Emodis—An End-Based Network Monitoring and Diagnosis System

Ningning Hu, Peter Steenkiste

June 2005

CMU-CS-05-146

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Network monitoring and diagnosis capabilities are critical for the seamless operation of a network. ISPs use sophisticated systems to routinely monitor and diagnose their networks, but end users do not have such capabilities. To address this problem, we develop Emodis—a network monitoring and diagnosis system. In this paper, we describe the architecture and the software components of Emodis. Like other end-user oriented network monitoring systems, Emodis is deployed on a diverse set of Internet nodes, so it shares common requirements such as security and robustness with these systems. However, the focus of Emodis is on route-sensitive path metrics such as available bandwidth and packet loss rate, resulting in two unique characteristics: (1) it implements a variety of measurement techniques, including sophisticated bandwidth measurement techniques, but hides many technical details from end users; (2) it implements a scheduling algorithm to synchronize the measurements from different vantage points, which relieves end users from complicated network measurement management.

Ningning Hu and Peter Steenkiste were in part supported by the NSF under award number CCR-0205266.

Keywords: Architecture, monitoring, diagnosis, measurement

1 Introduction

Improving Internet performance and reliability has always been a focus of networking research and engineering, but network failures such as disconnection are still not rare phenomena. ISP operators deploy sophisticated monitoring systems [7][6] to collect large amount of information so as to detect, diagnose and fix network problems in a timely matter. Regular end users, however, do not have these capabilities. Regular end users do not have the necessary knowledge and resources to monitor and diagnose their network connections. For example, measuring downstream network performance needs the cooperation of other Internet nodes which is hard to obtain. Also, network failures often disconnect end users thus preventing them from diagnosing their network connectivity. Fortunately, this problem has not escaped the attention of researchers and several systems, e.g. NIMI [15] and Scriptroute [17], have been developed and deployed. They provide general measurement and monitoring infrastructures, which include functionalities like automated deployment, privacy protection, and security. End users can use these infrastructures to implement and deploy measurement tools to collect network performance information.

However, these infrastructures mostly focus on solving the "resource" problem faced by end users. Little has been done to address the "knowledge" part required to do monitoring and diagnosis. That is, these systems make it *possible* for end users to monitor or diagnose their network performance, but it does not necessarily make it practical since they still face non-trivial difficulties. First, in most cases, end users need to install measurement tools by themselves. That requires a good understanding of these tools, some of which, like those used for bandwidth or packet loss measurements, can be fairly complicated. Second, many network properties like delay, loss rate, available bandwidth, are both path-sensitive and directional. They have different values for different paths, or for different directions (upstream or downstream) of the same path. Given a measurement system deployed on multiple vantage nodes, end users need to choose the right vantage node to conduct measurement in order to obtain the right data. This selection often requires knowing either the network topology or routing properties of the vantage nodes, which is beyond the grasp of end users.

Finally, existing monitoring systems do not provide measurement management functionality, which can be very tedious and time-consuming. A typical example is support for synchronized measurements. For monitoring or diagnosis purpose, it is often necessary to measure a common property from multiple vantage points *simultaneously*, so that they can be compared and correlated. However, measuring the same destination from multiple vantage points can not only trigger security alarms, but can also increase measurement error due to the interference among the measurements from different vantage points. Although this may not be a big problem for ping or traceroute measurements if the number of vantage points is relatively small, it can be a serious problem for bandwidth measurement techniques that rely on packet-train probing. For this reason, synchronized measurements sometimes have to be conducted sequentially. That is, the measurement from one vantage node to a destination does not start until the measurement from another node to the same destination has finished. To reduce measurement time, such sequential measurements have to be done continuously, and that requires the vantage points to closely collaborate with each other. Meanwhile, since there could be a large number of destinations that need synchronized measurements, the cooperation among measurement vantage nodes can be very complicated.

Emodis is designed to address these problems. Although Emodis has many similarities with existing network monitoring systems, it focuses on relieving end users from the measurement and management burden. That is, Emodis not only makes network monitoring and diagnosis possible, it also make such tasks *easier*. Emodis has the following three distinguishing properties. First, it implements a variety of network measurement and diagnosis techniques for delay (using ping), route (using traceroute), available bandwidth (using IGI/PTR [10]), bottleneck location (using Pathneck [8]), and end-edge bandwidth summary (using BRoute mechanism [9] whose measurements are based on Pathneck). To use this system, end users do not need to understand the technical details of related measurement techniques: they shall just submit a measurement/diagnosis request and wait for the results. Second, Emodis measurement software is installed on a diverse set of network nodes. That makes Emodis service immune to the network failures experienced by end users. Also, this distributed design makes it possible to measure downstream network performance for end users. Finally, Emodis automates measurement and diagnosis operations, including measurement scheduling and overhead management. Besides the issue of synchronized measurements that is discussed above, Emodis can also do load balancing of measurement work among multiple vantage points. This is a very useful functionality for the scenario where a large number of destinations need to be measured to get a snapshot of their performance. These scheduling capabilities of Emodis relieve end users from the complicated and tedious management operations required when conducting large measurement studies.

While our focus is on simplifying network measurements for end users, we believe that the same architecture can also help providers monitor their networks since many of the required functions (e.g. tools, scheduling and synchronization) are the same or very similar.

Emodis also provides a platform for end users to share network performance information. Information sharing is not only important for reducing measurement overhead, but it can also help derive information that is not easily obtainable by a single end user. For example, detecting loss-link location often requires the use of tomography techniques, which require measurement results from multiple sources. From a research point of view, by simplifying the tasks for end users, Emodis can hopefully attract more end users, which can provide an opportunity for collecting performance information from many real end users, thus allowing us to gain a better understanding on how to diagnose client network problems.

We use a top-down method in this paper to describe the Emodis design. We first describe the architecture of Emodis in Section 2, explaining the three node types in the system—client, master, and agent. The next four sections describe the system details: (1) Section 3 explains the client interface to show the measurement and diagnosis options that an end user can control; (2) Section 4 describes the main functionality of the master node—scheduling and synchronization; (3) Section 5 describes probing algorithms that are used in the agent, and (4) Section 6 deals with the problem of system nodes leaving and joining. We conclude with a discussion of related work, conclusions, and future work in Section 7 and Section 8.

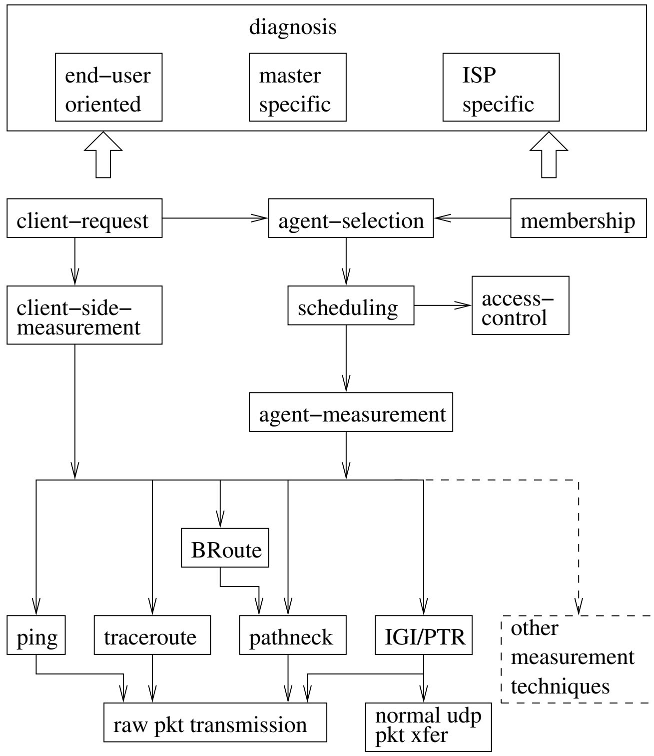

Figure 1: Emodis functional modules

2 Architecture

In this section, we first look at the functional modules in Emodis. We then examine the possible architectures that can be used for different application scenarios.

2.1 Emodis Functional Modules

The functional relationship among Emodis system modules is illustrated in Figure 1. A client call starts in the `client-request` module, which provides the end-user interface. Depending on whether the end-user request is for measuring upstream or downstream path performance, the request will be sent either to the `client-side-measurement` module (upstream), or to the `agent-selection` module (downstream). For upstream path performance measurements, the `client-side-measurement` module simply conducts the measurements using one of the techniques implemented in Emodis. The call completes once it receives the measurement results.

For downstream path performance measurement, the process is much more complicated. The `agent-selection` module needs to first decide which agents should be used to fulfill the request, schedule the measurement work among the selected agents (using the `synchronization` module), and then forward measurement requests to them (using the `agent-measurement` module). Upon receiving the measurement requests, the agent will conduct the actual network measurements. The measurements techniques supported by Emodis are ping, traceroute, BRoute [9], Pathneck [8], and IGI/PTR [10], implemented by the corresponding module respectively. Except IGI/PTR, which also uses regular UDP packets (using the `normal-udp-pkt-xfer` module), the other four techniques all rely on the `raw-pkt-transmission` module, which implements basic raw packet sending and receiving functions. As shown in dashed part of Figure 1, other measurement techniques can easily be added into Emodis.

Besides the above modules, there are three other important modules in the system— `access-control`, `membership-management`, and `diagnosis`. The `access-control` module is used to distinguish between the two types of users that Emodis supports: requesters and managers. Requesters are regular users, while managers are "super-users", such as system administrator, or a small number of authorized researchers who need internal system information. The differences between requesters and managers is that requesters have more limitations in the use of the system: (1) the measurement frequency from a requester is limited (e.g. less than once per five minutes) to avoid abuse; (2) a requester cannot query the internal system status, such as agent load, agent working status, other clients' information, etc.

The `membership-management` module keeps track of active agents and maintains information used by the `agent-selection` module. This module is very important for scenarios like Planetlab [1], where nodes frequently join and leave the system. We will discuss the implementation in more detail in Section 6.

The `diagnosis` module is used to analyze measurement results to determine the causes for network problems. This module is the most important module and, not surprisingly, also the hardest module in Emodis. The diagnosis techniques in the diagnosis module are likely to depend on the application scenario. Figure 1 shows three possible scenarios. The first one can be used by regular end-users to diagnose their end connection. In the second scenario, the master diagnoses problems that can only be detected by correlating the measurement results from multiple clients. In the final example, an ISP diagnoses problems for its customers, possibly in part based on information that the ISP collected from other sources.

We describe Emodis functional modules using regular end users as an example. This system design however can also be used by ISPs. In the ISP scenario, the requests will be submitted by ISP network operators, instead of end users.

2.2 Emodis Deployment Architecture

Based on the discussion of the Emodis functional modules, the system needs to be deployed on three different types of network nodes - we will call them client, agent, and master. The *client* is the node that the end users have access to. *Agents* are nodes that can conduct measurements, i.e., the vantage nodes mentioned above. The *master* oversees the overall operation of the system. As a result, the Emodis system software needs to be separated into three components: client component, agent component, and master component. The mapping between the functional modules and the different types of components may depend on the specific application scenario. For example, the scheduling module can either be placed in the master component or in the client component, depending on the work load we want the master to handle. The mapping of Emodis functional modules to software components and the interactions between the components is what we call

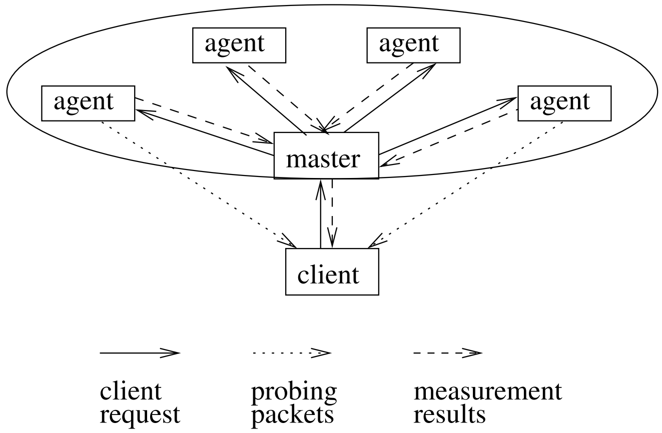

Figure 2: Emodis system architecture

the *Emodis deployment architecture*. In this paper, we only focus on a deployment architecture that supports end user diagnosis. Other possible scenarios include internal ISP deployment or collaborative multi-user deployment.

Figure 2 shows the architecture for end node diagnosis. The features of each type of node are as follows:

1. The client component is the interface between end users and Emodis. The client component has two responsibilities: (1) communicating with the master, including submitting measurement requests to Emodis and receiving measurement results from the system; and (2) conducting active measurements to measure upstream path properties: upstream path performance can only be measured by sending packets from local nodes, so the client component must have this capability.

2. The master component is the central controller of Emodis. It is installed on a pre-determined single machine, i.e., the master. The master accepts client requests, decides which agent nodes are needed to conduct measurement to resolve the requests, and forwards the requests to those agents. When receiving measurement results from agents, the master will first record status information, and then send the results back to the corresponding client. As part of the Emodis scheduling algorithm, the master also serves as the synchronization point as will be described in Section 4.

3. The agent component refers to the software that is installed on all measurement nodes. Each agent passively waits for measurement commands from the master, conducts the corresponding measurements, and sends the measurement results back to the master. Agents also participates the task scheduling in Emodis (see Section 4 for details).

Note that in the above procedure, every client request needs to go through the master. The purpose of this design is three-fold: (1) it significantly simplifies the synchronization among agents since

Table 1: Client Request Parameters

Measurement techniques	
PING	measure path delay
TRACEROUTE	measure path route
IGI	measure end-to-end available bandwidth using IGI/PTR
PATHNECK	locate path bottleneck location using Pathneck
BROUTE	this is a measurement algorithm based on Pathneck, it estimates the available bandwidth for all route branches of an end node
Measurement destinations	
default	the requesting client itself is the measurement destination
explicit	clients can specify one or more destinations to measure
Measurement agents	
explicit	explicitly specify the agents that should be used in measurement
ALL_AGENT	use all available agents to measure specified destinations
ANY_AGENT	use an arbitrary agent to measure specified destinations
AUTO_AGENT	the master automatically selects a diverse set of agents to cover all the edge routes [9] of specified destinations. This is the default mode.
Measurement scheduling	
PM_SYN	measurements from different agents to a common destination need to be synchronized, i.e., no two different agents probe a same destination at the same time; meanwhile, these measurements should be conducted continuously so that they can finish within the smallest time interval
PM_RAN	measurements from different agents do not need to be synchronized, and different agent measure the same set of destinations in random orders
Measurement period	
N	when set with non-zero value, measurement should be conducted every N seconds, this feature is useful for network monitoring

the master is the natural place to serialize measurement operations; (2) except for IGI/PTR measurement, which is a two-end control technique, agents do not need to interact with clients, thus reducing the number of places that can be attacked by malicious users; even in IGI/PTR, for the same security consideration, we always let agent connects to client, while client is not allowed to connect to agent; (3) the existence of a central point makes it very easy to record, aggregate, share, and extract measurement information from different clients. Of course, this centralized design also has drawbacks, such as being performance bottleneck and single-point of failure. We plan to use well-maintained server machine as the master to partly address this problem.

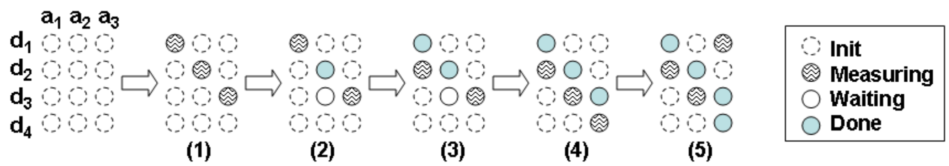

Figure 3: Example of the scheduling algorithm

3 Client Interface

The client interface provides a set of configuration options that clients can use to control how measurements should be conducted. The options allow the selection of the measurement technique, measurement destinations, measurement agents, measurement scheduling, and measurement period (refer to Table 1).

- **Measurement technique:** It is used to specify the probing technique that should be used for agent measurements. Emodis supports five techniques: ping, traceroute, BRoute, Pathneck, and IGI/PTR.

- **Measurement destinations:** This option allows client to specify the destinations that should be measured.

- **Measurement agents:** This option is used to specify the agents that should be used in the measurements. There are four ways to do it. The first method is to explicitly list the IP addresses of the corresponding agents. The other three methods are listed in Table 1. They do not need client to know the agent IP addresses, but instead, the master will decide which agents should be used. Among them "ALL_AGENT" and "ANY_AGENT" are self-explanatory. The most interesting option is "AUTO_AGENT", which means that the selected agents should cover all the clients' edge routes that could possibly be used. In this case, the selected agents should be as diverse as possible. In Emodis, this is implemented as the landmark selection algorithm, as discussed in the BRoute system [9].

- **Measurement scheduling:** As listed in Table 1, PM-SYN and PM-RAN are two parameters that are used to control the scheduling algorithm in Emodis. They control whether the measurements from different agents should be synchronized. We discuss their implementation in Section 4.

- **Measurement period:** Some measurement task needs to be conducted periodically, and this parameter is used to specify how often measurements should be repeated.

4 Scheduling

Scheduling in Emodis specifies the execution times of client requests. Currently, Emodis treats all clients equal and processes client requests based on an FIFO strategy. We plan to introduce priority among client requests in our future work. The focus of Emodis scheduling algorithm is to synchronize multiple client requests and to deal with periodical monitoring requests. Scheduling is either controlled by the master or by the agents, depending on whether explicit synchronization is needed.

If a measurement request does not need to be synchronized (i.e., specified by PM_RAN), the scheduling is managed by the agents. Each agent maintains a task queue, where each task is a measurement request forwarded by the master. The tasks in the queue are sorted by their starting times. Agent always works on the measurement listed at the head of the queue. After that measurement is done, the agent recomputes the starting time of the next round of measurements for that task and reinserts it into the task queue. It then moves on to the next measurement task in the queue.

If a measurement request needs to be synchronized as specified by PM_SYN, scheduling is managed by the master. The master decides which agent conducts measurements, to which destination, and at what time, thus naturally synchronizing the measurements from different agents. If there is only one client request, measurement scheduling is simply sequential. When there are multiple client requests to serve, to reduce the overall service time and to maximize the utility of the agents, Emodis schedules as many measurements as possible in parallel in a simple round-robin fashion.

The algorithm works as follows. Assume we want to use m agents a_i ($1 \leq i \leq m$) to measure n destinations d_j ($1 \leq j \leq n$) in a synchronized manner. This algorithm first lets a_1 measure d_1, a_2 measure d_2, ..., a_n measure d_n (assume $n < m$ for now). After a_i finishes measuring d_i, it is ready to measure d_{i+1}. The master then checks whether a_{i+1} has finished measuring d_{i+1}. If yes, a_i can proceed; otherwise, it has to wait until a_{i+1} finishes. For the same reason, when a_i finishes measuring d_i, the master will also check whether a_{i-1} is waiting for a_i. If yes, a_{i-1} should be allowed to proceed. When all agents $a_1, ..., a_n$ finish their measurements, agents $a_{n+1}, ..., a_{2n}$ can start the same procedure. When $m < n$, the procedure is simpler, since only one phase of measurements is needed. With this scheduling algorithm, the agents are parallelized as much as possible, and the measurements for each destination are all consecutive so that the measurements can be finished within the shortest possible time interval.

Figure 3 shows an example of this scheduling algorithm. Here we want to measure four destinations d_1, d_2, d_3, d_4 using three agents a_1, a_2, a_3. The scheduling algorithm works as follows:

- In step (1), a_1 is assigned to measure d_1, a_2 measures d_2, and a_3 measures d_3.
- In step (2), a_2 finishes its measurement and is ready to measure d_3. Since a_3 is still working with d_3, a_2 has to wait.
- In step (3), a_1 finishes measuring d_1, and immediately starts to measure d_2 since no agent is currently measuring d_2.
- In step (4), a_3 finishes measuring d_3, moves on to measure d_4, at the same time, a_2 is released to measure d_3.
- In step (5), a_3 finishes measuring d_4 and moves back to measure d_1.

5 Implementation of Measurement Techniques

Except IGI/PTR, all the other measurement techniques implemented in Emodis relies on raw packet transmission. Emodis implements these functions in a shared module—`raw-pkt-transmission`. It implements two basic functions: raw packet transmission and raw packet reception. The sending function takes five input parameters—packet size, packet number, TTL value, packet type, and sending rate. The packet type is used to specify whether the probing packets are ICMP ECHO packets or UDP packets. These two types of packets can generate two types of response packets: ICMP ECHOREPLY packets and ICMP TIMEXCEED packets. Consequently, the receiving function only accepts these two types of response packets. For each packet sent out or received, the sending and the receiving functions record the sending time and receiving time. To match the response packet with the out-going probing packet, the sending function uses a sequence number in the packet head, which is also included in the corresponding response packet.

With this `raw-packet-transmission` module, it is easy to implement several measurement techniques: ping only needs ICMP ECHO packets; traceroute uses UDP packets with TTL values set; Pathneck and BRoute need ICMP ECHO packets and UDP packets with TTL values properly set; and IGI/PTR uses UDP packets. Here traceroute is implemented slightly differently compared with the standard implementation—our implementation sends out all UDP packets together, in order to speed up the execution of traceroute.

The implementation of IGI/PTR is different with the other probing techniques due to the fact that IGI/PTR is a two-end control measurement algorithm. Each run of this algorithm needs to send multiple probing packet trains, and after finishing sending each packet train, the sending node needs the receiving node to send back feedback information to adjust the sending rate for the next probing packet train. We made two design decisions for the implementation of IGI/PTR in Emodis: (1) we allow an agent to establish a TCP connection with client for feedback transmission; the connection can only be initiated by agents, which is for security consideration; (2) we allow IGI/PTR to use normal UDP packets as probing packets, since it does not need special features of IP packets; this allows non-root users to at least conduct available bandwidth measurements.

6 Node Dynamics

Nodes in Emodis can join and leave dynamically, due to reasons like software crash or machine shutdown. Depending on the type of node, Emodis responds differently:

Client departures. Impatient client could simply kill the client-side process and leave the system when the measurement time exceeds the client's tolerance. In this case, the master will get a socket-level disconnection signal (SIGPIPE) when the client kills the connection. Based on this signal, the master will carry out the following operations: (1) clean up the data structures for the corresponding client, (2) discard any measurement data for this client, and (3) if this client request is a PM-RAN periodic task which is managed by the agents, the master will send a client-stop command to all involved agents, instructing them to remove the related task from the task queue.

Master departures. Since the master is the "brain" of Emodis, a crash of the master will disable the entire system. In this case, both clients and agents will get a socket disconnect signal. Based on this signal, all working clients shall exit. Agents do not exit—instead they clean up all their task queues, and enter an idle state. In the idle state, the agent will periodically try to connect to the master (the IP address of the master is fixed), so that when the master comes back up, all agents will automatically connect and register with the master. Since there could be many agents, this method allows us to reactivate the system by only rebooting the master, avoiding restarting each agent, which is tedious and time consuming when their number is large.

Agent departures. When an agent crashes, the master will be informed by the socket disconnect signal. The master will (1) clear the corresponding data structures for this agent; (2) see if any client is blocked waiting for this agent, and if so, release the client.

The above mechanism relies on the socket disconnect signals triggered by software crashes. These signals are not triggered if the disconnection is caused by problems in the middle of the network, as is explained in Section 5.12-5.16 in [18]. To deal with this case, we implement an application level keep-alive protocol between the master and the agents. That is, each agent periodically sends a KEEP_ALIVE message to the master, which updates the corresponding timer for the agent, and sends a KEEP_ALIVE_REPLY message back to the agent, which also keeps a timer. If the agent timer expires, the agent considers its connection to the master broken. In that case, it will close the socket, clean up the task queue, and start up the reconnect timer. Similarly, if the timer of an agent in the master expires, the master regards that agent as dead, and removes the corresponding state information for that agent. By properly setting the waiting time periods in the master and the agents, we can make sure that the master's timer expires later than the agent's timer but before the agent's reconnection attempt, so that the agent can successfully reconnect when its network connection resumes after a disconnection.

7 Related Work

There are several end-based systems that provide network measurement and monitoring capabilities. These include Scriptroute [17], NIMI [15], [2], and [12]. Scriptroute [17] and NIMI [15] provide infrastructure support for end users to develop and deploy their own measurement techniques. [2] discusses how to manage and share the huge amount of measurement data that has been collected by different research groups. Besides the architectural issues, it also considers many privacy issues, which are very important in monitoring systems. [12] considers collecting data from regular end users by monitoring traffic on their end hosts. Its focus is on supporting a communication infrastructure that systematically deals with measurement data uploading and downloading.

The main difference between Emodis and these systems is that Emodis targets regular end-users instead of the developers and researchers of network measurement techniques. Emodis implements all the measurement and scheduling functionality, and end-users only need to submit requests with a measurement configuration, but do not need to change the way measurements are conducted.

For end-system-based network diagnosis, we are only aware of one diagnosis system—PlanetSeer [20]. PlanetSeer periodically conducts traceroute measurements to end nodes that accessed a popular web proxy, and uses the information from the web proxy to detect changes in end-to-end performance. When it detects anomalies for an end connection, it uses its route measurements to identify the possible locations of the fault. The difference between PlanetSeer and Emodis is that the measurement module in PlanetSeer is manually managed and there is no synchronization between different measurement modules.

As for the detail measurement techniques, Emodis currently only implements five probing techniques, but it can integrate many other probing techniques such as [11, 16, 19, 3, 5, 13].

8 Conclusion and Future Work

In this paper, we described Emodis—a distributed network monitoring and diagnosis system. We explained the functionality of the three types of nodes in the system: client, master, and agent. In particular, we described the client interface that allows end users to request and configure measurements; the scheduling algorithm that manages synchronized measurements; and the ways Emodis deals with node departures.

However, we only discussed the system architecture of Emodis. The diagnosis and performance aspects of the system are future work. Specifically, we will:

1. Develop and study diagnosis rules for different types of end network failures.
2. Address performance issues such as response time under different measurement loads, the number of clients Emodis can support, etc.
3. Demonstrate that Emodis is really useful by doing a sequence of case studies. For example, we hope to explore how Emodis can help P2P applications such as ESM [4]. We also plan to make Emodis publicly available so end users can use it to help diagnose network problems.
4. Continue to refine the system architecture. This includes adding features such as a better client interface (e.g., using SOAP to implement an XML interface), adding security, allowing clients and client requests to have priority, and providing a better way to share and log measurement information.
5. Add more measurement techniques, such as the packet loss measurement and location tool–Tulip [14], to our system.

References

[1] Planetlab. `https://www.planet-lab.org`.

[2] M. Allman, E. Blanton, and W. M. Eddy. A scalable system for sharing internet measurements. In *Proc. PAM*, March 2002.

[3] R. Carter and M. Crovella. Measuring bottleneck link speed in packet-switched networks. Technical report, Boston University Computer Science Department, March 1996.

[4] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the Internet using an overlay multicast architecture. In *Proc. ACM SIGCOMM*, August 2000.

[5] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure? In *Proc. of ACM INFOCOM*, April 2001.

[6] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. Netscope: Traffic engineering for ip networks. *IEEE Network Magazine, special issue on Internet traffic engineering, 2000*, 2000.

[7] C. Fraleigh, C. Diot, B. Lyles, S. B. Moon, P. Owezarski, D. Papagiannaki, and F. A. Tobagi. Design and deployment of a passive monitoring infrastructure. In *Proceedings of the Thyrrhenian International Workshop on Digital Communications*, pages 556–575. Springer-Verlag, 2001.

[8] N. Hu, L. Li, Z. Mao, P. Steenkiste, and J. Wang. Locating Internet bottlenecks: Algorithms, measurements, and implications. In *Proc. ACM SIGCOMM*, August 2004.

[9] N. Hu and P. Steenkiste. Exploiting internet route sharing for large scale available bandwidth estimation. Under submission.

[10] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, 21(6), August 2003.

[11] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. In *Proc. ACM SIGCOMM*, August 2002.

[12] C. R. S. Jr. and G. F. Riley. Neti@home: A distributed approach to collecting end-to-end network performance measurements. In *Proc. PAM*, April 2004.

[13] K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *Proc. of the USENIX Symposium on Internet Technologies and Systems*, March 2001.

[14] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level internet path diagnosis. In *Proc. SOSP*, October 2003.

[15] V. Paxson, A. Adams, and M. Mathis. Experiences with nimi. In *In Proceedings of the Passive and Active Measurement Workshop*, 2000.

[16] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Proc. PAM*, April 2003.

[17] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A public internet measurement facility. In *USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[18] W. R. Stevens. *Unix Network Programming, Third Edition , Volume 1*. Prentice Hall.

[19] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proc. ACM IMC*, October 2003.

[20] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. Planetseer: Internet path failure monitoring and characterization in wide-area services. In *Proc. of OSDI*, December 2004.