200$, we switch to Global(0.3). Finally, at $t = 300$, we restore the Global(0) failure model. Figure 5.8 shows the relative errors of the answers provided by different schemes over time. We use relative error instead of RMS error because each data point corresponds to just a single aggregate answer.

As expected (Figure 5.8(a)), `TAG` is more accurate when loss rates are low ($t \in [0, 100]$ or $t \in [300, 400]$) whereas `SD` is more accurate when loss rates are high ($t \in [100, 300]$). Figure 5.8(b) and Figure 5.8(c) compare the relative errors of `TD-Coarse` and `TD` with the smallest of the errors given by `TAG` and `SD`. At a high level, both `TD-Coarse` and `TD`, when converged, have at most the error given by any of the two existing schemes. However, the graphs reveal a number of subtle differences between the two Tributary-Delta schemes. First, because `TD` can adjust its delta region at a finer granularity, it can



(a) GLOBAL(p) failure (b) REGIONAL($p, 0.05$) failure

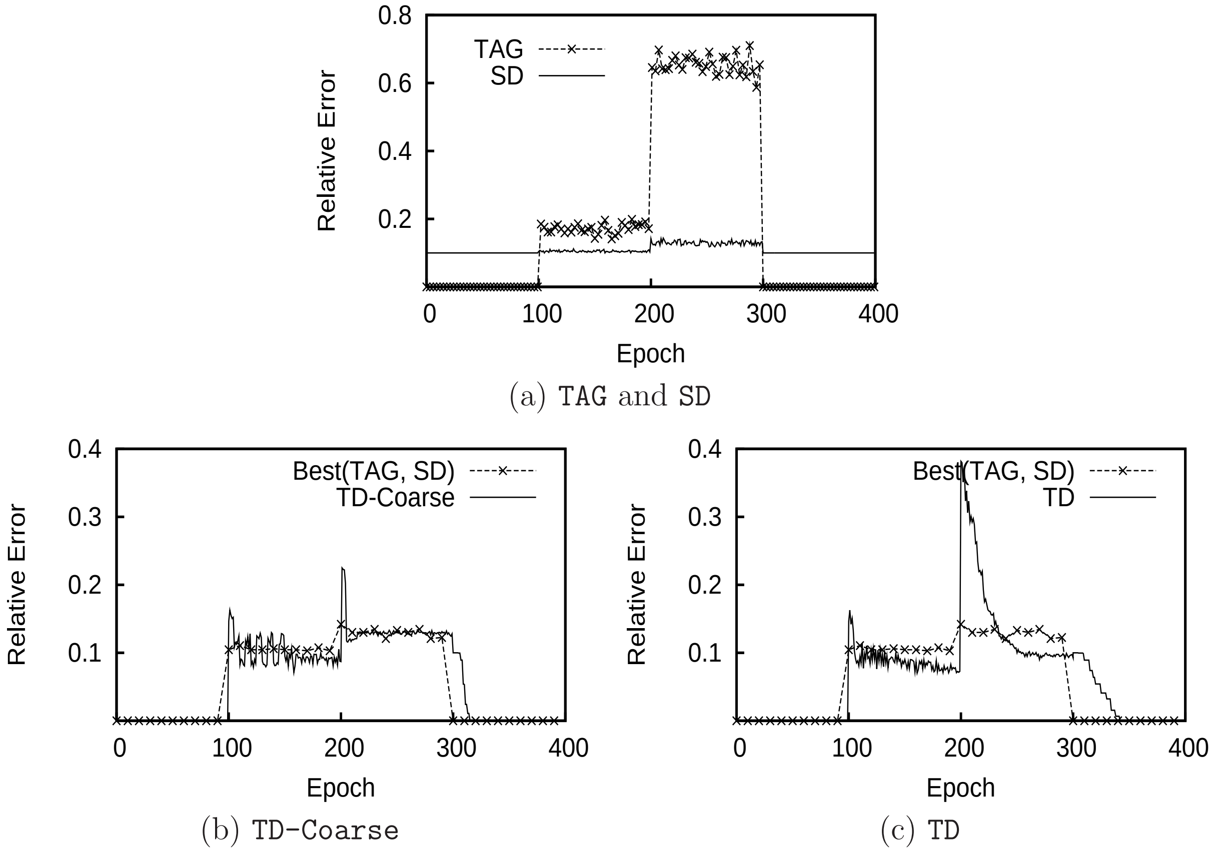
In this set of experiments, we keep the message size, and hence the energy consumption, same for all the schemes. The graph shows that Tributary-Delta improves robustness without additional energy.

Figure 5.7: RMS errors of the Count query under different loss rates

converge to provide a more accurate result. Second, the coarse granularity of `TD-Coarse` adversely affects its convergence: the delta region continues to expand and shrink around the optimal point (e.g., for $t \in [100, 150]$ in Figure 5.8(b)). The base station can use simple heuristics to stop the oscillation (e.g., at $t = 150$), but even then it may end up using a delta region larger than necessary. Finally, the benefits of `TD` come at the cost of a higher convergence time compared to `TD-Coarse`. As shown in Figure 5.8(c), `TD` takes around 50 epochs to converge after the network condition changes. The time can be reduced by carefully choosing some parameters (e.g., how often the topology is adapted), a full exploration of which is part of our future work.

Evaluation Using a More Complex Aggregate

We now consider a more complex aggregate, finding frequent items [MNG05]. The goal is to find all the items that appear more than a given threshold s (expressed as a fraction of the total number of items) in the network. In contrast to the Sum or the Count aggregate studied in Section 5.4.2, a multi-path partial result for the Frequent Items aggregate can consist of more TinyDB messages than a tree partial result (3 times on average in this experiment). Thus the robustness of the multi-path algorithm comes at the cost of additional message overhead. In other words, if we allow both the tree-based and multi-path-based schemes comparable energy to consume, tree can afford retransmission of lost messages. It is not immediately clear how the robustness of multi-path without



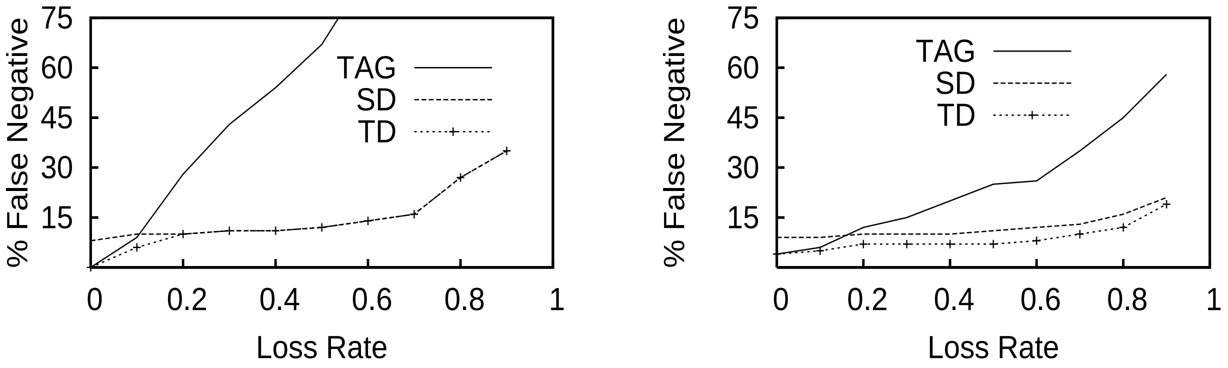
(a) `TAG` and `SD`

(b) `TD-Coarse`

(c) `TD`

We start this set of experiments with the GLOBAL(0) *failure model. We then introduce* REGIONAL(0.3, 0) *at epoch = 100,* GLOBAL(0.3) *at epoch = 200,* GLOBAL(0) *at epoch = 300. Both* `TD-Coarse` *and* `TD` *can adapt to this dynamics.*

Figure 5.8: Timeline showing relative errors of different aggregation schemes.



(a) GLOBAL loss (b) REGIONAL loss

In this experiment, both tree and multi-path schemes avoid retransmission. However, multi-path messages for the Frequent Items algorithm are around 3 times larger than tree messages. Thus, the multi-path scheme uses more energy than the tree scheme.

Figure 5.9: False negatives in the frequent items estimated without retransmission. (False positives are $< 3\%$.)

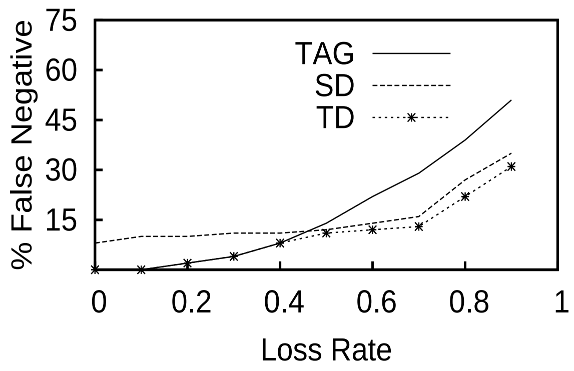
retransmission and tree with retransmission compare.

We compare different schemes in terms of false positives (percentage of rare items reported as frequent items) and false negatives (percentage of frequent items not reported) in finding frequent items from LABDATA. As shown in [MNG05], with no communication failure, tree has no false positives and multi-path has a small ($< 3\%$) false positive rate. Communication failures further reduce the false positive rate, but introduce false negatives in the estimated results because some of the items with frequency above the given threshold $s\%$ are not reported due to under-estimation (mostly in the tree part) resulting from message loss.

Without Retransmission. We first consider tree without retransmission. Figure 5.9(a) and Figure 5.9(b) show the false negative rates of different aggregation schemes under the GLOBAL(p) and REGIONAL($p, 0.05$) failure models, respectively. As in our previous results, `TD` performs as well as (for GLOBAL) or better than (for REGIONAL) the `TAG` or `SD` schemes alone. However, the better robustness of multi-path in this experiment comes at the cost of additional energy to transmit longer partial results.

With Retransmission. Next, to make both tree- and multi-path-based schemes use comparable energy while keeping the latency, which increases linearly with the number of retransmissions, acceptable, we let the tree nodes retransmit their messages twice.[5]

[5]Note that, in practice, two retransmissions would incur more latency than a single transmission of



In this experiment, we permit the tree nodes to retransmit upto 3 times. Multi-path nodes do not retransmit, but since they use longer messages, both tree nodes and multi-path nodes consume a comparable amount of energy.

Figure 5.10: False negatives in the frequent items estimated with Global **loss & retransmission. (False positives are** $< 3\%$**.)**

Keeping the latency low is particularly important for many real-time monitoring and control applications [SMAL+04]. The results are shown in Figure 5.10. As expected, retransmission significantly reduces the false negatives of `TAG`. Still, at loss rates greater than 0.5, the multi-path algorithm outperforms the tree algorithm and `TD` can effectively combine the benefits of both the algorithms.

5.4.3 Discussion

Our evaluation shows that Adaptive Rings is effective in adapting to long term node- and link-failures, node mobility, and asymmetric links. Our results show that Tributary-Delta, that uses Synopsis Diffusion over Adaptive Rings, performs not just as good as TAG or Synopsis Diffusion, but in fact *better* than both these schemes under typical loss rates, without additional energy overhead. We have shown its effectiveness with a simple aggregate where both TAG and Synopsis Diffusion have the same energy overhead, and with a more complex aggregate where TAG can afford retransmissions to match Synopsis Diffusion's larger message overhead.

a 3 times longer message, because each retransmission occurs after waiting for the intended receiver's acknowledgment. Other limitations of retransmission include a reduction in channel capacity (by $\approx 25\%$) and the need for bi-directional communication channels (often not available in practice) [ZG03].

Table 5.3: Properties of different adaptive aggregation schemes. *Adaptive Rings, unlike TAG, adapts its multi-path topology and does not require explicit message acknowledgements. Tributary-Delta, unlike other schemes, adapts both topology and algorithm and uses global adaptation decisions based on application feedback.*

Scheme	*Adapting Component*		*Adaptation Control*		*Message overhead*
	Topology	*Algorithm*	*Decision*	*Mechanism*	
TAG	**Yes** (Tree)	**No**	**Local** (on bad link to parent)	**Local** (parent switching)	ACKs (for link quality estimation)
Synopsis Diffusion + Adaptive Rings	**Yes** (Multi-path)	**No**	**Local** (on bad links to neighbors)	**Local** (ring switching)	None (implicit acknowledgements)
Tributary-Delta	**Yes** (Tree + multi-path)	**Yes**	**Local** (for topology) and **Global** (for algorithm)	**Local** (parents and algorithm switching)	ACKs (for tree link quality estimation) and *Switching messages* (for algorithm switching)

5.5 Summary

In this chapter, we have developed Tributary-Delta, a novel adaptive aggregation scheme that enables application-aware adaptation of aggregation algorithm and aggregation topology. Tributary-Delta combines existing tree-based aggregation schemes with Synopsis Diffusion over Adaptive Rings. Adaptive Rings is an adaptive multi-path topology that we have developed in this chapter. It uses implicit acknowledgements, provided by Synopsis Diffusion, to efficiently decide when and how to repair the topology. We believe that Tributary-Delta is a practical and useful technique to cope with the inherent dynamics of sensor networks.

Table 5.3 shows how different components of existing schemes and our schemes compare. Adaptive Rings is more robust than existing schemes, since it adapts a multi-path topology. Moreover, it is more energy-efficient than them because of its use of implicit acknowledgements. Tributary-Delta is more robust than all other schemes for its careful adaptation of both the aggregation topology and the aggregation algorithm.

There are several ways our Tributary-Delta scheme can be improved. One can imagine a scheme with more distributed control for adjusting the delta region; or a scheme that can have delta regions anywhere in the network, not just around the base station; or a scheme that adjusts the delta region not just based on the percentage of nodes contributing, but on the observed errors in aggregate results. We discuss these future research directions in Chapter 8.

Chapter 6

Subtleties in Tolerating Correlated Failures

In the previous two chapters, we discussed techniques that can be used by the data collection component of a sensing system (e.g., SAs of IrisNet) to achieve robust data collection from wireless sensors. Data collected from different types of sensors are indexed and stored within the storage component of the sensing system (e.g., OAs in IrisNet). In this chapter and the next chapter, we will discuss techniques that significantly improve the availability of the data storage component.

Designing highly available distributed database (or, general storage) systems has long been an active area in systems research. The standard availability techniques include replication, regeneration, and dynamic load balancing. Replication helps in tolerating transient crash-failures, regeneration helps in adapting to long-term crash-failures, and dynamic load balancing avoids failures due to the overload caused by flash-crowd like events. Although each of these components is well studied in the context of many existing systems, we face the following two key challenges in using them within an Internet-scale sensing system:

- Existing distributed databases, unlike the future sensing systems we envision, are not designed to scale to thousands of poorly maintained nodes in the Internet. Although one might think that we could use previously proposed availability techniques in our target sensing system, our experience shows that, surprisingly, these techniques are less effective in real-world deployments than one might hope, often resulting in system designs that are far from optimal. We identify that failure correlation, events when a large number of nodes fail almost simultaneously, is the primary reason behind this. Since large-scale correlated failures are common in today's Internet, the design and evaluation of an Internet-scale sensing system must consider them. For example, in using replication, a sensing system must choose suitable replication parameters (e.g., number of replicas, quorum systems, etc.) required to achieve a target availability despite the correlated failures common in the real world. The challenge is to understand the nature and impact of these correlated failures and to use the findings in system design and evaluation.

- Existing availability techniques have not yet been used in the context of sensing systems. Although, in general, they are not very effective in the presence of correlated failures, sensing systems have many unique properties (e.g., hierarchically scoped queries, tolerance for occasional data inconsistency, absence of write-sharing, easily serializable writes, etc.) that, we believe, can significantly simplify the problems. However, since the area of Internet-scale sensing is still unexplored, we need to find suitable replication, regeneration, and load balancing algorithms that can exploit the above properties, and thereby significantly improve the availability of a sensing system.

We address the first challenge in this chapter and the second challenge in the next chapter. In this chapter, we use a combination of experimental and mathematical analysis of several real-world failure traces to understand the failure properties of large distributed systems. We revisit previously proposed techniques for addressing correlated failures, and debunk four common myths about how to design systems to tolerate such failures. Based on our analysis, we identify a set of design principles that system builders can use to build applications more capable of tolerating correlated failures. In the next chapter, we address the second challenge above—we use the design principles and insights from this chapter to develop efficient techniques that can improve the availability of a sensing system by orders of magnitude. We describe the design, implementation within IrisNet, and evaluation of these techniques in the next chapter.

Note that although the design principles developed in this chapter allow us to exploit unique properties of sensing systems in order to improve their availability, they can be generalized and the insights can be used in a large class of distributed storage systems.

6.1 Myths Debunked: A Preview

In this chapter, we show that previously proposed availability techniques, although plausible, are less effective than one might hope under real-world failure correlation, often resulting in system designs that are far from optimal. Our study also reveals the subtleties that cause this discrepancy between the perception (the myth) and the reality. These new findings lead to four design principles for tolerating correlated failures in distributed storage systems. Specifically, our study reveals the following four myths about tolerating correlated failures, along with the corresponding realities, overlooked subtleties, and design principles.

- **Myth 1: Correlated failures can be avoided using previously proposed failure pattern prediction techniques.** We find that avoiding correlated failures by predicting the failure pattern (as in Oceanstore [WMK02]) provides *negligible* benefits in alleviating the negative effects of correlated failures. The subtle reason is that the top 1% of correlated failures (in terms of size) have a dominant effect on system availability, and their failure patterns seem to be the most difficult to predict. Thus, system designs must not overlook the unpredictability of these large failures.

- **Myth 2: Simple modelling of failure sizes is adequate.** We find that considering only a single (maximum) failure size (as in Glacier [HMD05]) leads to suboptimal system designs. Under the same level of failure correlation, the system configuration as obtained in [HMD05] can be both overly-pessimistic for lower availability targets (thereby wasting resources) and overly-optimistic for higher availability targets (thereby missing the targets). The subtle cause stems from the fact that system availability is determined by the combined effects of failures with different sizes. Even the largest 1% of the failures, which have the dominant effect, still cover a wide range of failure sizes (e.g., 15 to 60 nodes in one of our traces). Such effects cannot be summarized by failures with the *same* sizes, even with scaling factors. On the other hand, we show that using our *bi-exponential* model to capture the failure size *distribution* can help avoid overly-optimistic or overly-pessimistic designs.

- **Myth 3: Additional fragments/replicas are always effective in improving availability.** For popular $(n/2)$-out-of-n encoding schemes (used in OceanStore [KBC+00, WK02] and CFS [Cat03]), as well as majority voting schemes [Tho79] over n replicas, it is well known that increasing n yields an *exponential* decrease in unavailability under independent failures. In contrast, we find that under correlated failures, additional fragments/replicas result in strongly *diminishing* returns in availability improvement for many schemes including the previous two. The diminishing return effects are so strong that even *doubling or tripling* n provides only limited benefits after a certain point. These findings imply that designing systems assuming independent failures and then using overprovisioning (by increasing n) can easily fail to achieve availability targets in practice.

- **Myth 4: Better designs under independent failures remain better under correlated failures.** We find that the better system design (for availability) under independent failures is often not the better design under correlated failures. For example, our results show that while 8-out-of-16 encoding achieves *1.5 more nines* of availability than 1-out-of-4 encoding under independent failures, it achieves *2 fewer nines* of availability under our real traces with correlated failures. The subtle cause is that, with m-out-of-n encoding, the above diminishing return effect is more dramatic for larger m. Thus, system designs must be explicitly evaluated under correlated failures.

Our findings depend unavoidably on the failure traces we used. Among the four myths above, the first myth regarding failure pattern prediction may be the most dependent on the specific traces. The other three findings, on the other hand, are likely to hold as long as failure correlation is non-trivial.

In the rest of the chapter, we describe our analysis methodology and findings in greater details.

6.2 Methodology

Our study is based on mathematical analysis, system implementation, and experimental evaluation. The experiments mostly use a combination of trace and model driven simulation. In Chapter 7, we show that the results from our event driven simulator closely match the results obtained from real deployments of an IrisNet application.

For simplicity, we limit most of our discussion to a simple read-only system, although we show at the end of this chapter that the results naturally extend to read-write systems such as IrisNet (or any generic sensing systems). Use of a read-only system keeps our discussion simpler and provides better intuition behind our findings. We use $\textsc{Erasure}(m, n)$ to denote an m-out-of-n read-only erasure coding system. Also, to unify terminology, we often refer to replicas as fragments of an $\textsc{Erasure}(1, n)$ system. Unless otherwise mentioned, all designs we discuss in this chapter use regeneration to compensate for lost fragments due to node failures.

For the purpose of studying correlated failures, a *failure event* (or simply *failure*) crashes one or more nodes in the system. The number of nodes that crash is called

Table 6.1: Three traces used in our study.

Trace	Duration	Nature of nodes	# of nodes	Probe interval	Probe method
WS_trace [BWWG02]	09/2001-12/2001	Public web servers	130	10 mins	HTTP GET from a CMU machine
PL_trace [Str04]	03/2003-06/2004	PlanetLab nodes	277 on avg	15 to 20 mins	all-pair pings; 10 ping packets per probe
RON_trace [And05]	03/2003-10/2004	RON testbed	30 on avg	1 to 2 mins	all-pair pings; 1 ping packet per probe

the *size* of the failure. To distinguish a failure event from the failures of individual nodes, we explicitly call the latter *node failures*. A data object is *unavailable* if it can not be reconstructed due to node failures. We present availability results using standard "number of nines" terminology (i.e., $\log_{10}(1/\phi)$, where ϕ is the probability the data object is unavailable).

6.2.1 Failure traces

We use three real-world wide-area failure traces (Table 6.1) in our study. WS_trace is intended to be representative of public-access machines that are maintained by different administrative domains, while PL_trace and RON_trace potentially describe the behavior of a centrally administered distributed system that is used mainly for research purposes, as well as for a few long running services.

We call a complete round of all-pair pings or Web server probes as a probe *interval*. As mentioned in the table, the PL_trace and RON_trace traces consist of periodic probes between every pair of nodes. Each probe consists of multiple pings; we declare that a node has *failed* if *none* of the other nodes can ping it during that interval. We do not distinguish between whether the node has failed or has simply been partitioned from all other nodes—in either case it is unavailable to the overall system. The WS_trace trace contains logs of HTTP GET requests from a single source node at CMU to multiple Web servers. Our evaluation of this trace is not as precise because near-source network partitions make it appear as if all the other nodes have failed. To mitigate this effect, we assume our probing node is disconnected from the network if 4 or more consecutive HTTP requests to different servers fail.[1] We then ignore all failures during that probe

[1] This threshold is the smallest to provide a plausible number of near-source partitions. Using a smaller threshold would imply that the client (on Internet2) experiences a near-source partition > 4% of the time in our trace, which is rather unlikely.

Table 6.2: Gap analysis of PL_trace.
We group gaps by the number of nodes affected (i.e., no probe information is available for those nodes during the gap period), and then count the number of gaps in each category. Note that, 7 gaps affect all the nodes, a clear evidence of the failure of the central data collection server. We ignore these gap periods in our study.

# nodes affected by the gap	number of such gaps
all	7
7	1
6	3
4	1
3	6
2	22
1	123

period. Note that this heuristic may still not perfectly classify source and server failures, but we believe that the error is likely to be minimal.

In PL_trace, many large gaps (i.e., periods of time) appear in the trace where probe information is not available for all nodes. We call each such period of time as a *gap*. For those nodes that do not have probe information in a given gap, we say they are *affected* by the gap. These gaps occur in the trace primarily due to two reasons – (1) the affected node is removed from the production list of PlanetLab and added back later to the list, because of some problem with the node; (2) the central server that collects all-pair-ping measurements failed. In our study, we consider gaps due to the first reason as the failures of the affected nodes, since these nodes are removed from the infrastructure and not available for use. For gaps due to the second reason, we assume that the status (i.e, up or down) of the affected nodes do not change during those gaps. To determine which gap is due to which reason, we classify the gaps in the trace by the number of affected nodes in the gaps (Table 6.2). All together we find 163 gaps in the trace. Seven gaps affect all nodes (over 300 nodes). This is a clear evidence of central server failure. On the other hand, all other gaps affect 7 or fewer nodes, which are indications of node removals. Not having gaps affecting more than 7 but less than 300 nodes confirms that the gaps are most likely caused by the previous two reasons (which result in quite different gap behavior).

In PL_trace, the set of nodes in each interval are constantly changing due to node join and leave. The same is true in WS_trace because of the previous filtering step we use, which filters some of the nodes in certain intervals. The change in WS_trace, however, is rather small. In these two traces, when counting the number of failed nodes that were not failed as of the previous interval, the nodes counted must be present in both intervals, so that we can say it transits from up to down. Similarly, when calculating MTTF (mean time to failure), we must first observe one transition from down to up (where the node must be present in two adjacent intervals), and then observe a transition from up to down (similarly, the node must be present in two adjacent intervals). Further, the node must always be in the trace between the two transitions. Only in such a scenario will the time between the two transitions contribute to our calculation of MTTF. The same is true when we calculate MTTR (mean time to repair). This is the same methodology as in previous studies [CV03] of these traces. Finally, due to the limited duration of the traces, and also because failures (especially large correlated failures) are rare events, we do not observe occurrences of all failure sizes. On the other hand, we cannot simply use a probability of zero for those failure sizes, since zero would be negative infinity on log-scale. We thus use the following smoothing technique. Consider a sequence of consecutive failure sizes x, $x+1$, …, $x+k-1$ for which we do not observe any events, but we do observe z failure events with size of $x+k$. Then we treat the z events as z/k failure events for each of the failure sizes of x, $x+1$, …, $x+k-1$, $x+k$. Notice that such processing will only underestimate the strength of correlation, since it treats some of the larger failures as smaller failures. Also, the processing has larger impact on `WS_trace` than on `PL_trace` and `RON_trace`.

In studying correlated failures, each probe interval is considered as a separate failure event whose size is the number of failed nodes that were available during the previous interval.

6.2.2 Limitations

As mentioned in Section 6.1, although the findings from this chapter depend on the traces we use, the effects observed in this study are likely to hold as long as failure correlation is non-trivial. One possible exception is our observation about the difficulty in predicting failure patterns of larger failures. However, we believe that the sources of such large failures (e.g., DDoS attacks) make accurate prediction difficult in any deployment.

Besides the traces studied in this chapter, there are many other wide-area failure traces available, such as for peer-to-peer (P2P) systems [BSV03, SGG02]. Failure correlation in P2P systems can be dramatically different from other wide-area systems, because many failures in P2P systems are due to user departures. It is part of our future work to extend our study to P2P environments. Another limitation of our traces is that the long probe interval prevents the detection of short-lived failures. This makes our availability results slightly optimistic.

6.2.3 Steps in our study

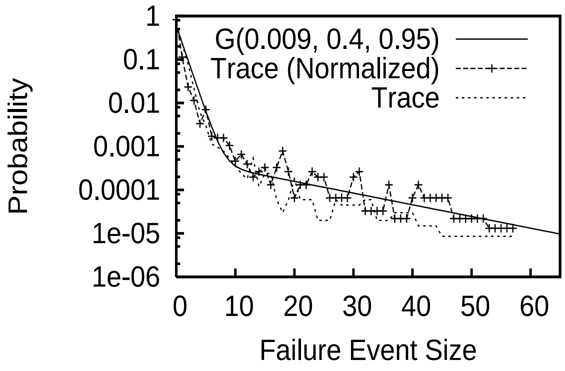
Section 6.3 constructs a tunable failure correlation model from our three failure traces; this model allows us to study the sensitivity of our findings beyond the three traces. Sections 6.4–6.6 present the four myths, and their corresponding realities, overlooked subtleties, and design principles. These sections focus on read-only ERASURE(m, n) systems, and mainly use trace-driven simulation, supplemented by model-driven simulation for sensitivity study. Section 6.7 shows that our conclusions for read-only systems readily extend to read-write systems. Note that most of the results in this chapter uses a custom simulator driven by traces and models; we validate the accuracy of the simulation results in Chapter 7.

6.3 A Tunable Model for Correlated Failures

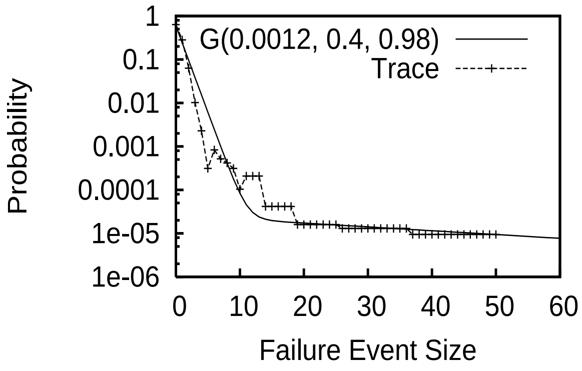
This section constructs a tunable parameterized failure correlation model from the three failure traces. The primary purpose of this model is to allow sensitivity studies and experiments with correlation levels that are stronger or weaker than in the traces. In addition, the model also later enables us to avoid overly-pessimistic and overly-optimistic designs, as well as to perform analytical studies for deeper understanding. The model balances idealized assumptions (e.g., Poisson arrival of correlated failure events) with realistic characteristics (e.g., mean-time-to-failure and failure size distribution) extracted from the real-world traces. In particular, it aims to accurately capture large (but rare) correlated failures, which have a dominant effect on system unavailability.

6.3.1 Correlated Failures in Real Traces

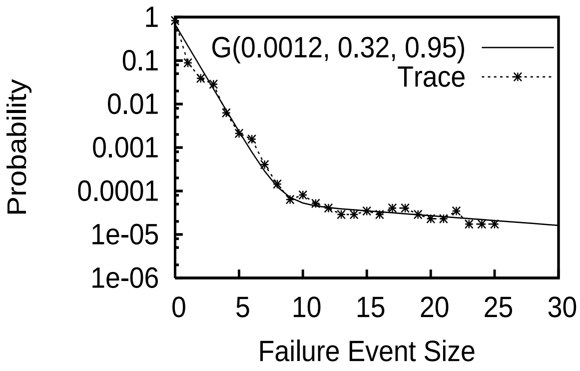
We start by investigating the failure correlation in our three traces. Figure 6.1 plots the PDF of failure event sizes for the three traces. While `RON_trace` and `WS_trace` have a roughly constant node count over their entire duration, there is a large variation in the total number of nodes in `PL_trace`. To compensate, we use both raw and normalized failure event sizes for `PL_trace`. The normalized size is the (raw) size multiplied by a normalization factor γ, where γ is the number of nodes in the interval divided by the average number of nodes (i.e., 277) in `PL_trace`. Because of the finite length of the traces, we cannot observe events with probability less than 10^{-5} or 10^{-6}.



(a) PL_trace



(b) WS_trace



(c) RON_trace

Figure 6.1: Probability distribution of correlated failures in three real-world traces. $G(\alpha, \rho_1, \rho_2)$ is our correlation model.

In all traces, Figure 6.1 shows that failure correlation has different strengths in two regions. In `PL_trace`, for example, the transition between the two regions occurs around event size 10. In both regions, the probability decreases roughly exponentially with the event size. However, the probability decreases significantly faster for small-scale correlated failures than for large-scale ones. We call such a distribution *bi-exponential*. Although we have only anecdotal evidence, we conjecture that different failure causes are responsible for the different parts of the distribution. For example, we believe that system instability, some application bugs, and localized network partitions are responsible for the small scale failure events. It is imaginable that the probability decreases quickly as the scale of the failure increases. On the other hand, human interference, attacks, viruses/worms and large ISP failures are likely to be responsible for large scale failures. This is supported by the fact that many of the larger PlanetLab failures can be attributed to DDoS attacks (e.g., on 12/17/03), system software bugs (e.g., on 3/17/04), and node overloads (e.g., on 5/14/04). Once such a problem reaches a certain scale, extending its scope is not much harder: the probability decreases relatively slowly as the scale of the failure increases.

6.3.2 A Tunable Bi-Exponential Model

In our basic failure model, failure events arrive at the system according to a Poisson distribution. The entire system has a *universe* of u nodes. The model does not explicitly specify which of the u nodes each failure event crashes, for the following reasons: Predicting failure patterns is not effective in improving availability, and pattern-aware fragment placement achieves almost identical availability as a pattern-oblivious random placement strategy (see Section 6.4). Thus, our availability study needs to consider only the case where the m fragments of a data object are placed on a random set of m nodes. Such a random placement, in turn, is equivalent to using a fixed set of m nodes and having each failure event (with size s) crash a random set of s nodes in the universe. Realizing the above point helps us to avoid the unnecessary complexity of modelling which nodes each failure event crashes – each failure event simply crashes a random set of s nodes.

To capture the bi-exponential property, we use a model that has two exponential components, one for each region. Each component has a tunable parameter ρ between 0 and ∞ that intuitively captures the slope of the curve and controls how strong the correlations are. When $\rho = 0$, failures are independent, while $\rho = \infty$ means that every failure event causes the failure of all u nodes. Specifically, for $0 < \rho < \infty$, we define the following geometric sequence: $f(\rho, i) = c(\rho) \cdot \rho^i$. The normalizing factor $c(\rho)$ serves to make $\sum_{i=0}^{u} f(\rho, i) = 1$.

We can now easily capture the bi-exponential property by composing two $f(\rho, i)$. Let p_i be the probability of failure events of size i, for $0 \le i \le u$. Our correlation model, denoted as $G(\alpha, \rho_1, \rho_2)$, defines $p_i = (1 - \alpha)f(\rho_1, i) + \alpha f(\rho_2, i)$, where α is a tunable parameter that describes the probability of large-scale correlated failures. Compared to a piece-wise function with different ρ's for the two regions, $G(\alpha, \rho_1, \rho_2)$ avoids a sharp turning point at the boundary between the two regions.

Figure 6.1 shows how well this model fits the three traces. We here provide a brief comparison among the traces here. The parameters of the model are different across the traces, in large part because the traces have different universe sizes (10 failures out of 277 is quite different from 10 failures out of 30). As an example of a more fair comparison, we selected 130 random nodes from `PL_trace` to enable a comparison with `WS_trace`, which has 130 nodes. The resulting trace is well-modelled by $G(0.009, 0.3, 0.96)$. This means that the probability of large-scale correlated failures in `PL_trace` is about 8 times larger than `WS_trace`.

Failure arrival rate and recovery

Up to this point, the correlation model $G(\alpha, \rho_1, \rho_2)$ only describes the failure event size distribution, but does not specify the event arrival rate. To study the effects of different levels of correlation, the event arrival rate should be such that the average mean-time-to-failure (MTTF) of nodes in the traces is always preserved. Otherwise with a constant failure event arrival rate, increasing the correlation level would have the strong side effect of decreasing node MTTFs. To preserve the MTTF, we determine the system-wide failure event arrival rate λ to be such that $1/(\lambda \sum_{i=1}^{u}(ip_i)) = \text{MTTF}/u$.

We observe from the traces that, in fact, there exists non-trivial correlation among node recoveries as well. Moreover, nodes in the traces have non-uniform MTTR and MTTF. However, our experiments show that the above two factors have little impact on system availability for our study. Specifically, for all parameters we tested, the availability obtained under model-driven simulation (which assumes independent recoveries and uniform MTTF/MTTR) is almost identical to that obtained under trace-driven simulation (which has recovery correlation and non-uniform MTTF/MTTR). For example, later in Figure 6.6, we demonstrate the matching of simulation results driven by `WS_trace` and the simulation results driven by $G(0.0012, 0.4, 0.98)$. Therefore, our model avoids the unnecessary complexity of modelling recovery correlation and non-uniform MTTF/MTTR.

6.3.3 Stability of the Model

Two important factors that decide the usefulness of our correlation model are how fast it converges and how long it remains stable. Fast convergence implies a small failure trace is sufficient to build a reasonable model. Better stability implies that a model remains effective for a long period of time.

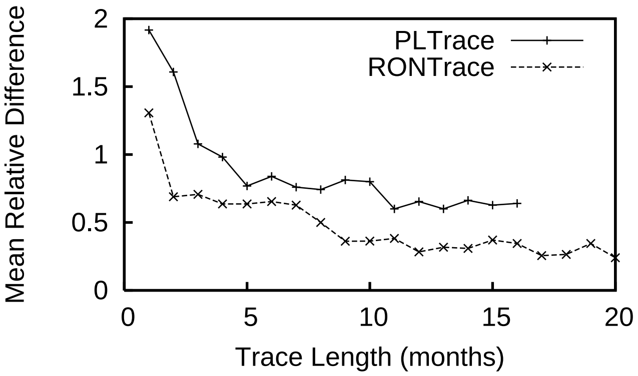
Convergence

We first study the convergence properties of our model, using `PL_trace` and `RON_trace` (we do not use `WS_trace` because of its short duration). For each trace, we obtain a "reference model" using the entire trace. Then, we consider successively longer portions of the trace, and compute the difference the model built from this smaller trace and the reference model. For our metric, we use *mean relative difference* (MRD), defined as the average of $|p_{i,t} - p_{i,m}|/p_{i,m}$ for all i's, where $p_{i,t}$ ($p_{i,m}$) is the probability of size-i failures from the trace portion (from the model, respectively). Intuitively, a small MRD means that the trace portion matches the reference model well.

Figure 6.2 plots the MRD between the reference models and suffixes of the traces, as a function of the length of the suffixes. For both traces, the figure shows that the MRD roughly monotonically decreases as we use a longer and longer trace. Also, the curves somewhat flatten after roughly 4 or 5 months. This means that the model does converge and the convergence time is roughly several months. We believe this convergence time is quite good given the rarity of large correlation events. For both traces, the MRD after 5 months is below 0.8. Note that the MRD from the model fittings in Figure 6.1 are 0.64, 0.60, and 0.24 for `PL_trace`, `WS_trace`, and `RON_trace`, respectively. This means that an MRD of 0.8 is quite satisfactory.

Effectiveness over time

Our model will later be used (among other things) to configure IrisNet to provide a given availability target. To be effective for such purposes, the model built ("trained") using a prefix of a trace should reflect well the failures occurring in the rest of the trace. To test this, we build models using the 2003 portions of `RON_trace` and `PL_trace`, and then compare these models to the 2004 portions of the respective traces. We find that the MRD is roughly 0.3 and 0.7 for `RON_trace` and `PL_trace`, respectively. Figures 6.3(a) and (b) provide a direct comparison of the availability achieved under the 2003-based models versus under the 2004 traces. In all cases, the availability difference is below half



For each trace, we obtain a "reference model" using the entire trace. Then, we consider successively longer portions of the trace, and compute the difference the model built from this smaller trace and the reference model. The differences decrease quickly over time, implying a fast convergence to the reference model. We do not consider `WS_trace` *here because of its short duration.*

Figure 6.2: Convergence of the correlation model. The Y-axis shows the difference between reference models and traces of different lengths.

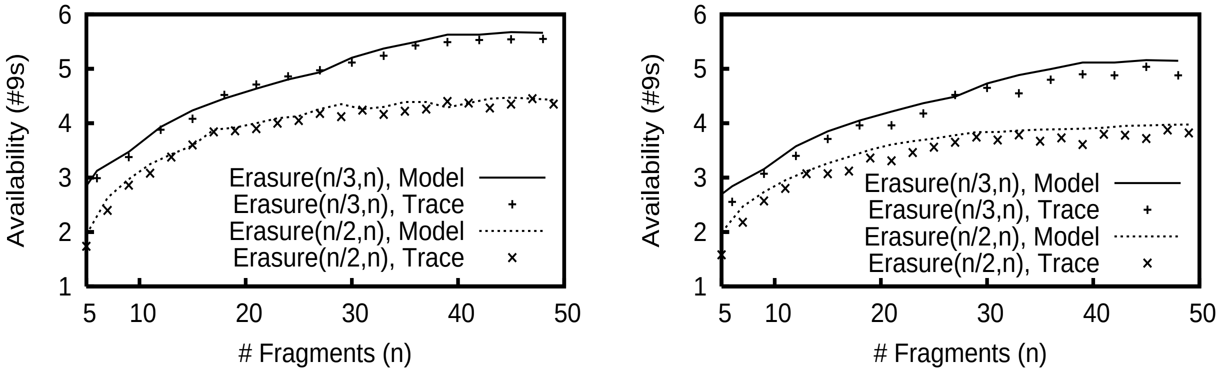
a nine. These results suggest that configuring IrisNet using the 2003-based model would have indeed been effective for reaching availability targets in 2004.

6.4 Myth: Correlated Failures Can Be Avoided Using Previously Proposed Failure Pattern Prediction Techniques

We present our four myths starting from this section. Unless otherwise stated, all our results will be based on the three real failure traces. The bi-exponential model is used only when we need to tune the correlation level, in which cases we will explicitly mention its use.

The Myth

Chun *et al.* [CV03] point out that node failure histories can be used to discover a relatively stable pattern of correlated failures (i.e., which set of nodes tend to fail together), based on a fraction of the `PL_trace` used in this study. Weatherspoon *et al.* [WMK02] (as part of the OceanStore project [KBC+00]) reach a similar conclusion by analyzing a four-week



(a) `RON_trace` (b) `PL_trace`

The Y-axis shows the predicted availability (based on our model computed from the first half of the trace) and the actual availability (given by the second half of the trace) under different traces. In each graph, the predicted availability closely matches the actual availability, showing that our model computed from the first half of the trace remains stable over the second half of the trace. We do not consider `WS_trace` *here because of its short duration.*

Figure 6.3: Stability of the correlation model.

failure trace of 306 web servers[2]. Based on such predictability, they further proposed a framework for online monitoring and clustering of nodes. Nodes within the same cluster are highly correlated, while nodes in different clusters are more independent in failure characteristics. They showed that the clusters constructed from the first two weeks of their trace are similar to those constructed from the last two weeks. Given the stability of the clusters, they conjectured that by placing the n fragments (or replicas) in n different clusters, the n fragments will not observe excessive failure correlation among themselves. In some sense, the problem of correlated failures goes away.

The Reality

We revisit this technique by using the same method as in [WMK02] to process our failure traces. For each trace, we use the first half of the trace (i.e., "training data") to cluster the nodes using the same clustering algorithm [SM00] as used in [WMK02]. Then as a case study, we consider two placement schemes of an Erasure$(n/2, n)$ (as in OceanStore) system. The first scheme (*pattern-aware*) explicitly places the n fragments of the same object in n different clusters, while the second scheme (*pattern-oblivious*) simply places

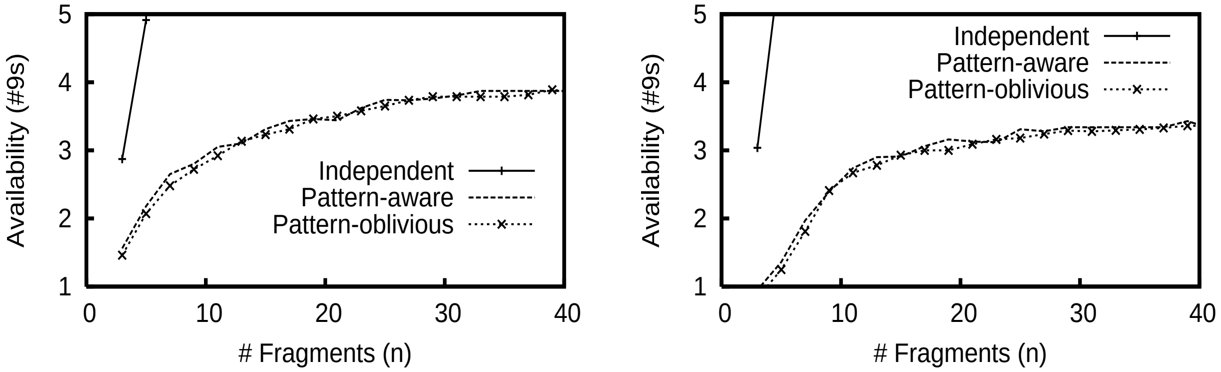
[2]The trace actually contains 1909 web servers, but they only analyze 306 servers because those are the only ones that ever fail during the four weeks.

the fragments on n random nodes. Finally, we obtain availability under the second half of the trace.

We first observe that most ($\approx$ 99%) of the failure events in the second half of the traces affect only a very small number (≤ 3) of the clusters computed from the first half of the trace. This implies that the clustering of correlated nodes is relatively stable over the two halves of the traces, which is consistent with [WMK02].

On the other hand, Figure 6.4 plots the achieved availability of the two placement schemes under `WS_trace` and `PL_trace`. We do not use `RON_trace` because it contains too few nodes for the clustering to be meaningful. The graph shows that explicitly choosing different clusters to place the fragments gives us only *negligible* improvement on availability. We also plot the availability achieved if the failures in `WS_trace` were independent. This is done via model-driven simulation and by setting the parameters in our bi-exponential model accordingly. For a fair comparison, we ensure that the machine MTTF and MTTR (and hence the machine unavailability) in the model match the MTTF and MTTR in `WS_trace`. Note that when the failures are independent, the two placement schemes do not make any difference. The large differences between the curve for independent failures and the other curves show that there are strong negative effects from failure correlation in the trace. Identifying and exploiting failure patterns, however, has almost no effect in alleviating such impacts. We have also obtained similar findings under a wide-range of other m and n values for $\textsc{Erasure}(m, n)$.

A natural question that arises is whether the above findings are because our traces are different from the traces studied in [CV03, WMK02]. Our `PL_trace` is, in fact, a superset of the failure trace used in [CV03]. On the other hand, the failure trace studied in [WMK02], which we call `Private_trace`, is not publicly available. Is it possible that `Private_trace` gives even better failure pattern predictability than our traces, so that pattern-aware placement would indeed be effective? To answer this question, we directly compare the "pattern predictability" of the traces using the metric from [WMK02]: the *average mutual information among the clusters* (MI) (see [WMK02] for a rigorous definition). A smaller MI means that the clusters constructed from the training data predict the failure patterns in the rest of the trace better. Weatherspoon *et al.* report an MI of 0.7928 for their `Private_trace`. On the other hand, the MI for our `WS_trace` is 0.7612. This means that the failure patterns in `WS_trace` are actually more "predictable" than in `Private_trace`.



(a) `WS_trace` (b) `PL_trace`

In this experiment, we compare two replica placement strategies. In the pattern-aware *strategy, we first cluster all the nodes based on their failure correlation such that nodes from different clusters have minimal correlation. Then we place replicas on different clusters such that individual replicas do not experience enough failure correlation. In the* pattern-oblivious *strategy, we choose random nodes as replicas. The graph shows that the pattern-aware strategy does not provide significant benefit over the simple pattern-oblivious strategy. We do not consider* `RON_trace` *here because of its small node population.*

Figure 6.4: Negligible availability improvements from failure pattern prediction for Erasure$(n/2, n)$ **systems.**

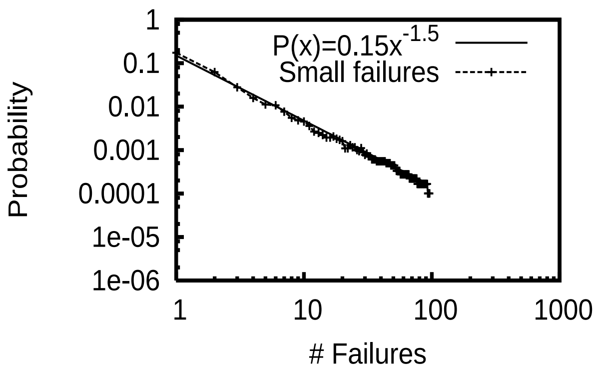
The Subtlety

To understand the seemingly contradictory observations in the previous section, we take a deeper look at the failure patterns. To illustrate, we classify the failures into small failures and large failures based on whether the failure size exceeds 15. With this classification, in all traces, most ($\approx 99\%$) of the failures are small.

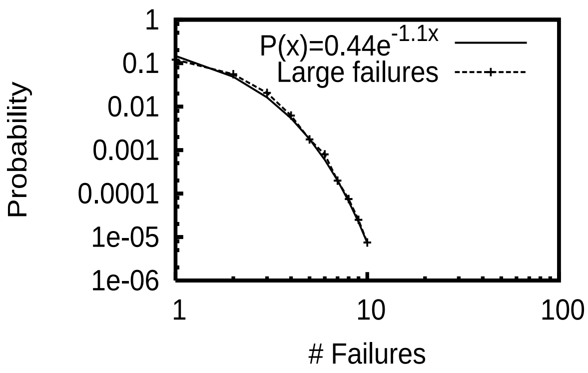
Next, we investigate how accurately we can predict the failure patterns for the two classes of failures. We use the approach used in [HBD97] for showing that UNIX processes running for a long time are more likely to continue to run for a long time in the future. For a random pair of nodes in `WS_trace`, Figure 6.5(a) plots the probability that the pair crashes together (in correlation) more than x times because of the small failures. The straight line in log-log scale indicates that the data fits the Pareto distribution of $P(\#failure \geq x) = cx^{-k}$. In Figure 6.5(b) and Figure 6.5(c), we observe similar fits for `PL_trace` ($P(\#failure \geq x) = 0.3x^{-1.45}$) and `RON_trace` ($P(\#failure \geq x) = 0.02x^{-1.4}$), respectively. Such a fit to the Pareto distribution implies that pairs of nodes that have failed together many times in the past are more likely to fail together in the future [HBD97]. Therefore, past pairwise failure patterns (and hence the clustering) caused by the small failures are likely to hold in the future.

Next, we move on to large failure events (Figure 6.5(d)). Here, the data fit an exponential distribution of $P(\#failure \geq x) = ae^{-bx}$. The memoryless property of the exponential distribution means that the frequency of future correlated failures of two nodes is independent of how often they failed together in the past. This in turn means that we cannot easily (at least using existing approaches) predict the failure patterns caused by these large failure events. Intuitively, large failures are generally caused by external events (e.g., DDoS attacks, worms, etc.), occurrences of which are not predictable in trivial ways. We observe similar results in the other two traces: Figure 6.5(e) and Figure 6.5(f) show exponential distribution fits for the large failure events in `RON_trace` and `PL_trace`, respectively.

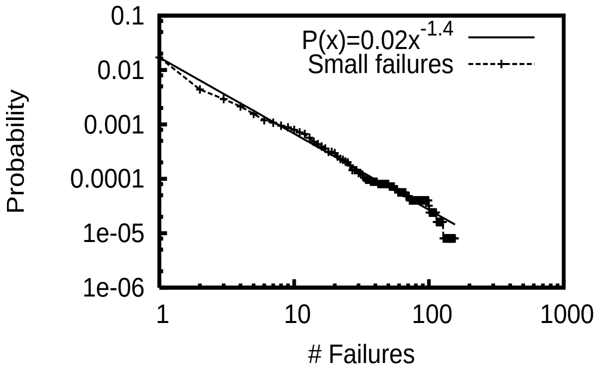
Thus, the pattern for roughly 99% of the failure events (i.e., the small failure events) is predictable, while the pattern for the remaining 1% (i.e., the large failure events) is not easily predictable. On the other hand, our experiments show that large failure events, even though they are only 1% of all failure events, contribute most to unavailability. For example, the unavailability of a "pattern-aware" $\textsc{Erasure}(16, 32)$ is 0.0003 under `WS_trace`. If we remove all of the small failure events, unavailability is still 0.0003. The intuition behind this result is that because small failure events only affect a small number of nodes, those failure events can be almost completely masked by data redundancy and



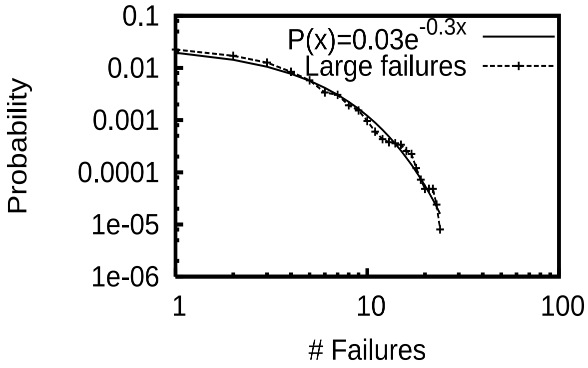
(a) Small failures in `WS_trace`



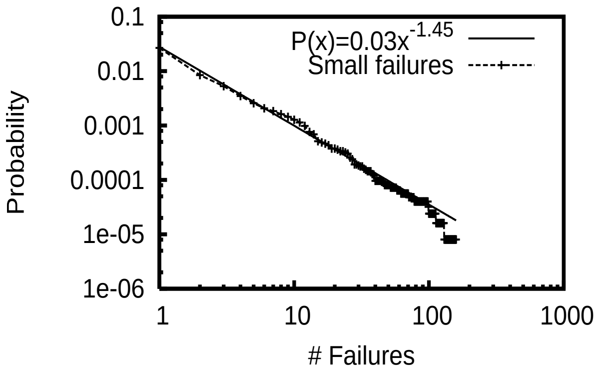
(a) Big failures in `WS_trace`



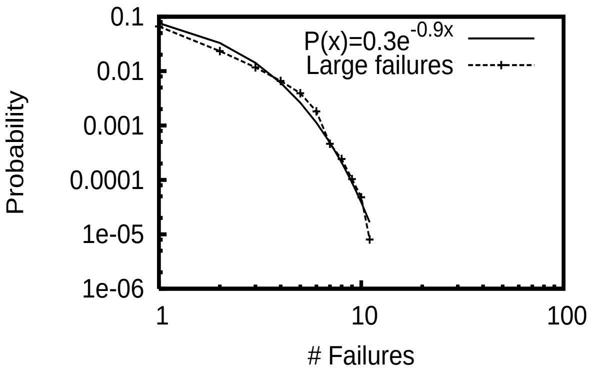
(c) Small failures in `RON_trace`



(d) Big failures in `RON_trace`



(c) Small failures in `PL_trace`



(d) Big failures in `PL_trace`

For two nodes selected at random, the plots show the probability that the pair will fail together (in correlation) more than x times during the entire trace. Distributions fitting the curves are also shown. Small failures show Pareto distributions, implying that their patterns can be predicted based on history. Big failures show exponential distributions, implying that their patterns can not be predicted in trivial ways.

Figure 6.5: Predictability of pairwise failures.

regeneration. This explains why on one hand, failure patterns are largely predictable, while on the other hand, pattern-aware fragment placement is not effective in improving availability. It is also worth pointing out that the sizes of these large failures still span a wide range (e.g., from 15 to over 50 in `PL_trace` and `WS_trace`), which means that capturing the distribution over all failure sizes is still important in the bi-exponential model.

The Design Principle

Large correlated failures (which comprise a small fraction of all failures) have a dominant effect on system availability, and system designs must not overlook these failures. For example, because failure pattern prediction that works well for the bulk of correlated failures fails for large correlated failures, it is not effective in alleviating the negative effects of correlated failures.

Discussion

We have established above that pattern-aware fragment placement is not effective because patterns of large failures can not be easily predicted in our wide-area failure traces. However, in some other scenarios the patterns of large failures can be predicted. For example, if the nodes in the systems are more heterogenous in terms of the software they run, nodes running the same version of the operating system may fail together because of some worm exploiting some bugs in that OS. Similarly, if we have several LAN clusters each with thousands of nodes, then clearly the patterns of some large failures (e.g., power outages that crash a whole cluster) can be predicted. In such cases, it is beneficial to place the fragments on different LANs. The same is true in P2P systems where failures are actually user leaves, which follow a clear diurnal pattern determined by the timezone of the user. In wide-area non-P2P systems, at least in our traces, there does not seem to exist large-scale strong correlations that would make the pattern of large failures predictable in this manner.

6.5 Myth: Simple Modelling of Failure Sizes Is Adequate

The Myth

A key challenge in system design is to obtain the "right" set of parameters that will be neither overly-pessimistic nor overly-optimistic in achieving given design goals. Most previous work on how to choose the data redundancy level for a given availability target assumes failure independence. Because of the complexity in availability estimation introduced by failure correlation, system designers sometimes make simplifying assumptions on correlated failure sizes in order to make the problem more amenable. For example, Glacier [HMD05] considers only the (single) maximum failure size, aiming to achieve a given availability target despite the correlated failure of up to a fraction f of all the nodes. Such simplification allows the system to use a closed-form formula of $availability = \sum_{k=m}^{n} \binom{n}{k} (1-f)^k f^{n-k}$, which is in fact the same as the formula under independent failures (with f being the failure probability). Using this formula, Glacier is then able to calculate the needed m and n values in Erasure(m, n).

The Reality

Figure 6.6 plots the number of fragments needed to achieve given availability targets under Glacier's model (with $f = 0.65$ and $f = 0.45$) and under `WS_trace`. Glacier does not explicitly explain how f can be chosen in various systems. But at least, we can expect that f should be a constant under the same deployment context (e.g., for `WS_trace`).

A critical point to observe in Figure 6.6 is that for the real trace, the curve is not a straight line (we explore the shape of this curve later). Because the curves from Glacier's estimation are roughly straight lines, they always significantly depart from the curve under the real trace, regardless of how we tune f. For example, when $f = 0.45$, Glacier over-estimates system availability when n is large: Glacier would use Erasure$(6, 32)$ for an availability target of 7 nines, while in fact, Erasure$(6, 32)$ only achieves slightly above 5 nines availability. Under the same f, Glacier also under-estimates the availability of Erasure$(6, 10)$ by roughly 2 nines. If f is chosen so conservatively (e.g., $f = 0.65$) that Glacier never over-estimates, then the under-estimation becomes even more significant. As a result, Glacier would suggest Erasure$(6, 31)$ to achieve 3 nines availability while in reality, we only needs to use Erasure$(6, 9)$. This would unnecessarily increase both the storage required for an object and the bandwidth used to create or update the object by over 240%.

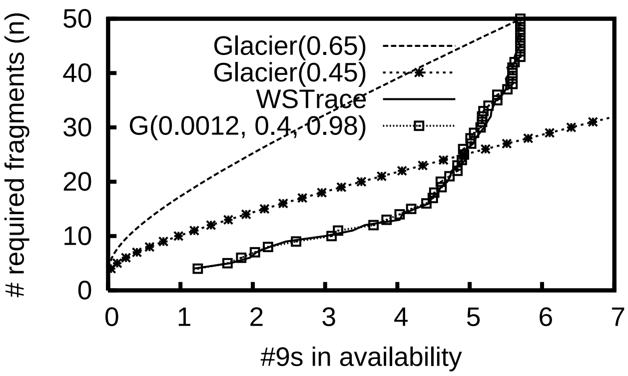


Figure 6.6: Number of fragments in $\textsc{Erasure}(6, n)$ **needed to achieve certain availability targets, as estimated by Glacier's single failure size model and our distribution-based model of G(0.0012,0.4,0.98). We also plot the actual achieved availability under the trace.**

The Subtlety

The reason behind the above mismatch between Glacier's estimation and the actual availability under `WS_trace` is that in real systems, failure sizes may cover a large range. In the limit, failure events of any size may occur; the only difference is their likelihood. System availability is determined by the combined effects of failures with different sizes. Such effects cannot be summarized as the effects of a series of single-sized failures (even with scaling factors).

To avoid the overly-pessimistic or overly-optimistic configurations resulting from Glacier's method, a system must consider a distribution of failure sizes rather than a single failure size. IrisNet uses the bi-exponential model for this purpose. Figure 6.6 also shows the number of fragments needed for a given availability target as estimated by our simulator driven by the bi-exponential model. The estimation based on our model matches the curve from `WS_trace` quite well. It is also important to note that the difference between Glacier's estimation and our estimation is purely from the difference between single failure size and a distribution of failure sizes. It is not because Glacier uses a formula while we use simulation. In fact, we have also performed simulations using a single failure size, and the results are similar. We choose to use Glacier's formula to adhere to the original method in [HMD05].

The Design Principle

Correlated failures should be modelled via a distribution instead of a maximum failure *size.*

6.6 Impact of Failure Correlation

Systems researchers have long been aware of the negative effects of correlated failures. However, systematically evaluating a design under correlated failures is quite difficult. Given such difficulty, system designs are often selected based on their evaluation under independent failures and then some overprovisioning is added in the hopes of offsetting the negative effects of failure correlation. This section discusses two myths associated with this approach.

6.6.1 Myth: Additional Fragments Are Always Effective in Improving Availability

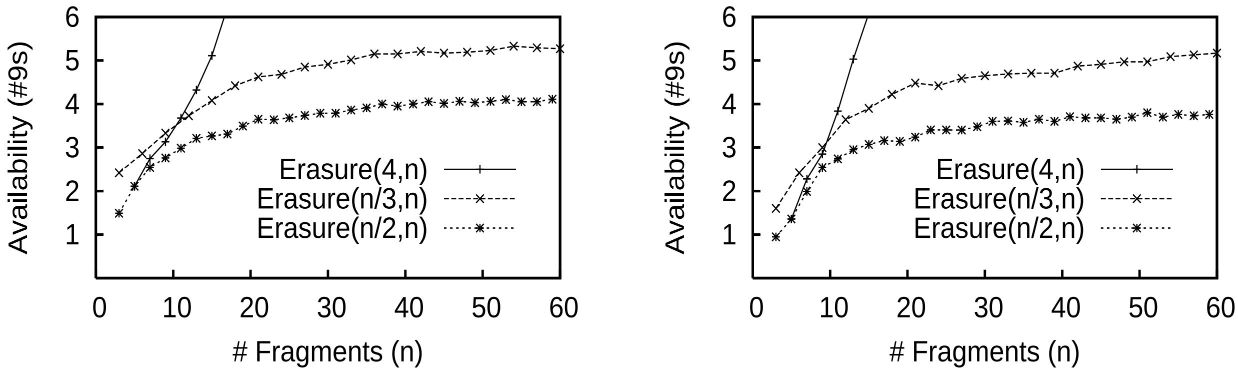
The Myth

Under independent failures, distributing the data always helps to improve availability, and any target availability can be achieved by distributing fragments across more and more machines, without increasing the storage overhead. For this reason, system designers sometimes fix the ratio between n and m (so that the storage overhead, n/m, is fixed), and then simply increase n to achieve the required availability target. As an example, in OceanStore, n/m is always kept at a constant of 2 (i.e., $\textsc{Erasure}(n/2, n)$ is used for some n), and OceanStore targets better availability by increasing n. For instance, under independent failures, $\textsc{Erasure}(16, 32)$ gives much better availability than $\textsc{Erasure}(12, 24)$, which, in turn, gives better availability than $\textsc{Erasure}(8, 16)$. CFS uses a similar $\textsc{Erasure}(n/2, n)$ scheme in its design. Given independent failures, we can even prove that increasing the number of fragments in these schemes *exponentially* decreases unavailability:

Theorem 6.1 *Consider n nodes where each node fails independently with probability p and $p < 0.5$. Then there exists a constant q, $0 < q < 1$, such that Prob[more than $n/2$ nodes fail] $\leq q^n$ for all n.*

Proof: Directly from Hoeffding's inequality [Hoe63]. $\square$

Thus, we can plausibly overprovision an $\textsc{Erasure}(m, n)$ system using a larger n to offset the negative effects of failure correlation, without increasing the storage overhead.



(a) WS_trace (b) PL_trace

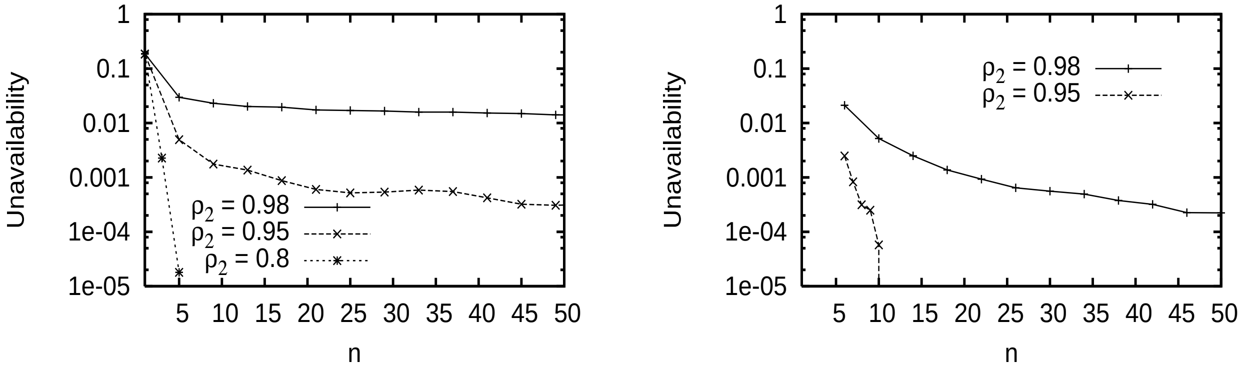
Figure 6.7: Availability of Erasure(m, n) **under different traces.**

There is a similar myth regarding read-write replication systems (i.e., overprovisioning by adding more replicas). Here the storage overhead increases with overprovision. Later in Section 6.7, we will show that a read-write replication system using majority voting [Tho79] for consistency has the same availability as Erasure$(n/2, n)$. Thus our discussion in this section also applies to read-write replication systems.

The Reality

We will show that, perhaps surprisingly, overprovisioning is not effective under correlated failures, even if we *double* or *triple* n. Figures 6.7(a) and 6.7(b) plot the availability of Erasure$(4, n)$, Erasure$(n/3, n)$ and Erasure$(n/2, n)$ under `WS_trace` and `PL_trace`, respectively, as a function of n. We do not use `RON_trace` because it only contains 30 nodes. Both Erasure$(n/3, n)$ and Erasure$(n/2, n)$ suffer from a strong diminishing return effect. For example, increasing n from 20 to 60 in Erasure$(n/2, n)$ provides less than a half nine's improvement. On the other hand, Erasure$(4, n)$ does not suffer from such an effect. By tuning the parameters in the correlation model and using model-driven simulation, we further observe (see Figure 6.8 and Figure 6.9) that the diminishing return effects become more prominent under stronger correlation levels as well as under larger m values.

The above diminishing return shows that correlated failures prevent a system from effectively improving availability by overprovisioning (without increasing storage). In read-write systems using majority voting, the problem is even worse: the same diminishing return effect occurs even allowing for significant increases in storage. Thus, for both popular erasure coding schemes and majority voting schemes, overprovisioning by increasing n is effective under independent failures, but not under correlated failures.



(a) $\textsc{Erasure}(n/2, n)$ (b) $\textsc{Erasure}(3, n)$

Diminishing return of availability becomes stronger as correlation level (ρ_2) increases or a larger m is used. In (b), when $\rho_2 = 0.8$, the unavailability is always below 10^{-5}.

Figure 6.8: Availability under the model $G(0.009, 0.4, \rho_2)$.

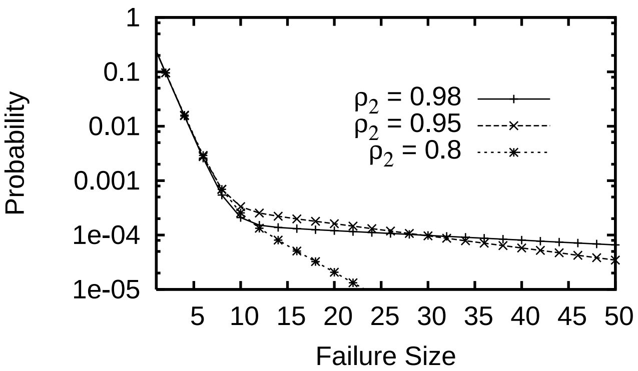


Figure 6.9: Failure size distribution under the model $G(0.009, 0.4, \rho_2)$.

The Subtlety

To find out the subtle cause behind the previous results, we analyze system availability under correlated failures using our bi-exponential model $G(\alpha, \rho_1, \rho_2)$. Let $P(j, n)$ be the probability that a failure event on a universe of u nodes causes exactly j failures among the n fragments ($0 \leq j \leq n \leq u$). As in our correlation model, we let $p_i = (1-\alpha)f(\rho_1, i) + \alpha f(\rho_2, i)$ be the probability that a failure event causes i failures in a universe of u nodes. Let $P(i, j, n)$, where $j \leq i \leq u$, be the probability that a failure event causes exactly i failures in the universe and causes exactly j failures among the n fragments. Then $P(j, n) = \sum_{i=j}^{u} P(i, j, n)$. We observe that if there are j failures out of n fragments, then there are $n - j$ non-failures among the n fragments, and hence at most $u - (n - j)$ failures in the universe. Thus $P(i, j, n) = 0$ whenever $i > u - (n - j)$. This gives $P(j, n) = \sum_{i=j}^{u-n+j} P(i, j, n)$.

We will now derive an equation for $P(i, j, n)$. Each $P(i, j, n)$ is the product of (1) the probability p_i that a failure event causes i failures and (2) the probability, call it $Q(i, j, n)$, that such a failure event causes j failures among n fragments. We can analyze $Q(i, j, n)$ by considering all the $\binom{u}{i}$ possible ways of selecting i out of u nodes for failure, and how many of these select exactly j out of the n fragments. There are $\binom{n}{j}$ ways of selecting j out of n and for each of these, there are $\binom{u-n}{i-j}$ ways of selecting the remaining $i - j$ failures out of the $u - n$ nodes not holding the n fragments. Because all $\binom{u}{i}$ ways are equally likely, we have:

$$Q(i, j, n) = \frac{\binom{n}{j} \cdot \binom{u-n}{i-j}}{\binom{u}{i}}$$

Thus, it follows that:

$$\begin{aligned}
P(i, j, n) &= p_i \cdot Q(i, j, n) \\
P(j, n) &= \sum_{i=j}^{u-n+j} p_i \cdot Q(i, j, n) \\
&= (1-\alpha)c(\rho_1) \sum_{i=j}^{u-n+j} \rho_1^i \cdot Q(i, j, n) + \\
&\quad \alpha c(\rho_2) \sum_{i=j}^{u-n+j} \rho_2^i \cdot Q(i, j, n)
\end{aligned}$$

To simplify the above equation, we define the following and simplify it by letting $k = i - j$:

$$h(\rho, j, n) = c(\rho) \sum_{i=j}^{u-n+j} \rho^i \cdot Q(i, j, n)$$

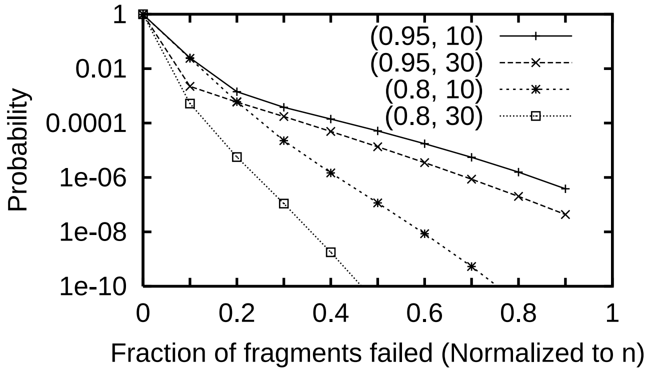


Figure 6.10: Fraction of fragments failed for $u = 277$, $\alpha = 0.009$ and $\rho_1 = 0.4$. The legends are in the form of (ρ_2, n).

$$= \frac{\binom{n}{j}(1-\rho)\rho^j}{1-\rho^{u+1}} \sum_{k=0}^{u-n} \rho^k \frac{\binom{u-n}{k}}{\binom{u}{k+j}}$$

With the above definition of $h(\rho, j, n)$, we have:

$$P(j,n) = (1-\alpha) \cdot h(\rho_1, j, n) + \alpha \cdot h(\rho_2, j, n) \tag{6.1}$$

A closer examination of this formula reveals a subtle, yet fundamental effect of correlated failures. Based on equation 6.1, Figure 6.10 plots the probability that a single failure event causes a certain fraction of node failures within the n nodes holding the n fragments, for two different values of ρ_2. Erasure($n/2, n$) is able to mask all the failure events that cause less than 50% of the fragments to fail, meaning that only the probabilities of the failure events causing more than 50% fragment failures matter. When $\rho_2 = 0.8$, increasing n from 10 to 30 significantly decreases the probability that over 50% of the fragments will all fail. However, when $\rho_2 = 0.95$, as in `PL_trace`, tripling n only slightly decreases this probability. The intuition for this is as follows. Although more fragments allow the system to tolerate more failures, they also mean that the scope of the vulnerability has increased – the same failure event may cause more failures within the n fragments. Such effects of a larger *vulnerability scope* become stronger when the correlation level increases. Ultimately, when $\rho_2 = 0.95$, such negative effects almost completely offset the benefits of being able to tolerate a larger absolute number of fragment failures.

To directly relate $P(j, n)$ to availability, we next provide a first-order approximation of the unavailability of Erasure($n/2, n$) by (1) considering that a single failure event that fails at least half the fragments is needed to initiate unavailability, and (2) ignoring that additional failure events may arrive while waiting to completely recover from this initial failure event:

$$U(n/2, n) \approx \lambda \sum_{j=n/2}^{n} P(j,n) \sum_{i=n/2}^{j} \frac{\text{MTTR}}{i}, \tag{6.2}$$

where $U(n/2, n)$ is the unavailability of $\textsc{Erasure}(n/2, n)$ and λ is defined in Section 6.3.2.

Using the above two approximations, we can study how doubling or tripling n affects availability. Let x be an integer that is not too large with respect to n and suppose $P(xj, xn) \approx P(j, n)$ (as in our earlier example), we have:

$$\begin{aligned}
& U(xn/2, xn) \\
\approx\ & \lambda \sum_{j'=xn/2}^{xn} P(j', xn) \sum_{i'=xn/2}^{j'} \frac{\text{MTTR}}{i'} \\
\approx\ & \lambda \sum_{j=n/2}^{n} \sum_{j'=xj}^{xj+x-1} \left(P(j', xn) \sum_{i'=xn/2}^{xj} \frac{\text{MTTR}}{i'} \right) \\
\approx\ & \lambda \sum_{j=n/2}^{n} \sum_{j'=xj}^{xj+x-1} \left(P(j', xn) \cdot \sum_{i=n/2}^{j} \left(\sum_{i'=xi}^{xi+x-1} \frac{\text{MTTR}}{i'} \right) \right) \\
\approx\ & \lambda \sum_{j=n/2}^{n} x \cdot P(xj, xn) \sum_{i=n/2}^{j} \frac{\text{MTTR}}{i} \\
\approx\ & x \cdot \lambda \sum_{j=n/2}^{n} P(j,n) \sum_{i=n/2}^{j} \frac{\text{MTTR}}{i} \\
\approx\ & U(n/2, n) \qquad \text{[See below how the coefficient } x \text{ gets cancelled out]}
\end{aligned}$$

This means that under certain correlation levels (so that $P(xj, xn) \approx P(j, n)$ holds, as in our earlier example), even doubling or tripling n will not be effective in improving availability under $\textsc{Erasure}(n/2, n)$.

A closer look at the above analysis may appear that $U(xn/2, xn)$ could actually be larger than $U(n/2, n)$ by x folds, if $P(xj, xn)$ actually equals $P(j, n)$. However, notice that it is impossible for $P(xj, xn)$ to equal (or be larger than) $P(j, n)$. The reason is that $\sum_{j=0}^{n} P(j, n) = \sum_{j'=0}^{xn} P(j', xn) = 1$. If $P(xj, xn)$ actually equals $P(j, n)$, then $\sum_{j'=0}^{xn} P(j', xn)$ will be larger than 1.0. In fact, the closest $P(xj, xn)$ can get with respect to $P(j, n)$ is around $1/x \cdot P(j, n)$ (in order to keep $\sum_{j'=0}^{xn} P(j', xn) = 1$). This factor of $1/x$ exactly cancels out the x coefficient in the previous analysis.

The only way to lessen these diminishing return effects is to non-trivially increase the n/m ratio (i.e., the storage overhead) as we increase n, by keeping m small. When m is small, the previous analysis no longer holds. Thus, unlike Erasure$(n/2, n)$, Erasure$(4, n)$, for example, is able to mask an increasingly large fraction of failed fragments as n increases, as confirmed by Figures 6.7(a) and 6.7(b).

The Design Principle

System designers should be aware that correlated failures result in strong diminishing return effects in Erasure*(m, n) systems unless m is kept small. For popular systems such as* Erasure*$(n/2, n)$, even doubling or tripling n provides very limited benefits after a certain point.*

6.6.2 Myth: Better Designs under Independent Failures Remain Better under Correlated Failures

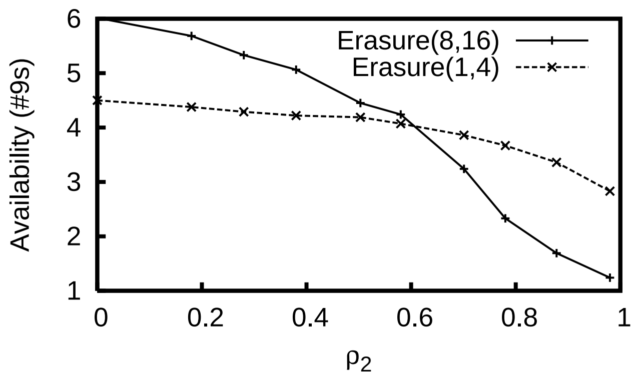
The Myth

If failure correlation affects all designs roughly equally, then we can plausibly still compare designs under independent failures. Namely, designs that are better under independent failures would remain better under correlated failures.

The Reality

We find that, unfortunately, this is not always true—correlated failures hurt some designs much more than others. Such non-equal effects are demonstrated via the example in Figure 6.11. Here, we plot the unavailability of Erasure$(1, 4)$ and Erasure$(8, 16)$ under different failure correlation levels, by tuning the parameters in our correlation model in model-driven simulation. These experiments use the model $G(0.0012, \frac{2}{5}\rho_2, \rho_2)$ (a generalization of the model for `WS_trace` to cover a range of ρ_2s), and a universe of 130 nodes, with MTTF = 10 days and MTTR = 1 day. Erasure$(1, 4)$ achieves around 1.5 fewer nines than Erasure$(8, 16)$ when failures are independent (i.e., $\rho_2 = 0$). On the other hand, under the correlation level found in `WS_trace` ($\rho_2 = 0.98$), Erasure$(1, 4)$ achieves 2 more nines of availability than Erasure$(8, 16)$[3].

[3] The same conclusion also holds if we directly use `WS_trace` to drive the simulation.



$\rho_2 = 0$ indicates independent failures, while $\rho_2 = 0.98$ indicates roughly the correlation level observed in `WS_trace`*.* ERASURE*(8, 16) is better under independent failure, but worse under correlated failures, than* ERASURE*(1, 4).*

Figure 6.11: Effects of the correlation model $G(0.0012, \frac{2}{5}\rho_2, \rho_2)$ on two systems.

The Subtlety

The cause of this (perhaps counter-intuitive) result is the diminishing return effect described earlier. As the correlation level increases, ERASURE$(8, 16)$, with its larger m, suffers from the diminishing return effect much more than ERASURE$(1, 4)$. In general, because diminishing return effects are stronger for systems with larger m, correlated failures hurt systems with large m more than those with small m.

The Design Principle

A better design under independent failures may not be better under correlated failures. In particular, correlation hurts systems with larger m more than those with smaller m. Therefore, system designs should be explicitly evaluated and compared under correlated failures.

6.7 Read-Write Systems

Thus far we have discussed the four myths in the context of read-only ERASURE(m, n) systems. Interestingly, our results extend quite naturally to a sensing system, or to any read-write system that uses quorum systems or voting techniques to maintain data consistency.

Consider a quorum system (or voting system) that has a replica count of n and a quorum size of m—an operation can be performed if *any* m of the n replicas are available. From an availability perspective, this is exactly the same as a read-only $\textsc{Erasure}(m, n)$ system. Therefore, our results on $\textsc{Erasure}(n/2, n)$ holds directly for a majority quorum system. Similarly, the availability of SQS with a quorum size of c is same as that of $\textsc{Erasure}(c, n)$.

Regeneration makes the equivalency slightly more complex. It is quite tricky to ensure data consistency during regeneration in a read-write system. In IrisNet, we adopt previous regeneration designs from RAMBO [LS02], and use the Paxos distributed consensus protocol [Lam98] to ensure consistency during regeneration. Paxos needs $n/2$ out of the n nodes to be available in order to terminate.[4] Thus, a read-write, SQS-based regenerating system with a quorum size of m requires $n/2$ (instead of m) replicas to regenerate, even though only m replicas are needed for normal reads and writes. In comparison, a read-only $\textsc{Erasure}(m, n)$ system requires m fragments to regenerate, as well as m fragments for user accesses. We have performed extensive analysis and simulation of these SQS-based regenerating systems and found that our previous design principles still apply.

Finally, quorum systems can also be used over erasure-coded data [GWGR04]. Despite the complexity of these protocols [GWGR04], they all have simple threshold-based requirements on the number of available fragments. As a result, their availability can also be readily captured by properly adjusting m in $\textsc{Erasure}(m, n)$ in our results.

In the next chapter, we will show that IrisNet uses SQS with replica count 7 and quorum size 2. Thus, all the results of $\textsc{Erasure}(2, 7)$ we presented in this chapter readily hold for IrisNet. IrisNet also uses a simplified version of RAMBO, as described above.

6.8 Artifacts

The traces mentioned in Table 6.1 can be found at `http://www.intel-iris.net/traces`.

We have also developed Correlated Failure Benchmark (CFB), a failure benchmark generator tool that produces synthetic failure traces according to the bi-exponential model we developed in this section. It takes as input the number of nodes in the system and the parameters of the model (either the built-in models (PlanetLab, RON, or WebServers) or the bi-exponential parameters (α, ρ_1, and ρ_2)), and outputs a stream of failure- and

[4] It is possible to use a completely different set (and a much larger number) of nodes for Paxos to maximize its success probability [YMV+03]. Such a design will make the results for $\textsc{Erasure}(m, n)$ directly apply to a read-write replication system with a quorum size of m. Namely, the extra adjustment as described in this section is no longer needed.

recover-events with their timestamps. Currently, the tool has several limitations, e.g., it does not distinguish between node- and link-failures, does not consider recovery correlation (based on our analysis we believe that node recoveries often show significant correlation), etc. Addressing these limitations is part of our future work. The tool is available at `http://www.intel-iris.net/benchmarks`.

6.9 Summary

In this chapter, we have shown that previously proposed approaches for combating correlated failures, although plausible, are less effective than one might hope under real-world failure correlation, often resulting in system designs that are far from optimal. Our study also reveals the subtleties that cause this discrepancy between the perception (the myth) and the reality. These new findings lead to the following four design principles for tolerating correlated failures in distributed storage systems.

[**P1**] Large correlated failures (which comprise a small fraction of all failures and whose patterns are difficult to predict) have a dominant effect on system availability, and system designs must not overlook these failures.

[**P2**] Correlated failures should be modelled via a distribution instead of a maximum failure size.

[**P3**] System designers should be aware that correlated failures result in strong diminishing return effects in systems with large quorums.

[**P4**] A better design under independent failures may not be better under correlated failures.

We have shown that the above design principles hold for both read-only or read-write systems.

Implications of Designing Sensing Systems. **P4** implies that correlated failures should be explicitly considered in our design of a sensing system. An important corollary of **P3** is that weak quorum systems (e.g., SQS) are more suitable for Internet-scale sensing systems since such systems can tolerate occasional data inconsistency caused by weak quorum systems and as a result can significantly improve availability. A corollary of **P1** is that the random replica placement strategy required by SQS provides near optimal availability. Finally, **P2** implies that a sensing system should use the failure distribution to determine the configuration parameters in order to optimize the target availability and the resource usage.

In the next chapter we show how we incorporate these principles in IrisNet's design.

Chapter 7

Design and Implementation of Sensor Data Storage in IrisNet

In this chapter, we show how several unique properties of a sensing system can be exploited to make its data storage component highly available. Our discussion here focuses on three components that are crucial for the storage component's high availability: replication, regeneration, and load balancing. We discuss the following techniques in this chapter:

- We use the principles from Chapter 6 to design effective replication techniques. By exploiting the weak data consistency requirements of sensing applications, we use Signed Quorum Systems (SQS) and show how it can be incorporated in a hierarchical sensor database design.

- We show how the easily serializable timestamped writes and the absence of write-sharing in a sensing system simplify an existing regeneration algorithm (RAMBO [LS02]). We also show how to deal with SQS's occasional inconsistency in the regeneration algorithm.

- We show how the hierarchical structure of the sensor database and the typical routing of user queries can be exploited in order to efficiently make complex load balancing decisions. Our techniques guard against unavailability due to node overload (e.g., during flash crowds).

We design and implement these techniques in IrisNet, and therefore, our description of these techniques uses the context of IrisNet. We start this chapter with a discussion of the general design space and the rationale behind our design choices (Section 7.1).

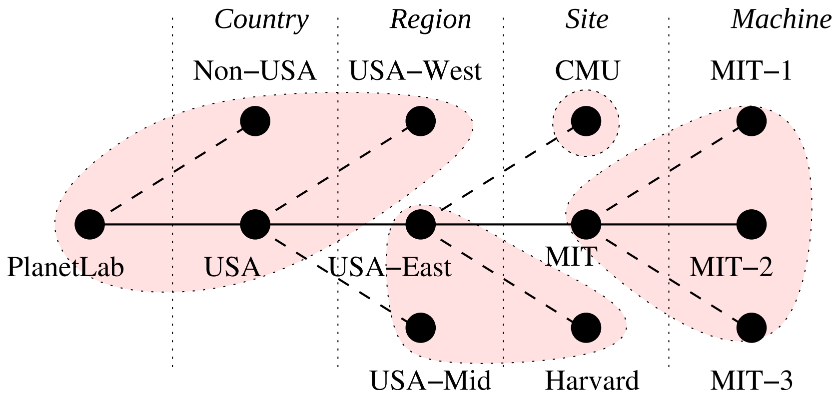
Table 7.1: Terminology used in this Chapter.

Term	**Meaning**
Node	An IrisNet OA node
Global database	The XML database that contains sensor data and is distributed among IrisNet OAs.
Local database	The portion of the global database stored by a node.
Tree	The logical hierarchy of the global database. The tree is used for query routing and in-network aggregation.
Element	An XML element. Represents a node in the tree.
Object	An element. We use the terms "element," "XML element," and "object" interchangeably.
Fragment	Part of the local database required to transfer to a remote OA.
Replica group	Set of all the replicas of an object.

Then, in Section 7.2, we describe IrisNet's data replication and its use of SQS for accessing data. In Section 7.3, we describe IrisNet's automatic replica regeneration algorithm. In Section 7.4, we describe IrisNet's dynamic load balancing algorithm. Finally, in Section 7.5, we evaluate these techniques with a combination of real deployment, emulation, and simulation.

To refresh the reader's memory, we here briefly repeat IrisNet's data indexing and query processing mechanism. Each IrisNet application stores its sensor data such as timestamped temperature data collected from motes, and user data such as installed triggers in a distributed XML database. The XML database can be viewed as an aggregation tree defining a hierarchy, where each tree node is an XML element. Table 7.1 lists the terminology we use in this section. The global database can be fragmented and replicated among a set of nodes; the OA running on each node maintains its fragments in a local database. IrisNet supports a flexible fragmentation of the XML database—any subset of XML elements can be placed in any set of nodes. Figure 7.1 shows part of the hierarchy used in IrisLog (described in Section 3.4.2) and a hypothetical fragmentation of it among four nodes.

IrisNet supports two types of queries. A *snapshot* query asks for a subset of the sensor data stored in the leaf elements of the tree. A query is routed to the nodes holding the relevant XML elements. A typical query is routed top down the tree, with query predicates being evaluated in intermediate elements of the tree (more details in Chapter 3). In a *continuous* query sensor data from the leaf elements are continuously pushed up the tree, and aggregate (derived) data are stored in the non-leaf elements of



The black solid circles denote XML elements. The database is fragmented and placed in four nodes (shown as shaded).

Figure 7.1: Part of the XML database used by our IrisLog application.

the tree.

7.1 Desiderata and Design Rationale

We aim to make the storage layer highly available, despite large correlated failures caused by catastrophes common in today's Internet. We use standard techniques, e.g., data replication, to tolerate such failures. We consider the following two types of correlated failures, and want IrisNet to be available during and survive these failures.

- **Crash-failures:** These failures happen because of IrisNet nodes crashing (e.g., due to attacks or software bugs) or being partitioned from the network. Such failures are generally *untargeted*—they do not directly target the nodes in a particular replica group. For example, suppose 10 of the 100 nodes on which IrisNet is deployed fail. For untargeted failures, in any 10 nodes that IrisNet may have chosen to use as a replica group, we will likely see 1 instead of 10 simultaneous failures. We expect most system software or hardware failures to be untargeted as long as we choose replicas randomly. Likewise, we expect most external attacks on IrisNet to result in untargeted failures as long as we are able to conceal the membership of replica groups.

- **Overload-failures:** These are the failures caused by IrisNet nodes getting highly overloaded (e.g., due to a flash crowd) and therefore not being able to provide the intended service (e.g., not returning query answer in a reasonable time). Such failures are generally *targeted* to the nodes in a particular replica group—a burst of queries or updates involving some specific sensor data can overload all of its replicas.

Standard techniques for tolerating crash-failures include replication along with automatic regeneration, and those for tolerating overload-failures include dynamic load balancing. Below we present the general design space and the rationale behind the particular solutions we adopt in the IrisNet design. Most of our design decisions are influenced by the target domain of sensing applications.

7.1.1 Replication Design

Important replication design decisions include choosing the appropriate quorum systems, replica count, and replica placement strategy. Being motivated by the design principle **P4** in Chapter 6, we explicitly take correlated failures into account in making these decisions.

Quorum Systems

As shown in the previous chapter, systems with smaller quorums are more effective in tolerating correlated failures (design principle **P3**). Strict quorum systems like majority voting suffers from diminishing availability returns. This implies the tradeoff between a system's strong data consistency and its tolerance for large correlated failures.

We choose to favor the tolerance for correlated failures in this tradeoff, and decide to use weak quorum systems in IrisNet. As we discussed in Chapter 1, most sensing applications can tolerate occasional data inconsistency, either by reissuing queries or by exploiting temporal or spatial correlation of sensor data. In contrast, at the cost of this small inconsistency, IrisNet can significantly improve its availability in the presence of catastrophic failures. Another advantage of using smaller quorums is reduced access overhead—readers and writers need to contact fewer nodes for each read and write.

Of the two weak quorum systems we discussed in Section 2.2.2, we choose Signed Quorum Systems (SQS) in IrisNet. We choose it for its smaller quorum sizes and simplicity. Note that the previous work on SQS [Yu04] only focuses on its theory; correlated failures were not considered and SQS were neither implemented nor studied in real systems. Our design, implementation, and evaluation show its performance in a real system.

Replica Count

The standard approach to determining the replica count necessary to provide a certain availability target is to use mathematical analysis based on individual node availability. Existing systems use either an independent failure assumption or a maximum failure size (as in Glacier) in determining the necessary number of replicas. For example, given the average failure probability of an individual node to be p, a quorum size m and a target system availability A, the replica count n can be computed by solving the equation $A = \sum_{k=m}^{n} \binom{n}{k}(1-p)^k p^{n-k}$. However, as we have shown in the previous chapter, real-world failures are correlated and correlated failures should be modelled via a distribution instead of a maximum failure size (design principle **P2**). Therefore, we use a different approach in IrisNet design. Our design explicitly quantifies and compares configurations (replica count and quorum size) via online simulation with our correlation model (described in Section 6.3) and chooses a configuration that achieves the target availability. As we have shown in Chapter 6, configurations chosen based on our simulation closely achieve the target availability.

Replica Placement

Many systems, including OceanStore and Phoenix, try to predict correlation pattern by finding nodes with mutually independent failure properties and then place replicas on these independent nodes with the hope of avoiding correlated failures. As we have shown in the previous chapter, such techniques are not effective in an Internet-scale system (design principle **P1**). We therefore do not use such techniques in IrisNet—each replica in IrisNet is just a randomly chosen node.

7.1.2 Regeneration Design

The biggest challenge in automatic replica regeneration in a read-write system is to avoid replica divergence due to false failure detection (recall the discussion in Section 2.2.3). The standard trick to avoid this is to use a regeneration quorum system, independent of the data access quorum system, to ensure that replicas agree on the membership of the new replica group. Existing regeneration systems differ on their choice of such quorum systems. For example, RAMBO [LS02] uses a majority quorum system—replicas must coordinate (using the Paxos consensus protocol [Lam98]) with a majority of the existing replicas to start regeneration. In contrast, Om [YV04] uses a randomized consensus protocol and a witness model [Yu03] that achieves similar functionality as a quorum system. In the witness model, quorum intersection is not always guaranteed, but is extremely likely. In return, a quorum in the witness model can be as small as a single node and therefore Om provides better availability than RAMBO.

Despite Om's improved availability, we adopt RAMBO's design in IrisNet. The choice is motivated by the following observations. First, regeneration is often not time sensitive; it is necessary to regenerate only before the next failure hits. Therefore unavailability of regeneration has a small effect on unavailability of data access. Second, since Om uses the witness model for regeneration, there is a certain probability of replica divergence. Once this happens, the system will remain inconsistent until the bad scenario (e.g., large-scale network partition) goes away. No guarantee is provided on how long such a scenario will last. We like to avoid such scenarios since we want to reduce human intervention and data may be accessed (written or read) frequently in a sensing application. Finally, since regeneration is rare (as rare as failures) compared to data access, improving the performance of regeneration has little impact on the overall system performance.

Note that IrisNet's use of weak data quorum systems and a strict regeneration quorum systems presents a different design point than Om (that uses strict data and weak regeneration quorum systems) and RAMBO (that uses strict data and regeneration quorum systems).

7.1.3 Load Balancing Design

To avoid overload-failures, an overloaded IrisNet node fragments its local database and sends the fragments to a lightly loaded node (so that the overloaded node does not see the queries and updates involving the transferred fragments). The design of IrisNet's dynamic load balancing algorithm is influenced by the data indexing and access characteristics of a typical sensing system (and of IrisNet): 1) each node hosts a portion of the hierarchically organized data, and 2) data is accessed in a top-down fashion (beginning with the starting point OA for the query—recall Section 3.3.2), with query predicates being evaluated at each level of the hierarchy. To efficiently process a query, it is desirable that it does not traverse a large number of nodes in its top-down traversal of the tree, since the fixed latency (and other overheads) of accessing the local database of each node adds up to the end-to-end latency of the query. This motivates careful fragmentation and placement strategies of hierarchically organized data such that a typical query needs to access a few number of nodes. Our solution, called Post, achieves these requirements.

As we show later, finding an optimal solution for the formal fragmentation and placement problem is NP-hard. To develop a practical solution, we make three simplifications.

First, we let each host partition its local database using efficient heuristics, instead of using expensive optimal algorithms. Second, we let an overloaded host partition its local database independently, based on its local load statistics. Finally, to mitigate the suboptimal results of these local decisions, we use placement heuristics that aim to yield "good" global clustering of data. When a node approaches overload, POST examines the workload and identifies portions of the local database that should be offloaded to other nodes. Compared to previous approaches [Luk74, SW03], POST exploits both the hierarchical structure of the sensor database and the typical routing of user queries, in order to make the complex fragmentation decisions efficiently.

In the rest of this chapter, we describe IrisNet's replication design in Section 7.2, regeneration design in Section 7.3, and load balancing design in Section 7.4. Finally, we evaluate the designs in Section 7.5 by implementing them in IRISLOG.

7.2 Replication in IrisNet

7.2.1 Replication Basics

For robustness, each object in the global database is replicated on multiple nodes. Each such node is called a *replica*. Different from such a design, a few previous systems such as SplitStream [CDK+03] achieve robustness using multiple trees. A simple analysis show that replication at the node level achieves better availability than replication at the entire tree level. Suppose we have n nodes in the tree and each one fails independently with a probability p. Suppose, in our approach, we use k replicas for each of the n tree nodes. The failure probability of the tree is roughly np^k when $p \to 0$. On the other hand, with k different trees, the probability of not having any tree available is $(np)^k > np^k$.

A replica is a *primary* of a replica group if it believes that it has the smallest IP among all live replicas in the replica group.[1] Such belief is obtained using inaccurate failure detection, and it is possible for one replica group to have multiple primaries.

On receiving a snapshot query, a node first reads from the read quorum of the relevant elements. This provides consistent data on which the node can evaluate query predicates to decide where to send subsequent subqueries. For continuous queries, a primary of a replica group containing particular XML elements periodically (every 10 seconds in IRISLOG) pushes corresponding updates to the replica groups containing the parent XML elements. A primary does not necessarily push updates upon every update in order to

[1] In our design, it is not necessary to use IPs for such ordering purpose. Any ordering among the replicas suffices as long as all replicas agree on such ordering.

control the traffic incurred. Otherwise if the XML elements on a replica group have many children XML elements, the replica group will constantly receive updates from below and try to push them upward. When the sensor has some critical data that needs to be propagated promptly, we also allow the sensor to flag the update which will incur immediate push.

A primary is also responsible for periodically evaluating user triggers installed at the replica group. We choose the primary to do this in order to avoid unnecessary redundant push traffic or trigger evaluation. The correctness of our system is not affected by multiple primaries, since multiple push of the same data will be filtered by the upper level replica group. While it is possible that a trigger is triggered multiple times, such a scenario is provably unavoidable in a failure-prone environment (since it reduces to the *Coordinated Attack Problem* [Lyn97]). This implies that IrisNet triggers are designed to be idempotent.

The above simple design would be sufficient if there were no failures. However, the presence of failures and inaccurate failure detection in WANs makes this design unable to ensure data consistency. We need to use quorum systems to ensure data consistency.

7.2.2 Providing Consistency

In IrisNet, our consistency model is to require any read that starts (in global physical time) after a write finishes to observe that write. Note that we assume that all writes commute and hence we do not serialize writes. This is motivated by our sensing context where writes from the same sensor have version numbers or timestamps to indicate which write to an element is fresher. On the other hand, writes from different sensors update different data items and do not conflict.

For better tolerance against correlated failures, IrisNet uses SQS for data consistency. The SQS construction in IrisNet is adopted from the *optimal* construction in [Yu04]: *n replicas are used, with quorum size m (a constant). A writer or reader probes the replicas according to the same fixed order (e.g., an ordering based on the hash of IP addresses and element names) until either m replicas are successfully accessed or all replicas have been probed.* In the former case, it can easily be proved that if the writer and reader do not intersect, there are at least m mismatches. In the latter case, the replica group is unavailable. The quorum size m can be tuned, and may be much smaller than $n/2$, as used by majority voting systems. A larger m decreases the probability of stale reads but increases unavailability. SQS also reduces the overhead of reads and writes, because they must access only m replicas rather than a majority. Note that although the particular SQS construction we use does not balance load within a single replica group (because replicas are accessed in a specific order), an IrisNet node contains replica of multiple objects. Thus, on average, load is evenly distributed among nodes.

At this point, it may appear strange why we use m smaller than $n/2$. With majority voting, we can tolerate $n/2$ simultaneous failures before the system becomes unavailable, while with only m fresh replicas, we can tolerate only m simultaneous failures before we always have stale reads on that data. However, a critical point is that the probability of $n/2$ simultaneous failures out of n nodes can be much larger than the probability of m simultaneously failures out of m nodes. More specifically, when $m = n/2$, the former probability is $\binom{n}{n/2}$ (e.g., 252 when $n = 10$) times the latter probability. As discussed above, the choice of m is an application-dependent trade-off: a smaller m achieves higher availability while a larger m achieves fewer stale reads.

It is obvious that the above SQS construction improves availability because we now need only m instead of $n/2$ available replicas. This becomes critically important in the presence of highly correlated failures, because once $n/2$ replicas have failed, we are unable to regenerate and must wait for nodes to recover. With majority voting, the replica group is unavailable during this time, while with IrisNet, reads and writes can still proceed as long as $m \ll n/2$ remain available. A less obvious, but equally important advantage is that each read and write now accesses only m replicas instead of $n/2$ replicas. In a large-scale distributed system, sometimes the limiting factor on the number of replicas is *not* whether there are sufficiently many nodes or sufficient disk space. For example, a popular web object can be replicated in a large number of proxy caches, and the same is true for a DNS entry. It is more likely that the overhead of reads and writes limits the number of replicas. Because SQS potentially decouples read and write overhead from the number of replicas, we are able to use a larger number of replicas than was previously feasible.

We can now explain why a primary replica must read data from a read quorum before it can push the data or evaluate any triggers or predicates. Suppose a replica group has three replicas A, B and C, of which A is the primary and an SQS with $m = 2$ is used. Imagine that this replica group is the parent of three sensors S_1, S_2 and S_3 in the tree. With SQS, suppose S_1 updates A and B, S_2 updates B and C (for it finds A unreachable), and S_3 updates A and C (for it finds B unreachable). Clearly, none of the three replica sees all three updates. For A to either push the data upward or evaluation a trigger (that potentially needs all three updates), A thus must first read from a SQS quorum. This ensures that the latest data is pushed above the tree.

7.2.3 Choosing SQS Parameters

Applications configure IrisNet by specifying an *availability target*, a *bi-exponential failure correlation model*, as well as a *cost function*. IrisNet decides to use the appropriate number of replicas n and quorum size m required to achieve the target availability.

The correlation model can be specified by saying that the deployment context is "PlanetLab-like," "WebServer-like," or "RON-like." In these cases, IrisNet will use one of the three built-in failure correlation models from Chapter 6. We also intend to add more built-in failure correlation models in the future. To provide more flexibility, IrisNet also allows applications to directly specify the three tunable parameters in the bi-exponential distribution. It is our long term goal to extend IrisNet to monitor failures in the deployment, and automatically adjust the correlation model if the initial specification is not accurate.

The cost function is an application-defined function that specifies the overall cost resulting from performance overhead and inconsistency (for read/write data). It takes three inputs, m, n, and i (for inconsistency), and returns a cost value that the system intends to minimize given that the availability target is satisfied. For example, a cost function may bound the storage overhead (i.e., n/m) by returning a high cost if n/m exceeds certain threshold. Similarly, the application can use the cost function to ensure that not too many nodes need to be contacted to retrieve the data (i.e., bounding m). We choose to leave the cost function to be completely application-specific because the requirements from different applications can be dramatically different.

With the cost function and the correlation model, IrisNet uses online simulation to determine the best values for m and n. It does so by exhaustively searching the parameter space (with some practical caps on n and m), and picking the configuration that minimizes the cost function while still achieving the availability target. The amount of inconsistency (i) is predicted [Yu03, Yu04] based on the quorum size. Finally, this best configuration is used to instantiate the system. Currently, IrisNet does not allow the configuration to change on the fly. Our simulator takes around 7 seconds for each configuration (i.e., each pair of m and n values) on a single 2.6GHz Pentium 4; thus IrisNet can perform a brute-force exhaustive search for 20,000 configurations (i.e., a cap of 200 for both n and m, and $m \leq n$) in about one and a half days. Many optimizations are possible to further prune the search space. For example, if $\{n = 32, m = 16\}$ does not reach the target, then $\{n = 32, m = 17\}$ replication can never reach the target either. Exploring such optimizations is part of our future work. This exhaustive search is only performed at system initialization time; its overhead does not affect system performance.

7.2.4 Improving Data Freshness in SQS

In this section we discuss two new techniques for reducing SQS's data inconsistency: data refresh and read auditing.

Transient node or network failures can cause data inconsistency in IrisNet. Consider a scenario with 5 replicas: A, B, C, D, and E, and an SQS quorum size of 2. If A and B are temporarily unavailable, data will be written to C and D. Later suppose C crashes (or gets partitioned from the network) and A recovers. Now, to ensure consistency, we need to refresh the object, so that A and D have fresh data and the reader can read it. The original design [Yu04] of SQS targets scenarios where the read happens not long after the write, so that the probability of such a scenario is negligible. In IrisNet however, a user may pose a query long after the data is written by the sensor, and replica failures/recoveries may result in a scenario where some of the first m replicas have stale data. As a result, we need to monitor the status of the replicas so that we always try to ensure the first m live replicas have the fresh data.[2] Specifically, each live replica monitors the preceding (according to the same order as used in SQS) live replica and ensures that this replica's data is at least as fresh. The very first live replica monitors the status of all replicas in the replica group, and rewrites the data to the first m live replicas whenever failures or recoveries occur. In IrisNet, these *data refresh checks* are performed every 30 seconds.

In addition, we observe that stale reads can often be detected later, when a replica receives an update it previously missed. We incorporate into IrisNet a *read auditing and recall* mechanism, which can be used to inform users when a stale read is detected. To achieve this, each replica maintains the log of processed reads and corresponding responses over the past 10 minutes. If the replica later receives an update (e.g., by the data refresh mechanism) that would have affected a read, it notifies the corresponding user. Stale reads will be detected as long as multiple mismatches do not last longer than 10 minutes and the replicas with the logs and the fresh data do not all crash within that period.

7.3 Regeneration in IrisNet

The replica regeneration protocol in IrisNet is a simplified version of RAMBO's protocol [LS02] with some small modifications. Below we provide a brief description, focusing

[2]Note that this refresh is fundamentally different from replica regeneration (in the next section) and we do not need to invoke a consensus protocol.

```
public class ReplicaGroup {
   int sequenceNum;
   Node[] replicas;
   long TTL;
}
public class Token {
   String objectID;
   // List of replica groups sorted by sequenceNum
   ReplicaGroup[] repGroupList;
}
```

Figure 7.2: A regeneration token.

on the differences between IrisNet and RAMBO.

The reads and writes in IrisNet use a much simplified protocol than RAMBO's two-phase protocol that serves to achieve *linearizability* [HW90] for reads and writes. Because updates from sensors are all commutable, IrisNet does not attempt to serialize writes. Thus in IrisNet, a read simply reads from a read quorum and a write writes to a write quorum.

Regeneration is tricky in the face of false failure detection. For example, two replicas may simultaneously believe the failure of each other and then regenerate independently. This may result in two disjoint new replica groups for the same object. To avoid such scenarios, as in RAMBO, the old replica group for an object uses the Paxos consensus protocol [Lam98] to guarantee that the new replica group for that object is unique. Whereas RAMBO had only one replica group system-wide, IrisNet has one replica group per object because different objects may reside on different sets of nodes. Paxos requires a majority of the old replica group to be up and to coordinate with each other. It is important to note that this majority can be different from the quorum system we use for data access. In IrisNet, data access uses the SQS approach discussed above, whereas regeneration uses Paxos (which requires a majority). Because generally, $m < n/2$, there are scenarios where the replica group is still available but it cannot regenerate.

After regeneration, care must be taken so that the old replica group is properly retired, otherwise it is possible that a reader uses the old replica group while a writer uses the new replica group. In RAMBO, old replica groups are explicitly garbage collected using the data access quorums. Any later reads or writes will see such information from their quorums, and thus realize that the replica group has expired.

In IrisNet, because we use SQS for accessing data and SQS does not always guarantee intersection, directly adopting RAMBO's design would potentially result in inconsistency regarding whether a replica group has expired. Thus we use the following design that utilizes loosely synchronized clocks on replicas. Every replica group, once established, has a time-to-live (TTL) of one day in IrisNet. We use a token (Figure 7.2) to denote the list of currently unexpired replica groups for a given object. A new token is created as the result of the Paxos protocol by appending the new replica group at the end of the list. The token is then published into IrisNet's naming layer. A token is *valid* if its TTL for the latest replica group has not expired. For a read-write to access the data, the reader or writer must obtain a valid token and then access a quorum from *every* unexpired replica group in the token. To ensure that there always exist valid tokens for an object, Paxos is executed once every 12 hours even when there is no need for regeneration.

Requiring the reader or writer to access a quorum from all unexpired replica groups may appear suboptimal. However, from a practical perspective, since regeneration occurs only when there are failures, the list of unexpired replica groups is likely to be short. Further, the different replica groups will have many replicas in common, so the cost of accessing one quorum from k replica groups is likely to be much smaller than the cost of accessing k disjoint quorums.

7.3.1 Replica Regeneration Optimizations

IrisNet is the first system that runs multiple concurrent instances (one for each failed replica group) of the Paxos protocol in WAN. Since Paxos is a heavy-weight protocol requiring coordination with a majority of the replicas in a replica group, concurrent execution of many instances of Paxos raises several practical problems. We here describe two major challenges.

Flooding Problem

After a large correlated failure, one instance of the Paxos protocol needs to be executed for each of the objects that have lost at least one replica. For example, our IrisLog application has 3530 objects storing PlanetLab sensor data, and each object has 7 replicas. A failure of 42 out of 206 PlanetLab nodes has been observed to flood the system with around 2,478 instances of Paxos. Due to the message complexity of Paxos, that many instances incur excessive overhead and stall the entire system.

Positive Feedback Problem

Determining whether a distant node has failed is often error-prone due to unpredictable communication delays and losses. Regeneration activity after a correlated failure increases the inaccuracy of failure detection due to the increased load placed on the network and nodes. Unfortunately, inaccurate failure detections trigger more regeneration which, in turn, results in greater inaccuracy. This positive feedback loop can easily crash the entire system.

To address these two problems, we design and implement the following two optimizations.

Opportunistic Paxos-Merging

One way to avoid the flooding problem would be to group the objects residing on exactly the same set of nodes into *clusters* and use one replica group per cluster. Then we could reduce the number of Paxos instances by invoking one Paxos for each cluster instead of for individual objects. However, it may be necessary to split and merge clusters for balancing load among nodes (details in the next section). The clustering needs to be dynamic such that when a cluster grows too large for some overloaded node, it can *split* the cluster into smaller clusters and shed load by transferring a cluster to another suitable node where it may get *merged* with other clusters. However, since Paxos is run in the granularity of clusters, the following invariant is required to guarantee the correctness of the regeneration: an object can not be part of more than one clusters. This in turn implies that a consensus needs to be run (e.g., by invoking Paxos) among the relevant nodes before every merge/split operation such that they all can synchronously do the same operation, introducing significant complexity into the already intricate RAMBO protocol.

To avoid these overheads, IrisNet instead uses *opportunistic Paxos-merging*. Paxos is still invoked for individual objects. However, if multiple such objects happen to reside on the same set of replicas, the regeneration module merges all the related Paxos invocations into one. The difference from the above clustering approach is that replica groups are maintained in the granularity of individual objects, and only the Paxos invocations are merged lazily, if possible. This avoids the need to maintain consistent split and merge operations on clusters and lets each node independently take load balancing decisions. This approach is particularly effective in the context of our fragmentation algorithm (see Section 7.4), which tends to place two objects either on exactly the same set of replicas or on completely disjoint sets of replicas.

Paxos Admission-Control

To avoid the positive feedback problem, we use a simple admission control mechanism on each node to control its CPU and network overhead, and to avoid excessive false failure detections. Specifically, each node, before initiating a Paxos, samples (by piggybacking on the periodic ping messages) the number of ongoing Paxos instances on the relevant nodes. Paxos is initiated only if the average and the maximum number of ongoing Paxos instances are below some thresholds (2 and 5, respectively, in our current IrisLog deployment). Otherwise, the node queues the Paxos and backs off for a small random time before trying again. With admission control, a Paxos invocation may be delayed. Right before a delayed Paxos is started, IrisNet rechecks whether regeneration is still necessary (i.e., whether there are still failures in the replica group). This helps to further improve failure detection accuracy, and also avoids regeneration when the failed replicas have already recovered.

7.4 Load Balancing in IrisNet

The replication and regeneration design described so far is the most effective to cope with crash-failures. Another important source of correlated failures in IrisNet is simultaneous overload of nodes due to query or update bursts. In this section, we describe how IrisNet uses load-shedding to guard against such overload-failures. We focus on CPU overload and network overload in our discussion, though our techniques can be applied to other resources.

At a high level, an overloaded IrisNet node sheds load by transferring portions of its local database to lightly-loaded nodes. This help reducing load in two ways. First, the local database access overhead, which is roughly proportional to the local database size,[3] is reduced. Second, the overloaded node no longer sees the queries and updates involving the transferred portion of the local database. The whole load-balancing process involves answering the following questions:

- **Reaction:** *When* should a node start shedding load?
- **Selection:** *Which* set of objects does the overloaded node transfer to other nodes?

[3]Current XML database engines [Apa01, Bel03] use the Document Object Model (DOM) that processes queries and updates in-memory. An update, for example, requires first reading the entire XML document and building an in-memory tree structure. After the update is applied to the tree, it is then converted to an XML document and written to disk. The whole operation takes time roughly proportional to the size of the XML document. We anticipate that future XML database engines will not suffer from this limitation.

- **Placement:** *Where* are the transferred object's placed?

As we will show, the constraints posed by IrisNet's hierarchical data indexing and query processing make answers to the above questions, particularly the last two, non-trivial. In the rest of the section, we elaborate on these components. A more detailed description can be found in [NGS05].

7.4.1 Reaction

Each node in IrisNet maintains exponentially weighted moving averages (EWMA) of the rates of queries and updates to each object in its database. Each node is configured with two thresholds, given in terms of the rates of queries and updates it can support without severely degrading the quality of service. An overloaded node starts load shedding when the load is above a *high-watermark* threshold until the load goes below a *low-watermark* threshold. Such use of two thresholds provides stability to the load balancing process.

7.4.2 Selection

To minimize CPU and network overhead for processing queries and updates, the objects transferred from a local database must be carefully chosen. A query requires accessing a set of XML elements in a given (partial) order, in order to properly evaluate the predicates and wildcards in the query. For most queries, this is a top-down order in which parent elements are accessed before their child elements. For example, in Figure 7.1, a query on all the MIT data requires first accessing the `MIT` element and then accessing its children elements (`MIT-1`, `MIT-2`, and `MIT-3`). Now, if the node containing all the MIT data (i.e., the rightmost shaded node) transfers its `MIT` element, the same query will generate three new subqueries, one for each child element, sent on the network and therefore three additional database accesses.[4] On the other hand, if the `MIT-1` element is transferred instead, that will result in only one new subquery and one more database access.

The Problem

More formally, the local database fragmentation problem can be stated as follows: *Given the global database, replica count of each object, the nodes where data can be placed, and the capacities (storage and load) of the nodes, adapt to the dynamic workload by determining in an online fashion (1) the fragments of the global database, possibly overlapping,*

[4]Each subquery results in one independent database access. With replication, these numbers are multiplied by the quorum size.

to be placed at different nodes, and (2) the assignments of these fragments to the nodes such that the capacity of each node is respected, the replica count is maintained for each object, the average query latency is low, and the wide-area traffic is low.

Since the global database can be modelled as a tree, selecting fragments and assigning fragments can be abstracted as a graph partitioning problem and a graph embedding problem respectively. Because of the complex data access patterns of sensing applications, the fixed capacities of individual nodes, and dynamic read-write workloads, the problem is nontrivial. Even dramatically simplified versions of this problem are NP-hard. For example, even when there is an unbounded number of nodes, all nodes have the same capacity C, all pairs of nodes have the same latency, the query workload is known, and there are no database writes, the problem of fragmenting a global database into a set of fragments of size at most C, such that the average query latency is below a given threshold, is NP-hard.[5] Likewise, the simplified problem of assigning a collection of fragments to a set of available nodes, such that either the average network latency or the wide area traffic is below a given threshold, is NP-hard.[6]

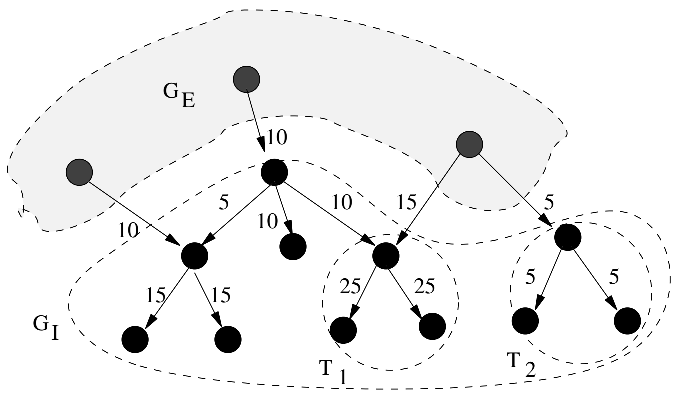
The POST Fragmentation Algorithm

Many approximation algorithms have been proposed for graph partitioning and graph embedding problems (e.g., in the VLSI circuit optimization literature [KK98, SW03]). None of these proposed solutions address the complex problem we consider. However, we get the following two (intuitive) insights from the existing approximation algorithms, which we use in IrisNet. First, the optimal partitions are highly-connected clusters. If, as in our case, the given graph is a tree, each partition is a subtree of the graph. Second, if the edges are weighted (in our case, the weights reflect the frequency in which a hop in the object hierarchy is taken during query routing), and the objective is to minimize the cost of the edges between partitions, most of the highly-weighted edges are within partitions.

To develop a practical solution, we make two simplifications. First, we permit each node to fragment its local database independently, based on its local load statistics. This may result in suboptimal global fragmentation, but our placement heuristics (Section 7.4.3) try to offset this negative effect and our evaluation shows that the final fragmentation produced by our local algorithm is reasonably good. Second, we use heuristics to reduce the computational overhead of local fragmentation. The goal is to partition the local database (represented as a tree) to minimize the wide-area traffic and to make local

[5] Can be shown by a reduction from the Knapsack Problem.

[6] Can be shown by a reduction from the Hamiltonian Path Problem.



G_I represents the local XML fragment of the node and G_E is a set of XML elements on other nodes. The edges and nodes are labelled with the load on the corresponding edge. The circles labelled T_1 and T_2 represent two partitions of size 3.

Figure 7.3: The workload graph of a node.

load below a threshold. However, previous tree partitioning algorithms [Luk74, SW03] tend to incur high computational costs with their $O(n^3)$ (n = number of objects) complexity, and hence prevent a node from shedding load promptly. With a 3 GHz machine with 1 GB RAM, the algorithms in [Luk74, SW03] take over an hour to partition a fragment with 1000 objects. Such excessive computational overhead would prevent IrisNet from shedding load in a prompt fashion. On the other hand, trivial algorithms (e.g., the greedy algorithm in Section 7.5.5) do not yield "good" fragmentation. To address this limitation, we exploit properties of typical query workloads to devise heuristics that provide near optimal fragmentation with $O(n)$ complexity. We call our algorithm POST (*Partitioning into Optimal SubTrees*). As a comparison, under the same experimental setup used with the previously mentioned optimal algorithms, POST computes the result in a few seconds. Below we describe the algorithm.

We use the following terminology in our discussion. For a given node, let G_I denote the set of (*I*nternal) objects in the local database, and G_E denote the set of non-local (*E*xternal) objects and the set of query sources. Define the *workload graph* (Figure 7.3) to be a DAG where the nodes are the union of G_I and G_E, and the edges are pointers connecting parent objects to child objects in the XML database and sources to objects. Under a given workload, an edge in the workload graph has a weight corresponding to the rate of queries along that edge. The weight of a node in G_I is defined as the sum of the weights of all its incoming edges (corresponding to its query load) and the weights of all its outgoing edges to nodes in G_E (corresponding to its message load).

For any set T of objects within G_I, we define T's *cost* to be the sum of the weights of nodes in T. The $cut_{internal}$ of T is the total weight of the edges coming from some node in G_I to some node in T, and it corresponds to the additional communication overhead incurred if T were transferred. The $cut_{external}$ is the total weight of the edges coming from some node in G_E to some node in T, and it corresponds to the reduction of load on the node if T were transferred. In Figure 7.3, the $cut_{internal}$ of T_1 is 10, while the $cut_{external}$ is 15.

The node passes a constant C to POST to indicate the *maximal* cost T may have. The value of C is determined by the extra load that other nodes may take. POST tries to find a "good" T under such constraints. Intuitively, we may want to minimize $cut_{internal}$ (achieved by T_2 in Figure 7.3) so that it introduces the minimum number of additional subqueries or maximize $cut_{external}$ (achieved by T_1 in Figure 7.3) so that it is the most effective in reducing external load.

To design an efficient fragmentation algorithm in IrisNet, we exploit the following important characteristics in the workload: *A typical monitoring query in a hierarchical database accesses all objects in a complete subtree of the tree represented by the monitoring database, and IrisNet routes the query directly to the root of the subtree.* This observation is well supported by the real IRISLOG query trace (more details in Section 7.5.5), which shows that $> 99\%$ of user requests select a complete subtree from the global database. Moreover, users make query on all the levels in the hierarchy, with some levels more popular than the others. Under such access patterns, the optimal T is typically a subtree. The reason is that transferring only part of a subtree T from a node N_1 to another node N_2 may imply that a top-down query accesses objects in N_1 (the top of T) then in N_2 (the middle of T) and then back in N_1 (the bottom of T), resulting in a suboptimal solution.

The above observation enables POST to restrict the search space and run in linear time. POST sequentially scans through all the nodes of the workload graph, and for each node it considers the whole subtree rooted at it. For all the subtrees with cost smaller than the given capacity C, it outputs the one with the optimal objective. The search space is further decreased by scanning the nodes in the workload-graph in a bottom-up fashion, thus considering the lower cost subtrees near the leaves before the higher cost subtrees further up the tree. As mentioned before, in typical settings, POST takes a few seconds to run. Yet, as we will show in Section 7.5.5, the quality of the resulting fragmentation in practice is very close to that of the $O(n^3)$ optimal algorithms that take tens of minutes to run.

Finally, we note that each time POST decides to transfer a fragment T from the overloaded node to a lightly-loaded node, it must coordinate with all the relevant replicas. Specifically, for each object in T, a consensus protocol must be run among the replicas for that object, in order to drop the old node from the replica group and add the new node.

7.4.3 Placement

The simplest approach to place a fragment it to select a random node capable of taking the extra load. For better performance, we use two heuristics that exploit IrisNet's query and data source characteristics to improve the overall performance.

Our first heuristic uses our previous observation that subtrees of the global database should be kept as clustered as possible. Therefore, IrisNet first considers the *neighboring* nodes as possible destinations for a fragment. A node N_1 is a neighbor of fragment f if N_1 owns an object that is a parent or child of some object in f. Thus, a node N_2, trying to split or replicate a fragment f, first sees if any of the neighboring nodes of f can take the extra load. If such a node N_1 is found, the fragment is sent to it. If more than one neighboring nodes are capable of taking the extra load, then the one having the highest *adjacency score* is chosen. The adjacency score of a node N_1 with respect to the fragment f is the total weight of the edges from any object in f to any object in N_1. This heuristic helps maintaining large clusters throughout the system.

When no neighboring node is found, IrisNet's second heuristic searches for any node which is capable of taking the extra load of the fragment f to transfer and is the closest to the *optimal location*. The optimal location for an object is the weighted mid-point of its read and write sources. We account for the location of sources using the GNP [NZ02] network mapping system. Specifically, each IrisNet node has corresponding GNP coordinates and the cartesian distance between hosts is used to estimate the latency between them. If the read and write loads of an object are R and W, and the average read and write source location coordinates are GNP_{read} and GNP_{write}, then the optimal location is given by $GNP_{opt} = (R \cdot GNP_{read} + W \cdot GNP_{write})/(R+W)$. The optimal location of a fragment is the average of the optimal locations of all the objects in the fragment. This heuristic tends to place objects near the sources of reads and writes.

7.4.4 A Simple Run

We illustrate a simple run of IrisNet's load balancing in Figure 7.4. Rectangular boxes represent nodes and the trees represent their local databases. Initially the global database is placed at one node (1), which experiences high read and write load. To shed write load the node then determines two fragments (one containing object 5 and the other containing objects 3 and 6) for splitting. The fragments are then placed at newly-discovered nodes

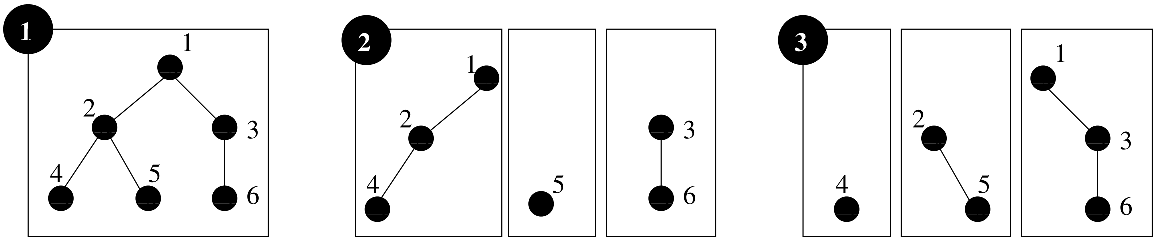


Figure 7.4: A simple adaptive data placement scenario.

near the write sources (2). To shed read load, it then determines two more fragments and places them in nodes already owning objects adjacent to the fragments (3).

7.4.5 Related Work

The general load balancing problem involves *fragmentation*—dynamically replicating and partitioning the contents of nodes, and *placement*—allocating the replica/partitions on available nodes. These problems have been extensively studied by the theory community (see [DF82] for a survey). The general problem has been found to be NP-Hard [WM91]. Practical approximate solutions for this problem can be categorized into two classes. The offline solutions assume that the complete workload, set of nodes, and their capacities are known a priori. Such solutions have been proposed in the context of theory [KDW01], distributed databases [Ape88, KP97, LY80], read-only content distribution networks [KR01], the Web [QPV01], etc. The online solutions dynamically replicate/partition and place data on available nodes as the workload is applied. Such solutions have been proposed in the context of theory [ABF93, WJH97], distributed databases [BLS95, SAS+96b], file systems [SKKM02], general storage systems [LA00], etc.

In [NGS05], we have provided a more comprehensive list of related work and shown that the overall load balancing problem is significantly more complex for a typical sensing system. The complexity comes from its data source characteristics (e.g., write-intensive data) and query characteristics (e.g., hierarchical in-network aggregation). As we have shown in [NGS05], existing solutions are not very effective in the context of sensing systems.

7.5 Evaluation

In this section, we evaluate the techniques presented in this chapter to answer the following questions.

- How well does IrisNet react to a single large (untargeted) crash-failure?
- How well does our implementation perform in a real deployment?
- What is the long-term availability of IrisNet?
- How much inconsistency does IrisNet experience because of its use of SQS?
- How effective is Post in tolerating correlated overload-failures?

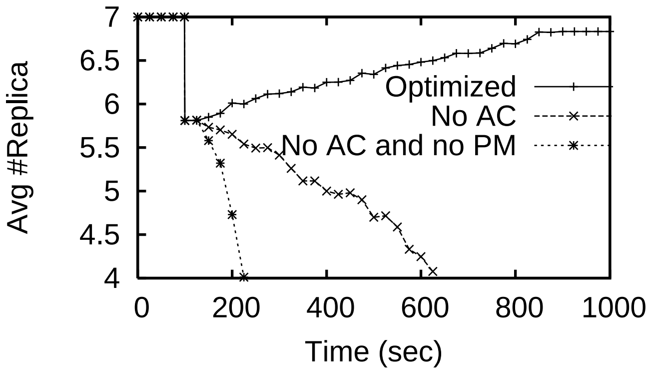
We use a real deployment of IrisLog, an IrisNet application, to answer the first two questions. Our experiments use IrisLog deployed on 200+ PlanetLab nodes. The last three questions are answered by using a combination of PlanetLab deployment, emulation, and simulation with real-world traces and failure models.

Throughout our evaluation, we use IrisLog(m) as the legend for IrisLog with a quorum size of m. Unless otherwise stated, we use 7 replicas for each object. In some experiments, we compare IrisLog with a hypothetical application Majority that uses majority quorum systems with the same number of replicas as IrisLog. Some of our experiments are based on replaying `PL_trace` (the PlanetLab failure trace described in Chapter 6) in either a PlanetLab or an EmuLab [Flu05] deployment of IrisLog. To replay a long trace in a reasonable amount of time, we speed up the replay by compressing the timescale of the trace. Namely, for two events that are t time apart such that $t > t'$, where t' is the time needed for the system to stabilize, we replay the events time t' apart.

7.5.1 Individual Crash-Failures

We here present IrisNet's behavior (in terms of availability, regeneration overhead, etc.) just after a single large failure. We use the event on 3/28/2004 that caused 42 out of the 206 live PlanetLab nodes to crash in a short period of time (an event found by analyzing `PL_trace`). We replay this event by deploying IrisLog on 206 PlanetLab nodes and then killing the IrisLog process on 42 random nodes.[7] The assignment of objects to nodes are based on running Post against the real IrisLog query trace (described in Section 7.5.5). Our IrisLog deployment on PlanetLab has 3530 objects representing different sensor data collected from different PlanetLab nodes. In steady state, each of these objects has 7 replicas.

[7] At the time of this experiment, a total of 218 PlanetLab nodes were up and we randomly chose 204 of them. Note that these are not the same set of nodes available during the original failure event we emulate.



In this experiment, we inject a large failure at time=100 in our PlanetLab deployment of IrisNet. After that, replica groups start automatic regeneration. The graphs shows that the regeneration process does not converge without our optimizations.

Figure 7.5: Average number of replicas per object during the regeneration after a large failure in PlanetLab. (AC = Admission-Control, PM = Paxos-Merging.)

Effectiveness of Paxos Optimizations

Figure 7.5 plots the average number of replicas per object and shows how the system gradually regenerates failed replicas after the failure at time = 100s. Before the failure, the average number of replica is 7. The failure causes the failed nodes to lose the replicas they store, and so the average number of replicas suddenly decreases to 5.8. In particular, the failure affects 2478 of the replica groups. We can easily see that without Paxos admission-control or opportunistic Paxos-merging, the system enters a vicious cycle of regeneration → overload → false failure detection → more regeneration, the regeneration process does not converge, and the average number of replicas actually drops.

Paxos-merging reduces the total number of Paxos invocations from 2478 to 233, while admission-control helps the regeneration process to converge. Note that the average number of replicas does not reach 7 even at the end of the experiment because some replica groups (total 140) lose a majority and cannot regenerate. Moreover, the positive feedback effect still appears in our experiment which explains the fact that the average number of replicas occasionally drops slightly (e.g., at time = 420). In particular, we noticed 19 unnecessary regenerations (due to inaccurate failure detection), which can be reduced by tuning the admission control parameter as a tradeoff with the overall convergence time.

Table 7.2: Breakdown of the average regeneration time.

Task	Time (sec)
Paxos	7.28
Data copy	14.28
Total	21.57

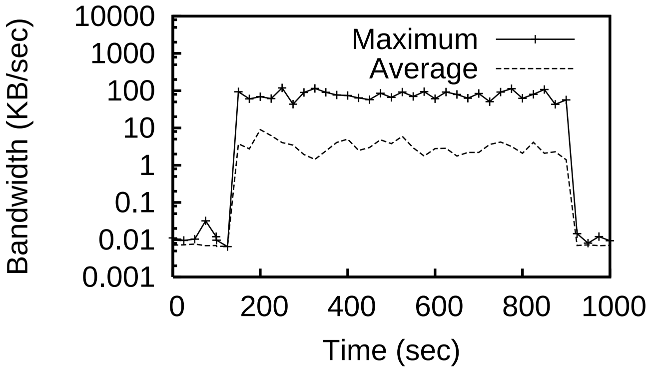


Figure 7.6: Bandwidth consumption during the regeneration after a large failure in PlanetLab.

Paxos Overhead

Table 7.2 breaks down the time required by different phases of the regeneration process. A single regeneration takes around 21.6 seconds on average which is dominated by the time for data transfer (14.3 seconds) and Paxos (7.3 seconds). The time for data transfer is determined by the amount of data and can be much larger. The convergence time of around 12 minutes is largely determined by the parameters of the admission control mechanism and may be further improved. However, regeneration is typically not a time-sensitive task, because we only need to finish regeneration before the next failure hits. Our simulation study based on `PL_trace` shows that 12-minute regeneration time achieves almost identical availability as say, 1-minute regeneration time. Thus we believe IrisLog's regeneration mechanism is adequately efficient.

Figure 7.6 shows the bandwidth used during regeneration. We observe that on average, each node only consumes about 3KB/sec bandwidth, out of which 2.8KB/sec is used to perform necessary regeneration (including data transfer), 0.27KB/sec for unnecessary regeneration (caused by false failure detection), and 0.0083KB/sec for failure detection (pinging) (Table 7.3). The worst-case peak for an individual node may reach 100KB/sec, which is however still sustainable with even home DSL links.

Table 7.3: Breakdown of the bandwidth usage during the regeneration phase ($time = 100$ to 720 in Figure 7.6).

Task	BW per node (KB/sec)
Periodic Ping	0.0083
Necessary Regeneration	2.8176
Unnecessary Regeneration	0.2672

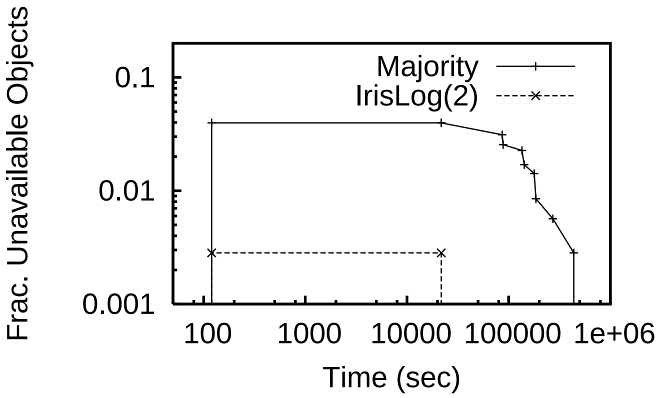
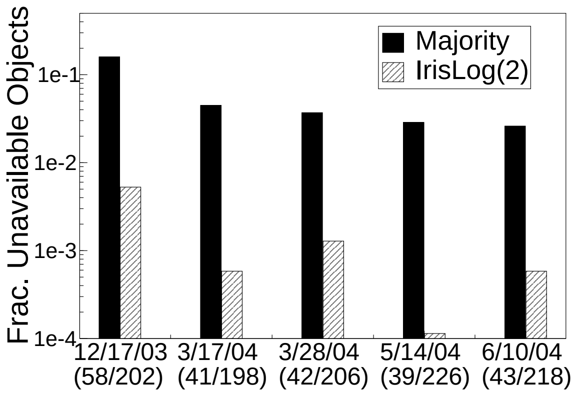


Figure 7.7: Fraction of unavailable objects after a large failure in PlanetLab

Availability

Figure 7.7 shows the fraction of unavailable objects (not having live quorums) in IrisLog(2) and Majority. IrisLog has fewer unavailable objects than Majority because of its smaller quorum size. Note that regeneration only helps to restore the replica number for available objects (to guard against additional failures). For an unavailable object to become available, some failed replica of that object needs to recover so that Paxos may regain a majority. To demonstrate the effect, we replay `PL_trace` for around 6 more days after the failure. As shown, the first such recovery happens at around 20000 seconds. The availability increases with time as more and more nodes come up. However, IrisLog is affected less by the failure, it offers 100% availability long (around five days) before Majority does.

To further understand the robustness of IrisLog under large correlated failure events, Figure 7.8 plots how Majority and IrisLog are affected by a number of large correlated failures in PlanetLab. These failures are caused by several reasons including DoS attacks (e.g., on 12/17/03), software bugs and maintenance purposes (e.g., on 3/17/04), node overloads (e.g., on 5/14/04), etc. As shown in the graph, in all cases, the total number of unavailable objects in Majority is at least one order of magnitude larger than that



The x-axis shows the dates of the events and the corresponding (number of nodes died) / (total live nodes before the failure)*. For example, just before the failure event on 12/17/2003, 202 PlanetLab nodes were alive of which 58 died because of the event.*

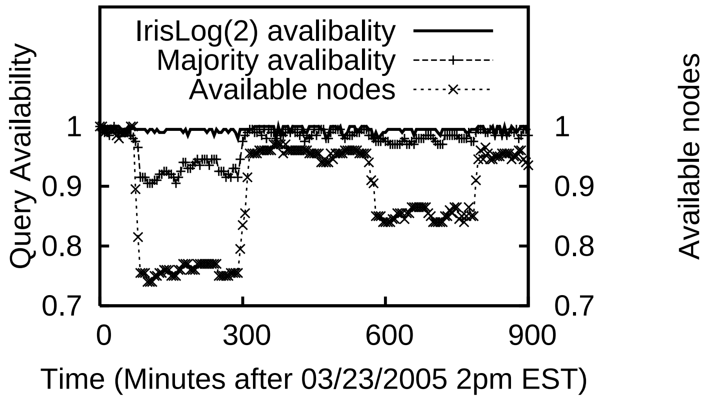
Figure 7.8: Availability of Majority and IrisLog, under several large correlated failure events of `PL_trace`.

in IrisLog.

7.5.2 IrisLog in the Wild

To demonstrate the robustness of our design and implementation in a harsh deployment environment, we here report IrisLog's availability over a 15-hour period, 2 days before the SOSP 2005 conference deadline (03/25/2005). This was one of the busiest periods of PlanetLab, with many researchers running their experiments on it to meet the conference deadline. Nodes were highly overloaded and many of the live nodes appeared to be dead to IrisLog users (because the nodes failed to respond within a reasonable time). Note that this is an unusually high failure-prone period; PlanetLab is relatively more stable during other times. We intentionally chose this harsh period because this enabled us to experience many failures in a short period of time. Moreover, this will test IrisLog's resiliency in the wild.

At the beginning of this experiment, IrisLog was running on around 200 nodes. During the experiment, we issued one query every five minutes. Each query involved all the objects stored in IrisLog's distributed database. Some queries failed to access some of the nodes, because either the nodes had crashed/partitioned or were so overloaded that subqueries timed out before receiving the expected data. We use 15 seconds as the timeout period. We compare IrisLog(2) and Majority, and each query returned the number of IrisLog objects accessible by these two schemes.



This graph shows the availability of IrisLog *deployed on PlanetLab, during a period when PlanetLab was excessively being used by users. It also shows the number of nodes accessible within a timeout period at different times. Query availability denotes the fraction of relevant objects returned by a query.*

Figure 7.9: Availability of IrisNet in the wild.

Figure 7.9 shows the fraction of available (i.e., accessible by IrisLog subqueries) nodes and the fraction of available (i.e., with accessible quorums) objects under IrisLog(2) and Majority. We observed two reasonably large correlated failures in this 15 hours period ($\approx$ 45 nodes after $\approx$ 1 hour and $\approx$ 25 nodes after $\approx$ 10 hours failed in short time). IrisLog(2) could tolerate these failures well, as shown by very small degradation of its availability. In contrast, Majority performs poorly in tolerating such failures. The experiment also shows that IrisLog was relatively stable during this period, and could regenerate and balance load automatically as evidenced by its high availability. Note that, according to `PL_trace`, IrisLog(2) is configured to achieve around 99.9% availability. Although the instantaneous availability, as shown in this graph, sometimes goes as low as 98%, the long term availability (i.e., averaged over time) reaches the target 99.9%. This is evidenced by our availability measurement over a two-month long period; Figure 7.9 represents a subset of this entire period.

7.5.3 Long-Term Availability under Crash-Failures

This section studies IrisNet's availability dynamics over a long period of time (much longer than the period considered in Section 7.5.2) as nodes in the system continuously fail and recover.

Methodology

Our basic methodology for such evaluation is to replay a stream of failure and recovery events, based on either `PL_trace` or the correlation model. However, doing this in a real system is tricky. The fundamental stumbling block is the requirement of exposing the system to a *significantly*, potentially a year or more, long (real or synthetic) failure trace capturing a reasonable distribution of different types of failures. This is necessary since failures, specially big failures that cause most of the unavailability, are rare events and only a long failure trace can capture most of them to provide a better confidence in long-term availability. For example, a system with slightly less than three nines of availability may have only two five-hour long failure events in a one-year period. A trace spanning few months may miss these events and show the system availability to be 1, a significant over-estimation.

The requirement of replaying long failure traces in real systems introduces two difficulties. First, evaluation takes a long time. Compressing the time scale of failure traces does not help much—even with a compression factor of 100, a one-year long trace would require several days to replay. Worse, in practice, such a large compression factor is not feasible since the system must be given enough time to stabilize before injecting two successive events in the compressed trace. Second, if a real testbed is used for such a long evaluation period, it is not trivial to isolate the effects of the experiment from the effects of the (unpredictable) failures on the testbed during the experiments.

However, we conjecture that an event-driven simulator can avoid the above problems and can evaluate the availability of a replicated system like IrisNet with reasonable accuracy. This is supported by our results in the previous section: network bandwidth and latency do not seem to be limiting factors for system availability, because each node on average consumes only 3KB/sec bandwidth during regeneration and network latency is orders of magnitude lower than regeneration time. This means, the simulation results for Erasure$(2, 7)$ presented in the previous chapter represent the long term availability of IrisNet.

We therefore validate the simulator (and thus the simulation results of Erasure$(2, 7)$ representing IrisNet) we used in the previous chapter. To show that our simulation results match with the results obtained from the real deployment of IrisNet, we take the following steps.

- First, we deploy IrisLog on PlanetLab and measure its long-term availability. The real system constraints limit us to use only a small part of `PL_trace`. Even after compressing the timescale of the trace, replaying a 6-days long trace in PlanetLab takes almost 12 hours.

- Then, we deploy IrisLog in EmuLab. Unlike PlanetLab, the EmuLab nodes communicate over a LAN, which provides much lower latency and higher bandwidth. This lets us replay a longer trace, including the part replayed in PlanetLab. By compressing timescale, we can replay a 30-days portion of the trace in 1 days.
- Finally, we replay the whole `PL_trace` over the event-driven simulator we used in Chapter 6 to compute the long term availability. The simple simulator, written in Java, models only node failures and recoveries, and ignores processing speed, bandwidth, etc. of the nodes.

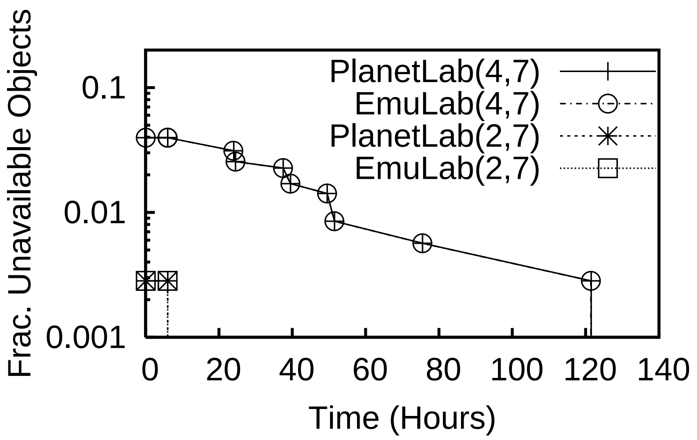
We now show that results from each step match fairly well with the next step, confirming that our simulation results of Erasure$(2, 7)$ in Chapter 6 represent IrisNet's availability in real world.

PlanetLab vs. EmuLab Results

To validate EmuLab experiments, we replay the same 7-day-long portion of `PL_trace` on both PlanetLab and EmuLab deployments of IrisLog. This validation is performed using two different configurations: 7 replicas with a quorum size of 4 (majority voting), and 7 replicas with a quorum size of 2 (SQS). Figure 7.10 plots the fraction of unavailable objects as measured in the two testbeds. Notice that the two curves from the EmuLab emulation almost exactly match the two curves from the PlanetLab deployment. We also inject the same big failures shown in Figure 7.8 in both the deployments, and find the results to be exactly the same for both of them. These results are not surprising since as shown in the previous section, each node on average consumes only 3KB/sec bandwidth during regeneration, and network latency is orders of magnitude lower than regeneration time. As a result, network bandwidth and latency do not tend to significantly impact system availability. Therefore, the emulation results closely match the results from a real deployment of IrisNet.

Emulation vs. Simulation Results

Next, to validate our simulation results, we replay the same 30-day-long portion of `PL_trace` on EmuLab and in our simulator. Figure 7.11 plots the unavailability observed from EmuLab experiments and from simulation. As we can see, the results from simulation match the emulation results nicely. These results are also expected since in our context, availability is minimally impacted by factors such as machine speed, latency,



In this experiment, we replay the same 7-day-long portion of `PL_trace` *in a PlanetLab deployment and an EmuLab deployment of* IrisLog. *The legend PlanetLab(m,n) (or, EmuLab(m,n)) represents* IrisLog *running on PlanetLab (or in EmuLab, respectively) with a configuration where each object has n replicas and the quorum size is m. The corresponding points from the PlanetLab and the EmuLab experiments fall on top of each other. This implies that the EmuLab results closely match the results obtained from the real PlanetLab deployment.*

Figure 7.10: Validation of EmuLab results against PlanetLab results.

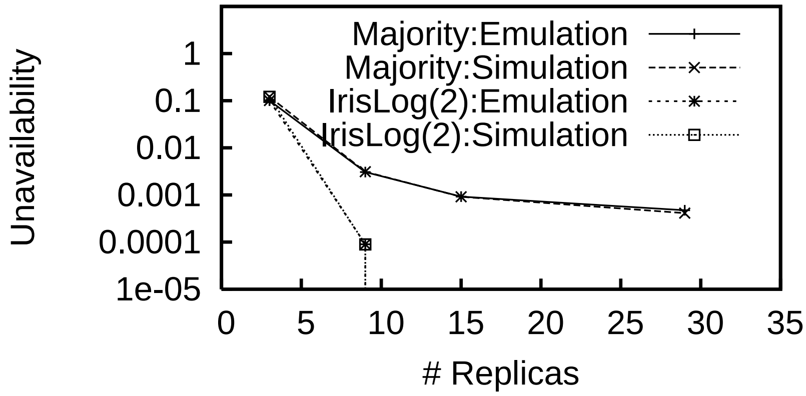
and bandwidth. Thus, the simulation results presented in Chapter 6 closely match with the results found from our emulation (and hence the real deployment).

7.5.4 Inconsistency from SQS

So far we have shown that IrisNet's use of SQS makes it more available than traditional systems (e.g., those using majority quorum systems). However, the benefit comes at the cost of small inconsistency (probability of stale reads), due to the smaller quorums used by SQS. In this section, we experimentally quantify the inconsistency.

There are two causes of inconsistency in an IrisNet application (or, in any SQS-based system). First, because of network failures, a reader may not be able to read from the nodes updated by the sensors. Such inconsistency would persist as long as the reader and the sensors remain partitioned. Second, after a transient node failure, scenarios like the one mentioned in Section 7.2.4 can arise and readers can get stale data. The probability of such inconsistency depends on the refresh rate by the sensors. In IrisNet, data is refreshed every 30 seconds.

To quantify the effects of these two causes, we conduct the following trace-driven simulation with varying quorum sizes. First, we consider only network failures. The



In this experiment, we replay the same 30-day-long portion of `PL_trace` *in our custom event-driven* IrisLog *simulator and in our EmuLab deployment of* IrisLog. *The legend* IrisLog*(m) represents a configuration where each object has 7 replicas and the quorum size is m. The graph shows that results from these two experiments match closely, validating our simulation results.*

Figure 7.11: Validation of simulation results against EmuLab results.

all-pair ping data of `PL_trace` provides us reachability data of any two pairs of nodes. We fix two nodes in `PL_trace` as the reader and the writer, and choose 7 random nodes as the replica of an IrisLog object. We then replay the whole `PL_trace` to compute the inconsistency, the fraction of time the read quorum reachable by the reader does not intersect with the write quorum reachable from the writer. We repeat this with 100 different reader-writer pairs, and report the average inconsistency. The curve labelled "Without Failure" in Figure 7.12 shows the amount of inconsistency as a function of SQS quorum size.[8] To understand the effect of node failures on inconsistency, we repeat the above experiment except that we let the replicas fail according to `PL_trace`. The curve labelled "With Failure" in Figure 7.12 shows the resulting inconsistency. Both these graphs confirm that the amount of inconsistency is extremely small, even with a small quorum size of 2. As described in Chapter 1, sensing applications can easily tolerate such small inconsistency. Further, we observe that our recall mechanism recalls 90% of all the stale reads within 10 minutes.

To understand SQS's inconsistency beyond `PL_trace`, we evaluate it with our correlation model. Figure 7.13 plots inconsistency in IrisLog under different ρ_2 values and quorum sizes m. The "No failures" curve in the graph represents network failures (mis-

[8]The inconsistency in SQS is largely determined by the quorum size m, and is not sensitive to the total number of replicas, since inconsistency means mismatches on *the first m live replicas*.

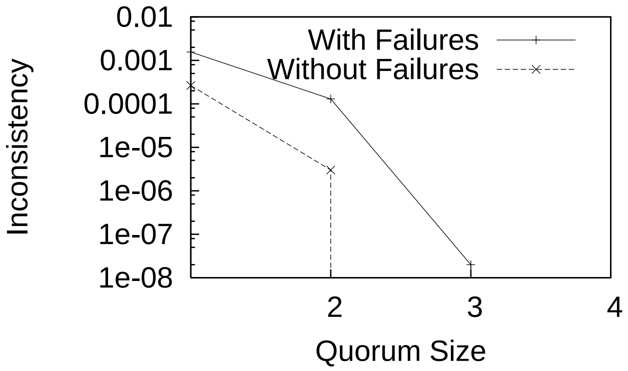


Figure 7.12: Inconsistency of IrisLog under `PL_trace`.

matches) and is generated based on the simple closed-form formula of 0.1^i given in [Yu03] as the probability of having i simultaneous mismatches. The graph shows that larger ρ_2 has a negative effect on inconsistency, because correlated failures result in fewer nodes with fresh data (until the data is refreshed) and, hence, make it more likely for all the nodes with fresh data to fail simultaneously. Once this happens, IrisLog must wait for a relatively long time (i.e., MTTR/m time on average) for one of these nodes to recover. An important observation, however, is that while larger ρ_2's have a devastating effect on the availability of Majority, they have only a modest effect on the consistency (and availability) of IrisLog. The curve for $\rho_2 = 0.9$ is still close to the (idealized) curve where there are no node failures (and hence no dependence on ρ_2) and the only source of inconsistency is reachability mismatches. Even when $\rho_2 = 1.0$, the curve is still decreasing roughly linear in our log-scale graph. We also find that these results are fairly robust to different parameter settings. For example, with small quorum size (≤ 12), increasing the time between data refresh checks from 30 seconds to 10 minutes increases inconsistency only slightly, because MTTR/m is still an order of magnitude larger than the refresh time.

7.5.5 POST under Targeted Node Overload

In this section, we evaluate how Post enables IrisLog to mask targeted overload-failures. We drive our evaluation by using `IrisLog_trace`, a real user query trace collected from our IrisLog deployment on 310 PlanetLab nodes from 11/2003 to 8/2004 (Table 7.4). The trace consists of 6467 user queries, 99.1% of which select complete

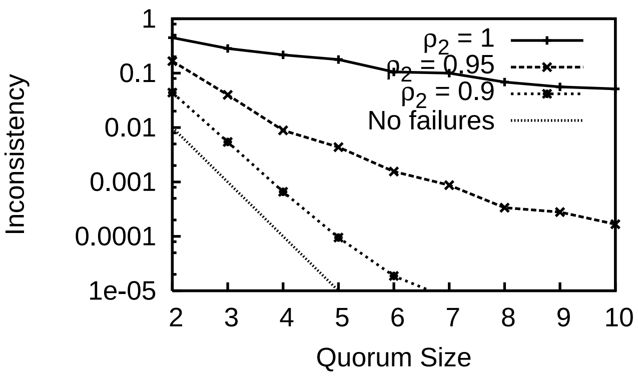


Figure 7.13: Inconsistency of SQS under synthetic traces with different values of the correlation level ρ_2 in the model $G(0.95, 0.45, \rho_2)$.

Table 7.4: `IrisLog_trace`: Trace of user queries from 11/10/2003 to 8/27/2004 for the IrisLog service deployed on 310 PlanetLab nodes.

Total queries	6467 (100%)
Queries selecting a complete subtree	6409 (99.1%)
Queries selecting all nodes	401 (6%)
Queries selecting a country	1215 (19%)
Queries selecting a region	3188 (49%)
Queries selecting a site	1469 (23%)
Queries selecting a machine	136 (2%)
Queries not selecting a complete subtree	58 (0.9%)

subtrees rooted at different levels of the hierarchy.[9]

We compare Post with three other algorithms: Greedy, Local OPT and Oracle. In Greedy, overloaded nodes evict individual objects in decreasing order of their loads. As a result, nodes make decisions in finer granularity, but do not try to keep the objects hierarchically clustered. In Local OPT, each node partitions its local database using an optimal tree partitioning algorithm [Luk74] with $O(n^3)$ complexity. However, because each invocation of Local OPT takes hours of computation time, we do not use it for our live experiments on PlanetLab. Oracle is an offline approach that takes the whole global database and the query workload, and computes the optimal fragmentation (again using the algorithm in [Luk74]). Oracle cannot be used in a real system and only serves

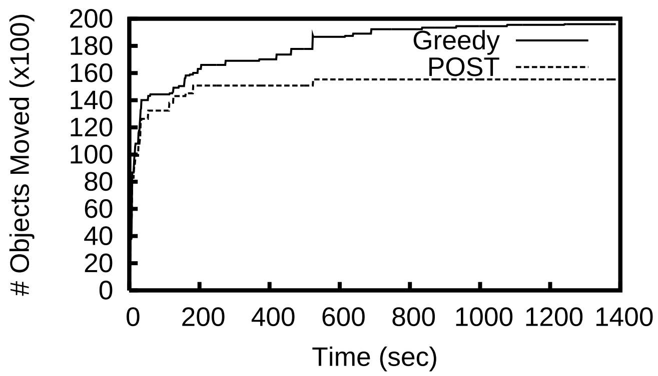
[9]This large fraction of pure hierarchical queries is probably biased by IrisLog's query interface that makes it relatively easier to pose queries over subtrees. However, we believe that queries over subtrees are natural and therefore this fraction would be reasonably high for any typical sensing application.

as a lower bound for comparison purposes.

Our evaluation consists of two phases. In Phase 1, we fragment the IrisLog global database by using all but the last 1000 queries of `IrisLog_trace` as the warm-up data. To do this within a reasonable time, we perform this phase on EmuLab and speed up the replay such that we can finish it in an hour. We start with a single node holding the entire database, and then we inject the workload and let the overloaded nodes dynamically perform fragmentation and place the fragments on other lightly loaded nodes. In Phase 2 of our experiment, we place the fragments resulted from Phase 1 in our PlanetLab deployment and inject the last 1000 queries from `IrisLog_trace`. Each fragment is replicated in 7 nodes, and we use a quorum size of 2. An object is considered available if a query can retrieve it within 30 seconds timeout period (similar results are obtained using a 15 second time-out). For this set of experiments, we define availability to be the percentage of available objects out of the objects required by the queries. To emulate targeted node overload, we increase the replay *speedup factor*, defined as the ratio between the original duration of the trace and the replay time. Because our trace has a low average load, we use a relatively large speedup factor.

Fragmentation Overhead. Figure 7.14 plots the cumulative overhead of the fragmentation algorithms over time during the warm-up phase of the experiment. The overhead is measured as the number of objects transferred over the network due to load-shedding. (The cost of Paxos during the split/merge operations is proportional to the number of split/merge operations.) The graph shows that the amount of fragmentation decreases over time, which means that our load-based invocations of the algorithms do converge under this workload. Greedy incurs higher overhead than Post because Greedy's non-clustering fragmentation increases the overall system load which makes the nodes fragment more often. We do not use Local OPT or Oracle in this experiment due to their excessive computation overhead.

Adaptation Effectiveness. To understand the advantage of Post's adaptive fragmentation, we also use Static-Post which, under all speedup factors, starts from the same warm-up data (generated from Phase 1 with a speedup factor of 1200) and does not further fragment the database during Phase 2. Figure 7.15 shows the unavailability of IrisLog under different fragmentation algorithms and under different speedup factors. Greedy is very sensitive to node overload and suffers from high unavailability even under relatively smaller speedup factors (i.e., smaller load). Post is significantly more robust and satisfies more queries even at a higher speedup factor. This is because Greedy produces suboptimal fragments and overloads the system by generating a large number of subqueries per query (Figure 7.16). The effectiveness of Post comes from its superior



We start this experiment by placing the complete IrisLog *database in one EmuLab node. We then inject all but the last 1000 queries in* `IrisLog_trace`*. As nodes get overloaded, they fragment their local databases and transfer the fragments to other nodes. We use two online fragmentation schemes, and for each scheme, we measure the number of objects transferred between nodes over time.* Post *incurs less overhead and converges faster.*

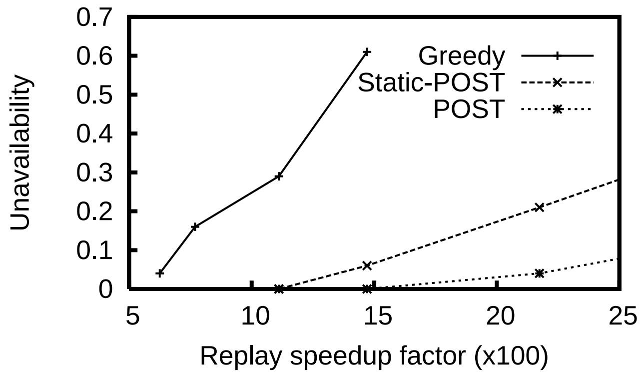
Figure 7.14: Fragmentation cost of different fragmentation algorithms.

choice of fragments, which generate a near optimal number of subqueries (as shown in Figure 7.16). The difference between Static-Post and Post in Figure 7.15 demonstrates the importance of adaptive load-shedding for better robustness against targeted node overload.

7.6 Summary

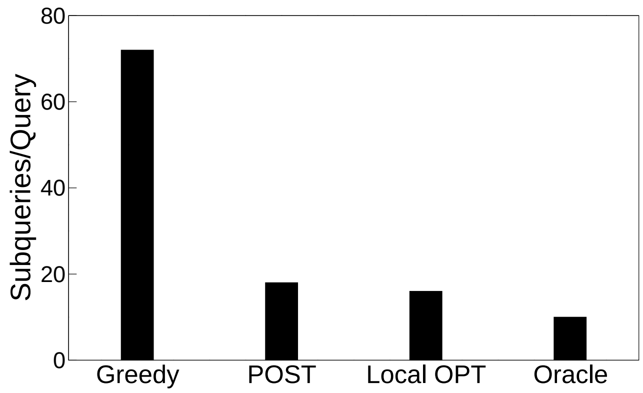
In this section, we have described the design, implementation, and evaluation of the mechanisms IrisNet uses to ensure high availability of its storage layer. We have presented the following components of IrisNet.

- **Replication:** We have exploited sensing applications' tolerance for occasional data inconsistency to use SQS in IrisNet. We have shown how to use SQS with IrisNet's hierarchical database. IrisNet is the first implementation of SQS. We have shown that evaluating availability in a real replicated system is difficult, mainly because it takes a long time. However, availability is only minimally impacted by a system's hard-to-model factors such as latency, CPU load, etc. Therefore, availability can be accurately evaluated with a simple event-driven simulator that ignores these factors.



In this experiment, we consider an object to be unavailable if it can not be retrieved within 8 seconds. An object is unavailable when all of its replicas are overloaded.

Figure 7.15: Unavailability caused by overload with different fragmentation algorithms.



For each query, we count the number of subqueries it generates between nodes. More subqueries are generated if the objects are not clustered and the query needs to access more nodes in its top-down traversal along the hierarchy. Thus, a good fragmentation scheme provides a small number of subqueries.

Figure 7.16: Network overhead per query with different fragmentation algorithms.

- **Regeneration:** We have exploited several sensing-specific properties (e.g., single writer, timestamped data, etc.) to design a simplified version of RAMBO's regeneration protocol for IrisNet. We have, for the first time, shown how to use a large number of concurrent instances of the Paxos consensus protocol in a wide-area system. We have proposed several optimizations that are necessary for the convergence of the protocol.

- **Load balancing:** We have exploited a sensing system's hierarchical database and access patterns to propose a novel and efficient dynamic load balancing algorithm called POST. It produces near-optimal solutions at a significantly smaller overhead compared to locally optimal algorithms.

Replication and regeneration cope with crash-failures, whereas dynamic load balancing guards against overload-failures. They in combination make IrisNet highly available.

Chapter 8

Conclusion

We conclude this thesis with a summary of our contributions and directions for future work.

8.1 Summary

This thesis identifies two fundamental challenges that affect the end-to-end availability of an Internet-scale heterogenous sensing application.

- Robust and energy-efficient data collection from wireless sensor networks.

  Existing in-network aggregation schemes *require* tree topologies for correctness. The inherent fragility of the tree topology makes the schemes susceptible to node- and link-failures. Existing robust solutions sacrifice energy-efficiency by using reliable communication, thus showing a strong tradeoff between energy-efficiency and robustness. Existing adaptive solutions use low-level observations of network characteristics to repair the tree topology, without taking application semantics into account.

- Highly available distributed storage in Internet-connected nodes.

  Replication and regeneration are two standard techniques to build highly available storage systems. However, the availability of a distributed storage system on the Internet is severely limited by correlated failures. Existing storage systems either ignore failure correlation or use simple techniques that fail to provide a target availability without large resource overhead. The fundamental stumbling block (prior to our work) in combatting correlated failures is the poor understanding of their nature and impact on real systems.

8.1.1 Our Approach

We argue that traditional techniques of buying robustness with additional resources (e.g., energy, storage) are not feasible for sensing applications because of their inherent constraints. A more suitable approach is to exploit several deployment- and application-specific properties of sensing systems, which weakens the robustness *and* resource efficiency tradeoffs, which in turn enables robust and resource-efficient sensing systems. We showed the feasibility of this approach through several novel techniques that significantly improve the availability of a sensing system, without significant resource overhead.

We addressed the first challenge above with Synopsis Diffusion, a novel aggregation scheme that *decouples* aggregation algorithms from aggregation topologies. This decoupling enables aggregation over highly robust and energy-efficient multi-path topologies. To adapt to long-term dynamics in the deployment environment, we proposed Tributary-Delta, a novel scheme that enables highly robust aggregation by efficiently combining tree- and multi-path-based aggregation and dynamically adapting topologies according to application-level semantics.

To tolerate correlated failures in Internet-scale storage of sensor data, we used the following solutions. First, to combat crash-failures of nodes, we used recently proposed Signed Quorum Systems (SQS) that, as we showed, are more effective than traditional strict quorum systems in tolerating correlated failures. Second, in determining the configuration parameters of SQS, we used a correlation model that we developed by studying several large real-world systems. Finally, we used a novel load balancing algorithm that exploits sensing applications' hierarchical data indexing and access patterns to quickly shed load from an overloaded node and to avoid overload-failures during a flash crowd.

8.1.2 Contributions

This thesis has made contributions in two major areas. The first area is conceptual and consists of the novel ideas and design principles generated by our work. The second area of contribution is a set of artifacts—a few major components of IrisNet and a number of sensing applications that we implemented to validate some of our ideas.

Conceptual Contributions

In this thesis, we addressed the aforementioned challenges and propose solutions that result in significant improvement in end-to-end robustness of an Internet-scale, heterogeneous sensing system. Because there is a tremendous heterogeneity and diversity in the problems, there is no single panacea for all of them. We recognized this heterogeneity and developed a suite of algorithms, techniques, and design principles to combat these problems. The components of this solution suite include the following:

1. **Synopsis Diffusion:** Synopsis Diffusion is a general framework for computing aggregates over an arbitrary (multi-path) topology. It decouples aggregation algorithm and aggregation topology, providing the opportunity to optimize them independently. We provided the formal foundation of this framework, example algorithms, and evaluation results. Synopsis Diffusion is the first to show the feasibility of general in-network aggregation over an arbitrary topology.

2. **Adaptive aggregation:** We proposed *Tributary-Delta*, an energy efficient scheme that combines the benefits of existing tree-based and our multi-path-based aggregation by simultaneously running them on different parts of the network and automatically balancing the proportion of these two components as operating conditions change. Tributary-Delta is the first to demonstrate application-aware adaptation of aggregation algorithms to cope with long-term dynamics of operating conditions. In developing Tributary-Delta, we also proposed *Adaptive Rings*, a multi-path aggregation topology that can automatically adapt to link- or node-failures in an energy-efficient manner.

3. **Design principles to tolerate correlated failures:** Using a combination of experimental and mathematical analysis of several real-world failure traces, we debunked a number of common myths about how to design highly available systems. Based on our analysis, we identified a set of design principles that can be used to build Internet-scale sensing systems capable of tolerating correlated failures. Although the design principles allow us to exploit unique properties of sensing systems, they can be generalized and used in a large class of distributed storage systems.

4. **Lessons from availability evaluation:** Aside from evaluating our own ideas, we implemented in IrisNet and evaluated a number of existing techniques that were either not implemented (e.g., SQS) or not evaluated (e.g., the Paxos consensus algorithm) before in a large-scale WAN system. These implementations have demonstrated the behavior of these algorithms in practice; e.g., we showed that a number of important optimizations are required for a large number of concurrent Paxos instances to converge on WAN. Our experiments with IrisNet's storage component have identified important differences between the methodologies for evaluating availability and performance. We showed that, unlike performance, availability

can be accurately evaluated with simple *simulation*, with a realistic *failure trace* collected over a reasonably *long period* of time.

The first two contributions aid in robust data collection from wireless sensor networks, while the last two contributions are in the area of reliable storage of sensor data in Internet-connected nodes. In combination, they provide the end-to-end robustness of a sensing system, without any significant resource overhead.

Artifacts

In the course of this thesis, we developed the following artifacts in the context of IrisNet:

1. We designed and implemented the basic architecture of IrisNet that demonstrates the viability of our ideas related to Internet-scale sensing. The source code of IrisNet is available at `http://www.intel-iris.net`.

2. We developed a number of prototype applications on IrisNet that have validated our ideas. One application, IrisLog, has been publicly available since September 2003, letting users make live queries regarding resource usage of 450+ PlanetLab machines. IrisLog can be accessed at `http://www.intel-iris.net/irislog`. The source code of this application comes with the standard IrisNet distribution.

3. We designed and implemented IrisNet's distributed storage component that can tolerate correlated crash failures of system components and provide high availability of archived sensor data. It uses SQS, which we showed to be better in tolerating correlated failures. It automatically regenerates new replicas when existing ones fail. In choosing replication parameters, it explicitly considers failure correlation in real systems. The storage component allows the system administrator to specify a target availability; it then uses our empirically validated failure model to choose the replication parameters required to achieve the target availability. To avoid overload failures, we developed and implemented an efficient load balancing mechanism that quickly spreads load across multiple nodes when a flash-crowd approaches. The source code comes with the standard IrisNet distribution.

4. We developed a failure benchmark that generates failure traces (i.e., information about when individual nodes in a system fail and recover) for certain classes of distributed systems. The benchmark uses realistic models capturing certain failure properties (e.g., failure correlation) that we identified by studying the aforementioned failure traces. We believe that such a benchmark will be extremely useful in

more accurately evaluating availability of distributed systems. The benchmark can be found at `http://www.intel-iris.net/benchmark`.

The first two artifacts are specific to Internet-scale sensing. However, the last two artifacts are more general and can be used by a large class of distributed systems.

8.2 General Remarks

Our contributions in this thesis demonstrate a useful approach to addressing robustness in sensing systems. Our techniques, although designed for sensing systems, can help a large class of non-sensing systems as well.

This thesis argues that traditional techniques of improving robustness with additional resources (e.g., energy, storage) are not well suited for sensing applications because of their inherent constraints. A more suitable approach is to exploit applications' tolerance for approximate or slightly inconsistent data, which weakens the robustness *and* resource efficiency tradeoffs, which in turn enables robust and resource-efficient sensing systems. In a dynamic environment, such domain-specific techniques often pose weak tradeoffs between resource usage and performance/robustness. This thesis argues for application-aware adaptation in resolving these tradeoffs. Although such adaptation has been found effective in other domains such as mobile computing, sensor networks pose a different set of challenges because of their resource constrained and distributed nature. Through examples of such adaptation, we showed that this is feasible and cost-effective. We believe that the general ideas of exploiting domain-specific properties and adapting according to application semantics will remain the primary ways to address robustness of sensor networks. Such properties are definitely not limited to the ones we exploited; for example, one might improve robustness by exploiting spatial and temporal correlation of data in order to infer missing information.

Many systems other than sensor networks can benefit from the techniques and design principles we developed in this thesis. Aggregation is an important construct that many distributed systems can use to collect the global statistics necessary for their automatic adaptation. Synopsis Diffusion might be a suitable aggregation technique for non-sensing applications where an approximate answer is sufficient and the loss rate is high (e.g., systems using wireless or unreliable communication such as UDP). Its resilience against node mobility could make it useful in the context of mobile computing. Its independence of aggregation topology might make it appropriate for large-scale peer-to-peer systems, where building a dedicated tree aggregation topology in the face of high churn rate can be extremely challenging. Similarly, the general idea of our Tributary-Delta can be used in other domains to design efficient multi-modal protocols [ABNS02] that change their operating modes depending on operating conditions. The design principles we developed for wide area storage can be applied to a wide variety of Internet-scale systems. Although the storage layer that we designed and implemented is specific to sensing systems, it has a number of tuning knobs that could be adjusted for being used in other systems. For example, one could tune the quorum size of SQS to be a majority, and thus use our system for applications that do not allow inconsistency; one could also tune our system to use erasure-coding, instead of replication, and use it in a read-only system. In summary, we believe that although our techniques are designed in the sensing context, many other systems can be benefited from them.

8.3 Future Directions

Robust sensing remains a relatively new area in systems research. Consequently, each of the components on which we have worked on has opportunities for further improvement. In the rest of this section, we discuss some of the interesting paths for future research that we have not yet had time to explore.

8.3.1 Robust Aggregation in Wireless Sensors

We here list several future research directions related to Synopsis Diffusion.

A Relaxed Synopsis Diffusion Framework

Our formal framework of Synopsis Diffusion provides a set of easy-to-check properties that someone can use to prove correctness of a new algorithm. One drawback of this framework is its strict deterministic nature; it guarantees that the final answers produced by a Synopsis Diffusion algorithm are *exactly the same* irrespective of duplication and reordering of sensor data during in-network aggregation. The necessary conditions for this guarantee, as given by our framework, severely limit the set of functions that one can use within a Synopsis Diffusion algorithm.

However, in many situations, such a strong guarantee is unnecessary—an approximate answer within a small specified error margin is sufficient if that can be computed energy-efficiently. In such cases, the aggregation algorithm can produce *any* answer within the error margin; the answers do not need to always be the same. For example, the ϵ-approximation algorithm provided by Greenwald and Khanna [GK04] computes an approximate median V of N sensor data such that V's rank is at most ϵN away from $N/2$, the rank of the true median. Similarly, the ϵ-deficient counting algorithm by Manjhi et al. [MSDO05] can efficiently compute all the items with frequencies above a given threshold T, as well as a few false positive items whose frequencies exceed $T - \epsilon N$ but not T, where N is the total number of items. Depending on the order in which sensor data gets combined within the network, different executions may produce different answers, although each answer satisfies the error guarantees.

It is possible to compute ϵ-approximate answers in the Synopsis Diffusion framework. In [MNG05], we provided a Synopsis Diffusion algorithm for computing Frequent Items, based on the ϵ-deficient counting technique mentioned above. The false positives in the answer of this algorithm can differ over executions. This relaxed requirement lets us use functions beyond those allowed by our original Synopsis Diffusion framework. For example, the Synopsis Fusion function $SF()$ used in the above Frequent Items algorithm is not associative. In contrast, our current framework does not allow this flexibility; it requires that $SF()$ must be associative, which is an essential property to ensure that the final answers will be the same irrespective of the duplication and reordering. Therefore, we can not directly use our current framework to prove the correctness of an ϵ-approximation algorithm.[1] This poses a big problem in developing new such algorithms.

An open research problem is to find a suitable formal framework for ϵ-approximate Synopsis Diffusion algorithms. The framework would provide a set of easy-to-check properties (like the ones provided by our current framework) that one can use to prove the correctness of a new algorithm. Since the final answers of an ϵ-approximate algorithm do not need to strictly be the same, this new formal model is likely to be a relaxed version of our current formal framework. This model would enable using more general functions (e.g., those that are not associative) to design new ϵ-approximate Synopsis Diffusion algorithm.

More Examples

Another interesting direction for future research is to develop Synopsis Diffusion algorithms for new aggregates. Although we provided algorithms for a large number of useful aggregates, algorithms for many aggregates (such as wavelets, isobars, etc.) are still unknown. One approach to addressing this is to take the corresponding tree-based algorithms (e.g., the tree-based algorithm in [HHMS03] for computing wavelets), and replace

[1]To show the correctness of the Frequent Items algorithm in [MNG05], we showed that after every application of the Synopsis Fusion function, the intermediate result remains ϵ-approximate with respect to the input data. However, this technique is not always feasible.

the basic functions (e.g., additions) within them with equivalent ODI functions. We used this approach in designing the Frequent Items algorithm in [MNG05]. The challenge here is to deal with functions whose ODI equivalents are not known. For example, in the above mentioned Frequent Items algorithm, we had to devise ways to deal with the subtraction function (we do not know of any existing ODI subtraction function with good error guarantees using only a small sysnopsis).

A more fundamental question is whether there always exists an equivalent Synopsis Diffusion algorithm for any tree-based algorithm. After having an answer to this question, the next logical step would be to explore whether it is possible to automatically generate a Synopsis Diffusion algorithm from a tree-based algorithm.

Multi-sink Aggregation Topology

As is the norm in common aggregation topologies, our topologies for Synopsis Diffusion use a single base station. However, one can imagine a wireless sensor network that has multiple base stations (sinks). In such a topology, each base station independently collects aggregate data from a subset of sensors and sends its final synopses to a common point, such as an IrisNet SA. The SA, after collecting synopses from all the base stations, evaluates the fused synopsis to the final aggregate answer. Such a multi-sink topology makes the aggregation process more robust—even if sensors in some part of the network get disconnected from one base station, they still can contribute to the final answer through other base stations. Note that, this opportunity is unique to Synopsis Diffusion—in a multi-sink topology, the reading from a single sensor may reach multiple base stations and therefore an aggregation scheme that is not duplicate insensitive can no longer provide correct answers.

An interesting future direction would be to explore issues such as how to efficiently construct a multi-sink topology, how many base stations to use, where to place them in the network, how to organize the sensors to communicate with nearby base stations, etc. Studying the tradeoffs of energy-efficiency and robustness of such topologies would also be useful.

8.3.2 Aggregation over Imprecise and Incorrect Data

Our aggregation techniques assume that the data reported by sensors is exact. However, in practice, sensor data can often be imprecise because of environmental noise, calibration error, and resource optimizations. Providing robustness against such imprecise, and possibly incorrect, data is challenging. One can imagine two interesting schemes to deal with such impreciseness. In the first scheme, one could use techniques to remove noise from sensor data before aggregating it. Several properties of sensed data, such as temporal and spatial correlation, could be used for this data cleaning purpose. In the second scheme, sensors could represent all possible values of the sensed data with a probability distribution. For example, a motion sensor could report that what it has detected is a human with 70% probability and a vehicle with 30% probability. A suitable aggregation scheme could report a probability distribution of the final result, instead of a single final result. For example, for the query "total number of people", it could return the result "40 to 50 people with 70% probability, and 50 to 60 people with 30% probability". Such a probabilistic result seems to be more natural for imprecise, probabilistically defined sensor data. Although previous efforts [CKP03, EN03] have investigated some of these issues, incorporating them with multi-path-based aggregation would be useful.

One could also imagine sensors to be malicious, reporting wrong data. However, given that sensor readings are spatially correlated, in a sufficiently dense deployment, neighbors of a malicious sensor might be able to detect its malicious intention. Thus peers could provide feedbacks about each others' activities, building a distributed reputation system of the sensors. A robust aggregation scheme could then consider the reputation of a sensor while using its data. Such a scheme would be robust against malicious sensors.

8.3.3 Adaptation of Aggregation Schemes

We here list a number of interesting area of future research related to adaptive aggregation from wireless sensors.

Application-aware Adaptation

Our Tributary-Delta construction uses application-semantics to dynamically adapt aggregation algorithm and topology. In doing so, it uses a simple algorithm based on a count-threshold—it adapts the topology to ensure that the number of nodes contributing to the final answers at base station is above the given threshold (e.g., 90%). However, it is often impossible to directly relate such a threshold with the errors in the final aggregate answers. For example, an Average value computed over 90% of the sensors may be very close to or it can be very different from the Average computed over all the sensors, depending on the data in the 10% sensors not contributing to the final answer. Applications generally like to have guarantees about errors in the final answers (e.g., an Average answer within 95% of the true Average), but a count-threshold-based scheme does not help to obtain that.

One could improve our Tributary-Delta scheme with strategies that provide guarantees based on an error-threshold, instead of a count-threshold. For example, it can adapt the topology such that the Average value computed at the base station is within 5% of the true Average. In general, it is a difficult problem since the uncertainties due to missing data prevent a system from making such guarantees.[2] However, sensor data is often spatially and temporally correlated. This correlation might be used to approximately infer the missing data. This correlation, along with the knowledge of the total number or sensors, can provide an approximate error margin and confidence of the final answer. The aggregation scheme could use this approximate information to adapt itself and reduce errors.

More general Tributary-Delta

There are several ways our Tributary-Delta construction can be improved. First, the multi-path region can be used *anywhere* in the network, not just around the base station. Such a construction would be effective when the region with high loss rate is far from the base station; in such cases, our existing constructions would unnecessarily use multi-path between that lossy region and the base station. Such a more general topology needs to be carefully constructed to ensure correctness; namely, there must be exactly one *egress* node in the multi-path region that is reachable from all other nodes in the region and that connects to some tree node outside the region. Selecting such a node would require an appropriate leader election algorithm.

Second, the adaptation control can be decentralized. In our current construction, the base station controls the size and shape of the delta region by broadcasting control messages. In a network that needs to be adapted frequently, the control overhead, specially for `TD`, can be high, since the control messages need to be propagated from the base station to the switchable nodes. One simple way to address this is to divide the whole network into multiple regions and to assign one particular node in each region to be the controller of that region. The controller nodes then can control adaptation of their respective regions. However, to ensure that the final answer is within the global (count- or error-) thresholds, controllers would require to collaborate with each other. Doing this in energy efficient way might be challenging. It would be interesting to explore the feasibility of such designs with extensive analysis, simulation, and real deployments.

[2] This problem does not arise with count-threshold, since we assume that the base station knows the total count a priori.

Multi-modal Protocols

The basic idea of Tributary-Delta can be used in designing new multi-modal protocols for sensor networks. A multi-modal protocol changes its mode depending on the operating conditions. For example, one can imagine a routing protocol that uses energy-efficient unreliable communication when the loss rate is low or when the data to deliver can tolerate losses while it uses energy-expensive reliable communication when loss rate is high or when the data to deliver is too important to lose. The protocol even could use a mixture of reliable and unreliable communication in different parts of the network. To change operating mode, the protocol would need information about the operating conditions, and in most cases aggregate information (e.g., the average loss rate) would be sufficient for this purpose. Therefore, such protocols could use aggregation algorithms (e.g., Synopsis Diffusion) as building blocks to monitor the sensor network.

8.3.4 Wide-area Robustness

Understanding wide-area correlated failures and addressing them in system design is a relatively new area. In the following, we describe a number of areas in which our work can be extended.

Better Understanding of Wide-area Failures

Our study of wide-area failures is restricted due to several limitations of the traces we used. For example, our traces do not contain information about causes of failures. Moreover, some of them have long probe intervals (e.g., `PL_trace`), some have few nodes (e.g., `RON_trace`), and some have small length (e.g., `WS_trace`). These limitations restrict our study. For example, our study fails to answer the following important questions:

1. What are the different types of failures and what is their distribution? Which ones are the most common? In our study, we have seen that failure sizes show a bi-exponential property. What types of failures span different regions of the curve? If we consider only a certain class of failures (e.g., network failures), what does the curve look like?

2. What failure types have the most drastic impact on system availability? The impact can be different on different systems, e.g., a network partitioning may not have enough impact on SETI@Home type systems, small failure events may not have big impact on replicated systems, etc.

3. Can we model/predict patterns of certain types of failures? Is there any spatial/temporal correlation within different types of failures?

One could collect traces that do not suffer from the limitations of our traces and study them to understand the above questions. Findings from such study, along with our previous understanding of failure size distribution, could be useful in generating better failure workloads and studying a variety of availability problems. Such a study would also provide better insights for building highly available distributed systems.

Dynamic Adaptation

Our replication design in IrisNet uses a failure model in the beginning of configuration to determine parameters and does not support changing parameters on the fly. We can imagine a better system, one which is autonomous and can dynamically adapt its configuration as required. Such a system would require i) automatically monitoring the failure properties of the system, and ii) dynamically changing system parameters accordingly to ensure the target availability. A key challenge in online monitoring is scalability. In particular, monitoring network connectivity scales very poorly with the number of nodes.[3] The challenge of dynamic reconfiguration is to ensure availability and consistency of data during the change in replication parameters.

Single-server Regeneration

Our regeneration design could be greatly simplified by using a single regeneration server. In this novel design, the regeneration server works as the serialization and arbitration point of the regeneration protocol. When multiple replicas of the same replica group want to initiate regeneration, they all contact the regeneration server which then decides which replica can actually start the regeneration. In this way, a replica group never diverges into multiple groups. The use of a single server eliminates the necessity of complicated consensus protocols. However, one might raise three concerns regarding this design. We here briefly outline how these concerns might be addressed.

- **Scalability:** *Can a single server scale to handle regeneration of millions of replica groups?*

[3]For example, J. Stribling has been collecting the all-pair ping traces of PlanetLab [Str04]. The monitoring daemon pings each pair of nodes once every 15 minutes. As of this writing, the number of nodes in PlanetLab is approaching 600 and he is planning to stop the service because of its excessive overhead [Str05].

Note that regeneration is a rare event (as rare as failures), therefore the load imposed on the regeneration server will not be excessive. To further reduce load, the total replica groups in the system can be partitioned into smaller sets a priori such that each set has its own regeneration server. The correctness of this design is ensured by the condition that a replica group does not change its regeneration server. Therefore, we believe that scalability is not a major concern for this design.

- **Availability:** *Does the single server become the bottleneck of availability?* If the regeneration server is down, no replica groups depending on that server can regenerate.

  The key insight we use to address this concern is that regeneration is not a time sensitive task—a replica group needs to regenerate only before all the read quorums become unavailable. For the failure patterns in the traces we studied, this time is more than half a day. So, as long as the MTTR of the server is no more than half a day, the regeneration process would remain available. This requirement can be met easily; since there is only a single regeneration server, it can be maintained well enough to make the MTTR less than half a day.

- **Security:** *Is the server the single point of failure?* An attacker can launch a DoS attack on the server to stop regeneration of the whole system.

  Addressing this concern is challenging. One possible solution can be to manually shut down the old server, start a new one, and broadcast the identity of the new server to all the relevant replica groups through a secure channel. The correctness of this scheme is given by the requirement that a human shuts down the old server before starting a new one, and in this way the human acts as the serialization point during the transition. Finding more practical and effective solutions requires further research.

We believe that the approach of single server regeneration is promising. However, its feasibility needs to be explored with careful design, analysis and deployment.

8.4 Closing Remarks

It appears likely that robustness will continue to be a significant challenge of sensing systems for the foreseeable future. Since failures are prevalent in harsh deployment environments and masking them is not trivial given the constraints of sensing systems (such as their resource constraints, large scale, and unattended nature), finding efficient techniques capable of addressing them will continue to be an important research area.

In this thesis, we argued that traditional techniques of improving robustness with additional resources (e.g., energy, storage) are not well suited for sensing applications because of their inherent constraints. A more suitable approach is to exploit applications' tolerance for approximate or slightly inconsistent data, which weakens the robustness and resource efficiency tradeoffs, which in turn enables robust and resource-efficient sensing systems. We showed the feasibility of this approach through algorithm designs, experiments, and analysis. Our results have shown that this approach can improve robustness by up to several orders of magnitude, without significant resource overhead.

Moving forward, we believe that the general idea of exploiting domain-specific properties will remain the primary way to address robustness of sensor networks. Such properties are definitely not limited to the ones we exploited: weak data semantics, broadcast medium, etc. For example, one might improve robustness by exploiting spatial and temporal correlation of data in order to infer missing information. Such domain-specific techniques appear to be practical and cost-effective, and they will play a significant role in addressing the grand challenge of building critical sensing systems that we can count on.

Bibliography

[ABB$^+$03] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Utkarsh Srivastava, Dilys Thomas, Rohit Varma, and Jennifer Widom, *STREAM: The Stanford stream data manager*, IEEE Data Engineering Bulletin **26** (2003), no. 1, 19–26.

[ABF93] Baruch Awerbuch, Yair Bartal, and Amos Fiat, *Competitive distributed file allocation*, Proceedings of the Annual ACM Symposium on Theory of Computing (STOC), 1993.

[ABGM90] Rafael Alonso, Daniel Barbara, and Hector Garcia-Molina, *Data caching issues in an information retrieval system*, ACM Transactions on Database Systems (TODS) (1990).

[ABKM01] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris, *Resilient overlay networks*, Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), 2001.

[ABNS02] Aditya Akella, Ashwin Bharambe, Suman Nath, and Srinivasan Seshan, *Multimodal network protocols: Adapting to highly variable operating conditions*, Tech. Report CMU-CS-02-170, Carnegie Mellon University, Computer Science Department, 2002.

[Aka05] Akamai, *Akamai: The trusted choice for online bussiness*, http://www.akamai.com/, 2005.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy, *The space complexity of approximating the frequency moments*, Journal of Computer and System Sciences **58** (1999), 137–147.

[And05] David G. Andersen, *The MIT RON trace*, Private communication, 2005.

[Apa01] Apache, *Apache Xindice project*, http://xml.apache.org/xindice/, 2001.

[Ape88] Peter M. G. Apers, *Data allocation in distributed database systems*, ACM Transactions on Database Systems (TODS) **13** (1988), no. 3, 263–304.

[AS93] Divyakant Agrawal and Soumitra Sengupta, *Modular synchronization in distributed, multi-version databases: Version control and concurrency control*, IEEE Transactions on Knowledge and Data Engineering (1993).

[Atm03] Atmel Corporation, *Atmel AVR microcontroller datasheet. http://www.atmel.com/dyn/resources/prod_documents/2467s.pdf*, 2003.

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom, *Models and issues in data stream systems*, Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 2002.

[BDET00] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer, *Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs*, ACM SIGMETRICS Performance Evaluation Review **28** (2000), no. 1, 34–43.

[Bel03] Bell Labs, *Galax: An implementation of XQuery*, http://www.galaxquery.org/, 2003.

[BGGMM04] Mayank Bawa, Aristides Gionis, Hector Garcia-Molina, and Rajeev Motwani, *The price of validity in dynamic networks*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 2004.

[BGM87] Daniel Barbara and Hector Garcia-Molina, *The reliability of voting mechanisms*, IEEE Transactions on Computers (1987), 1197–1208.

[BGS01] Philippe Bonnet, J. E. Gehrke, and Praveen Seshadri, *Towards sensor database systems*, Proceedings of the IEEE Mobile Data Management, 2001.

[BK00] Atul Butte and Isaac Kohane, *Mutual information relevance networks: functional genomic clustering using pairwise entropy measurements*, Proceedings of the Pacific Symposium on Biocomputing, 2000, 2000.

[BLS95] Anna Brunstrom, Scott T. Leutenegger, and Rahul Simha, *Experimental evaluation of dynamic data allocation strategies in a distributed database* *with changing workloads*, Proceedings of the 4th International Conference on Information and Knowledge Management, 1995.

[BRY+04] Maxim A. Batalin, Mohammad Rahimi, Yan Yu, Duo Liu, Aman Kansal, Gaurav S. Sukhatme, William J. Kaiser, Mark Hansen, Gregory J. Pottie, Mani Srivastava, and Deborah Estrin, *Call and response: experiments in sampling the environment*, Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys), 2004.

[BSV03] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker, *Understanding availability*, Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS), 2003.

[BTC+04] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoff M. Voelker, *TotalRecall: Systems support for automated availability management*, Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), 2004.

[BWWG02] Mehmet Bakkaloglu, Jay J. Wylie, Chenxi Wang, and Gregory R. Ganger, *On correlated failures in survivable storage systems*, Tech. Report CMU-CS-02-129, Carnegie Mellon University, May 2002.

[BYKS01] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar, *Sampling algorithms: lower bounds and applications*, Proceedings of the Annual ACM Symposium on Theory of Computing (STOC), 2001.

[Cat03] Josh Cates, *Robust and efficient data management for a distributed hash table*, Masters Thesis, Massachusetts Institute of Technology, May 2003.

[CDK+03] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh, *SplitStream: High-bandwidth multicast in a cooperative environment*, Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), 2003.

[CE04] Alberto Cerpa and Deborah Estrin, *ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies*, IEEE Transactions on Mobile Computing **3** (2004), no. 3, 272–285.

[CEG+04] Peter Corbett, Bob English, Atul Goel, Tomislav Grcanac, Steven Kleiman, James Leong, and Sunitha Sankar, *Row-diagonal parity for double disk fail-*

ure correction, Proceedings of the USENIX Conference on File and Storage Technologies (FAST), 2004.

[CGMR05] Graham Cormode, Minos Garofalakis, S. Muthukrishnan, and Rajeev Rastogi, *Holistic aggregates in a networked world: distributed tracking of approximate quantiles*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 2005.

[CGN+05a] Jason Campbell, Phillip B. Gibbons, Suman Nath, Padmanabhan Pillai, Srinivasan Seshan, and Rahul Sukthankar, *IrisNet: An Internet-scale architecture for multimedia sensors*, Proceedings of the ACM Multimedia, November 2005.

[CGN05b] Shimin Chen, Phillip B. Gibbons, and Suman Nath, *Database-centric programming for wide-area sensor systems*, Proceedings of 1st International Conference on Distributed Computing in Sensor Systems (DCOSS), 2005.

[CIL03] CIL (Coastal Imaging Lab), College of Oceaninc and Atmospheric Sciences, Oregon State University, *The Argus program*, http://cil-www.oce.orst.edu:8080/, 2003.

[CKP03] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar, *Evaluating probabilistic queries over imprecise data*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 2003.

[CLFK01] Robert T. Collins, Alan J. Lipton, Hironobu Fujiyoshi, and Takeo Kanade, *Algorithms for cooperative multisensor surveillance*, Proceedings of the IEEE **89** (2001), no. 10, 1456–1477.

[CLKB04] Jeffrey Considine, Feifei Li, George Kollios, and John Byers, *Approximate aggregation techniques for sensor databases*, Proceedings of the International Conference on Data Engineering (ICDE), 2004.

[CM05] Graham Cormode and S. Muthukrishnan, *An improved data stream summary: the count-min sketch and its applications*, Journal of Algorithms **55** (2005), no. 1, 58–75.

[Com04] ComputerWorld.com, *Akamai now says it was targeted by DDoS attack*, http://www.computerworld.com/securitytopics/security/story/0,10801,93862,00.html, 2004.

[CP92] S. W. Chen and C. Pu, *A structural classification of integrated replica control mechanisms*, Tech. Report CUCS-006-92, Columbia University, 1992.

[CSK02] Weidong Cui, Ion Stoica, and Randy H. Katz, *Backup path allocation based on a correlated link failure probability model in overlay networks*, Proceedings of the IEEE International Conference on Network Protocols (ICNP), 2002.

[CV03] Brent Chun and Amin Vahdat, *Workload and failure characterization on a large-scale federated testbed*, Tech. Report IRB-TR-03-040, Intel Research Berkeley, November 2003.

[DF82] Lawrence W. Dowdy and Derrell V. Foster, *Comparative models of the file assignment problem*, ACM Computing Surveys **14** (1982), no. 2.

[DGM+02] Amol Deshpande, Carlos Guestrin, Sam Madden, Joseph M. Hellerstein, and Wei Hong, *Model-driven data acquisition in sensor networks*, Proceedings of the International Conference on Very Large Data Bases (VLDB), 2002.

[DKK+01] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica, *Wide-area cooperative storage with CFS*, Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), October 2001.

[DLS+04] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris, *Designing a DHT for low latency and high throughput*, Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), 2004.

[DNGS03] Amol Deshpande, Suman Nath, Phillip B. Gibbons, and Srinivasan Seshan, *Cache-and-query for wide area sensor databases*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 2003.

[DP02] Brian A. Davey and Hilary A. Priestley, *Introduction to lattices and order*, 2002.

[DRS+05] Karthik Dantu, Mohammad H. Rahimi, Hardik Shah, Sandeep Babel, Amit Dhariwal, and Gaurav S. Sukhatme, *Robomote: Enabling mobility in sensor networks*, Proceedings of the IEEE/ACM Fourth International Conference on Information Processing in Sensor Networks (IPSN), 2005.

[DW01] John R. Douceur and Roger P. Wattenhofer, *Competitive hill-climbing strategies for replica placement in a distributed file system*, Proceedings of the International Symposium on Distributed Computing (DISC), 2001.

[EDHD02] Ahmed Elgammal, Ramani Duraiswami, David Harwood, and Larry S. Davis, *Background and foreground modeling using nonparametric kernel density estimation for visual surveillance*, Proceedings of the IEEE, vol. 90, July 2002.

[EN03] Eiman Elnahrawy and Badri Nath, *Cleaning and querying noisy sensors*, Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications, 2003.

[Flu05] Flux Research Group, The University of Utah, *Emulab - network emulation testbed home*, http://www.emulab.net, 2005.

[FM85] Philippe Flajolet and G. Nigel Martin, *Probabilistic counting algorithms for database applications*, Journal of Computer and System Sciences **31** (1985), 182–209.

[GGSE02] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin, *Highly-resilient, energy-efficient multipath routing in wireless sensor networks*, Mobile Computing and Communications Review (M2CR) **1** (2002), no. 2.

[GHOS96] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha, *The dangers of replication and a solution*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 1996.

[GK04] Michael Greenwald and Sanjeev Khanna, *Power-conserving computation of order-statistics over sensor networks*, Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 2004.

[GKK$^+$03] Phillip B. Gibbons, Brad Karp, Yan Ke, Suman Nath, and Srini Seshan, *Irisnet: An architecture for a worldwide sensor web*, IEEE Pervasive Computing **2** (2003), no. 4.

[GM99] Phillip B. Gibbons and Yossi Matias, *Synopsis data structures for massive data sets*, Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999.

[GT01] Phillip B. Gibbons and Srikanta Tirthapura, *Estimating simple functions on the union of data streams*, Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA), 2001.

[GT02] ______, *Distributed streams algorithms for sliding windows*, Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA), 2002.

[GWGR04] Garth Goodson, Jay Wylie, Gregory Ganger, and Michael Reiter, *Efficient byzantine-tolerant erasure-coded storage*, Proceedings of the International Conference on Dependable Systems and Networks (DSN), June–July 2004.

[HBD97] Mor Harchol-Balter and Allen B. Downey, *Exploiting process lifetime distributions for dynamic load balancing*, ACM Transactions on Computer Systems **15** (1997), no. 3, 253–285.

[HHMS03] Joseph M. Hellerstein, Wei Hong, Samuel Madden, and Kyle Stanek, *Beyond average: Towards sophisticated sensing with queries*, Proceedings of the International Workshop on Information Processing in Sensor Networks (IPSN), March 2003.

[HMD05] Andreas Haeberlen, Alan Mislove, and Peter Druschel, *Glacier: Highly durable, decentralized storage despite massive correlated failures*, Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), 2005.

[Hoe63] Wassily Hoeffding, *Probability inequalities for sums of bounded random variables*, Journal of the American Statistical Association **58** (1963), no. 301, 13–30.

[HSOH03] Rob Holman, John Stanley, and Tuba Özkan Haller, *Applying video sensor networks to nearshore environment monitoring*, IEEE Pervasive Computing **2** (2003), no. 4.

[HSW$^+$00] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister, *System architecture directions for networked sensors*, Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2000.

[HTB+05] Wen Hu, Van Nghia Tran, Nirupama Bulusu, Chun tung Chou, Sanjay Jha, and Andrew Taylor, *The design and evaluation of a hybrid sensor network for cane-toad monitoring*, Proceedings of the 4th Information Processing in Sensor Networks (IPSN), 2005.

[HW90] Maurice Herlihy and Jeannette Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Transactions on Programming Languages and Systems **12** (1990), no. 3.

[IGE+03] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva, *Directed diffusion for wireless sensor networking*, IEEE/ACM Transactions on Networking (TON) **11** (2003), no. 1, 2–16.

[Int03a] Intel Research Pittsburgh, *IrisLog: A distributed syslog*, http://www.intel-iris.net/irislog.php, 2003.

[Int03b] ______, *IrisNet: Internet-scale resource-intensive sensor network services*, http://www.intel-iris.net/, 2003.

[Int05] Intel Research Berkeley, *Intel lab data*, http://berkeley.intel-research.net/labdata/, 2005.

[JBH+05] Flavio Junqueira, Ranjita Bhagwan, Alejandro Hevia, Keith Marzullo, and Geoffrey M. Voelker, *Surviving Internet catastrophe*, Proceedings of the USENIX Annual Technical Conference, 2005.

[JM96] David B. Johnson and David A. Maltz, *Dynamic source routing in ad hoc wireless networks*, Mobile Computing (Imielinski and Korth, eds.), vol. 353, 1996.

[JM02] Flavio Junqueira and Keith Marzullo, *Designing algorithms for dependent process failures*, Proceedings of the International Workshop on Future Directions in Distributed Computing (FuDiCo), 2002.

[KB91] Narayanan Krishnakumar and Arthur J. Bernstein, *Bounded ignorance in replicated systems*, Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 1991.

[KBC+00] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao, *OceanStore: an architecture for global-*

scale persistent storage, Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2000.

[KDG03] David Kempe, Alin Dobra, and Johannes Gehrke, *Gossip-based computation of aggregate information*, Proceedings of the Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2003.

[KDW01] Konstantinos Kalpakis, Koustuv Dasgupta, and Ouri Wolfson, *Optimal placement or replicas in trees with read, write, and storage costs*, IEEE Transactions on Parallel and Distributed Systems **12** (2001), no. 6, 628–637.

[KK98] George Karypis and Vipin Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing **20** (1998), no. 1, 359–392.

[KP97] Kamalakar Karlapalem and Ng Moon Pun, *Query-driven data allocation algorithms for distributed database systems*, Database and Expert Systems Applications, 1997.

[KR01] Jussi Kangasharju and James Roberts Keith W. Ross, *Object replication strategies in content distribution networks*, Proceedings of 6th International Web Caching and Content Distribution Workshop, 2001.

[KRA+03] Dejan Kostic, Adolfo Rodriguez, Jeannie R. Albrecht, Abhijeet Bhirud, and Amin Vahdat, *Using random subsets to build scalable network services*, Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS), 2003.

[KT02] Greg Kogut and Mohan Trivedi, *A wide area tracking system for vision sensor networks*, Proceedings of 9th World Congress on Intelligent Transport Systems, 2002.

[LA00] Thanasis Loukopoulos and Ishfaq Ahmad, *Static and adaptive data replication algorithms for fast information access in large distributed systems*, Proceedings of the International Conference on Distributed Computing Systems (ICDCS), 2000.

[Lam98] Leslie Lamport, *The part-time parliament*, ACM Transactions on Computer Systems **16** (1998), 133–169.

[LGC05] Philip Levis, David Gay, and David Culler, *Active sensor networks*, Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), 2005.

[LMG+04] Philip Levis, Sam Madden, David Gay, Joe Polastre, Robert Szewczyk, Eric Brewer, Alec Woo, and David Culler., *The emergence of networking abstractions and techniques in TinyOS*, Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), 2004.

[LR00] Qun Li and Daniela Rus, *Sending messages to mobile users in disconnected ad-hoc wireless networks*, Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom), 2000.

[LS02] Nancy Lynch and Alex A. Shvartsman, *RAMBO: a reconfigurable atomic memory service for dynamic networks*, Proceedings of the International Symposium on Distributed Computing (DISC), 2002.

[Luk74] J. A. Lukes, *Efficient algorithm for the partitioning of trees*, IBM Journal of Research and Development **18** (1974), no. 3, 217–224.

[LY80] K. Lam and Clement T. Yu, *An approximation algorithm for a file-allocation problem in a hierarchical distributed system*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 1980.

[Lyn97] Nancy Lynch, *Distributed algorithms*, 1997.

[Mad03] Samuel Madden, *The design and evaluation of a query processing architecture for sensor networks*, Ph.D. thesis, U. C. Berkeley, December 2003.

[MCC04] Matthew L. Massie, Brent N. Chun, and David E. Culler, *The Ganglia distributed monitoring system: Design, implementation, and experience*, Parallel Computing **30** (2004), no. 7, 817–840.

[MD88] Paul V. Mockapetris and Kevin J. Dunlap, *Development of the Domain Name System*, Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 1988.

[MFHH02] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong, *TAG: A tiny aggregation service for ad hoc sensor networks*, Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2002.

[MFHH03] ______, *The design of an acquisitional query processor for sensor networks*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 2003.

[MNG05] Amit Manjhi, Suman Nath, and Phillip B. Gibbons, *Tributaries and deltas: Efficient and robust aggregation in sensor network streams*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 2005.

[MPD04] Shoubhik Mukhopadhyay, Debashis Panigrahi, and Sujit Dey, *Model based error correction for wireless sensor networks*, Proceedings of the First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON), 2004.

[MRW97] Dahlia Malkhi, Michael Reiter, and Rebecca Wright, *Probabilistic quorum systems*, Proceedings of the Annual ACM Symposium on Principles of Distributed Computing (PODC), 1997.

[MSDO05] Amit Manjhi, Vladislav Shkapenyuk, Kedar Dhamdhere, and Christopher Olston, *Finding (recently) frequent items in distributed data streams*, Proceedings of the International Conference on Data Engineering (ICDE), 2005.

[MSFC02] Samuel Madden, Robert Szewczyk, Michael J. Franklin, and David Culler, *Supporting aggregate queries over ad-hoc wireless sensor networks*, Proceedings of Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), 2002.

[Mut03] S. Muthukrishnan, *Data streams: Algorithms and applications*, Tech. report, Rutgers University, 2003.

[NGS05] Suman Nath, Phillip B. Gibbons, and Srinivasan Seshan, *Adaptive data placement for wide-area sensing services*, Proceedings of the USENIX Conference on File and Storage Technologies (FAST), 2005.

[NGSA04] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson, *Synopsis diffusion for robust aggregation in sensor networks*, Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys), 2004.

[NKG+04] Suman Nath, Yan Ke, Phillip B. Gibbons, Brad Karp, and Srinivasan Seshan, *A distributed filtering architecture for multimedia sensors*, Proceedings of the First Workshop on Broadband Advanced Sensor Networks (BaseNets), 2004.

[NZ02] TS Eugene Ng and Hui Zhang, *Predicting Internet network distance with coordinates-based approaches*, Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 2002.

[OSM+04] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam, *Fault-aware job scheduling for BlueGene/L systems*, Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), 2004.

[OW02] Chris Olston and Jennifer Widom, *Best-effort cache synchronization with source cooperation*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 2002.

[PACR02] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe, *A blueprint for introducing disruptive technology into the Internet*, Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I), 2002.

[PG03] Mark A. Paskin and Carlos E. Guestrin, *A robust architecture for distributed inference in sensor networks*, Tech. Report IRB-TR-03-039, Intel Research Berkeley, 2003.

[PGF02] Christopher R. Palmer, Phillip B. Gibbons, and Christos Faloutsos, *ANF: A fast and scalable tool for data mining in massive graphs*, Proceedings of the 8th ACM SIGKDD International Conference on Knowledge and Data Mining, 2002.

[PL91] Calton Pu and Avraham Leff, *Replica control in distributed system: An asynchronous approach*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 1991.

[Pla97] James Plank, *A tutorial on reed-solomon coding for fault-tolerance in RAID-like systems*, Software – Practice & Experience **27** (1997), no. 9, 995–1012.

[Pla05] PlanetLab Consortium, *PlanetLab*, http://www.planet-lab.net/, 2005.

[PSP03] Bartosz Przydatek, Dawn Song, and Adrian Perrig, *SIA: secure information aggregation in sensor networks*, Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys), 2003.

[QPV01] Lili Qiu, Venkata N. Padmanabhan, and Geoffrey M. Voelker, *On the placement of web server replicas*, Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 2001.

[RRH00] Ram Ramanathan and Regina Rosales-Hain, *Topology control of multihop radio networks using transmit power adjustment*, Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 2000.

[SAS+96a] Jeff Sidell, Paul M. Aoki, Adam Sah, Carl Staelin, Michael Stonebraker, and Andrew Yu, *Data replication in mariposa*, Proceedings of the International Conference on Data Engineering (ICDE), 1996.

[SAS+96b] ______, *Data replication in mariposa*, Proceedings of the International Conference on Data Engineering (ICDE), 1996.

[SGG02] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble, *A measurement study of peer-to-peer file sharing systems*, Proceedings of Multimedia Computing and Networking (MMCN), 2002.

[SGW01] Loren Schwiebert, Sandeep K.S. Gupta, and Jennifer Weinmann, *Research challenges in wireless networks of biomedical sensors*, Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom), 2001.

[SH03] Fred Stann and John Heidemann, *RMST: Reliable data transport in sensor networks*, Proceedings of 1st IEEE International Workshop on Sensor Net Protocols and Applications (SNPA), 2003.

[SKKM02] Yasushi Saito, Christos Karamanolis, Magnus Karlsson, and Mallik Mahalingam, *Taming aggressive replication in the pangaea wide-area file system*, Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2002.

[SM00] Jianbo Shi and Jitendra Malik, *Normalized cuts and image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **22** (2000), no. 8, 888–905.

[SMAL+04] Gyula Simon, Miklós Maróti, Ákos Lédeczi, György Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and K. Frampton, *Sensor network-based countersniper system*, Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys), 2004.

[SMP01] Mani Srivastava, Richard Muntz, and Miodrag Potkonjak, *Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving enviroments*, Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom), 2001.

[SMP+04] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler, *An analysis of a large scale habitat monitoring application*, Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys), 2004.

[Str04] Jeremy Stribling, *PlanetLab - all pair pings*, http://www.pdos.lcs.mit.edu/∼strib/pl_app/, 2004.

[Str05] ______, *The fate of all pairs pings*, PlanetLab mailing list. The users archive. http://lists.planet-lab.org/pipermail/users/, 2005, Email on 6/24/2005.

[SW03] CN Sze and Ting-Chi Wang, *Optimal circuit clustering for delay minimization under a more general delay model*, IEEE Transactions on Computer-Aided Design of Integrated Circuits System **22** (2003), no. 5.

[Tho79] Robert H. Thomas, *A majority consensus approach to concurrency control for multiple copy databases*, ACM Transactions on Database Systems (TODS) **4** (1979), 180–209.

[TI92] Dong Tang and Ravishankar K. Iyer, *Analysis and modeling of correlated failures in multicomputer systems*, IEEE Transactions on Computers **41** (1992), no. 5, 567–577.

[TKC+04] Yufei Tao, George Kollios, Jeffrey Considine, Feifei Li, and Dimitris Papadias, *Spatio-temporal aggegration using sketches*, Proceedings of the International Conference on Data Engineering (ICDE), 2004.

[WCK04] Chieh-Yih Wan, Andrew T. Campbell, and Lakshman Krishnamurthy, *Reliable transport for sensor networks: PSFQ - Pump Slowly Fetch Quickly paradigm*, Wireless Sensor Networks (2004), 153–182.

[WJH97] Ouri Wolfson, Sushil Jajodia, and Yixiu Huang, *An adaptive data replication algorithm*, ACM Transactions on Database Systems (TODS) **22** (1997), no. 2, 255–314.

[WK02] Hakim Weatherspoon and John Kubiatowicz, *Erasure coding vs. replication: A quantitative comparison*, Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS), 2002.

[WM91] Ouri Wolfson and Amir Milo, *The multicast policy and its relationship to replicated data placement*, ACM Transactions on Database Systems (TODS) **16** (1991), no. 1, 181–205.

[WMK02] Hakim Weatherspoon, Tal Moscovitz, and John Kubiatowicz, *Introspective failure analysis: Avoiding correlated failures in peer-to-peer systems*, Proceedings of International Workshop on Reliable Peer-to-Peer Distributed Systems, October 2002.

[WWW99] WWWC : The World Wide Web Consortium, *XML Path Language (XPATH)*, http://www.w3.org/TR/xpath, 1999.

[XHE00] Ya Xu, John Heidemann, and Deborah Estrin, *Adaptive energy-conserving routing for multihop ad hoc networks*, Tech. Report 527, USC/Information Sciences Institute, October 2000.

[XRC+04] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin, *A wireless sensor network for structural monitoring*, Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys), 2004.

[YG03] Yong Yao and Johannes Gehrke, *Query processing in sensor networks*, Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR), 2003.

[YMV+03] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, and Mike Dahlin, *Separating agreement from execution for byzantine fault tolerant services*, Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), 2003.

[YNY+04] Praveen Yalagandula, Suman Nath, Haifeng Yu, Phillip B. Gibbons, and Srinivasan Seshan, *Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems*, Proceedings of the First Workshop on Real, Large Distributed Systems (WORLDS), 2004.

[Yu03] Haifeng Yu, *Overcoming the majority barrier in large-scale systems*, Proceedings of the International Symposium on Distributed Computing (DISC), October 2003.

[Yu04] ———, *Signed quorum systems*, Proceedings of the Annual ACM Symposium on Principles of Distributed Computing (PODC), 2004.

[YV01] Haifeng Yu and Amin Vahdat, *The costs and limits of availability for replicated services*, Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), 2001.

[YV04] ———, *Consistent and automatic replica regeneration*, Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), 2004.

[ZG03] Jerry Zhao and Ramesh Govindan, *Understanding packet delivery performance in dense wireless sensor networks*, Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys), 2003.

[ZGE03] Jerry Zhao, Ramesh Govindan, and Deborah Estrin, *Computing aggregates for monitoring wireless sensor networks*, Proceedings of 1st IEEE International Workshop on Sensor Network Protocols and Applications, 2003.

[ZKOS05] Rui Zhang, Nick Koudas, Beng Chin Ooi, and Divesh Srivastava, *Multiple aggregations over data streams*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 2005.

[ZSC+03] Stanley B. Zdonik, Michael Stonebraker, Mitch Cherniack, Ugur Çetintemel, Magdalena Balazinska, and Hari Balakrishnan, *The Aurora and Medusa projects*, IEEE Data Engineering Bulletin **26** (2003), no. 1, 3–10.

Index