Resource Management in Multi-hop Ad Hoc Networks

David A. Maltz

November 21, 1999
CMU-CS-00-150

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

An ad hoc network is a collection of wireless mobile nodes dynamically forming a temporary network without the use of any existing network infrastructure or centralized administration. Routing protocols have been developed that allow such nodes to communicate, but these protocols typically provide only best-effort service. In particular, they do not provide a way to control the consumption of resources in the network, such as the battery power or the carrying capacity of the nodes. This proposal describes a scheme of *path-state* and *flow-state* mechanisms that can used to explicitly manage resources in an ad hoc network, explains how the scheme can be evaluated, and argues for its importance.

This work was supported in part by the National Science Foundation (NSF) under CAREER Award NCR-9502725, the Caterpillar Corporation, and by the Air Force Materiel Command (AFMC) under DARPA contract number F19628-96-C-0061. David Maltz was also supported under an Intel Graduate Fellowship. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, AFMC, DARPA, Caterpillar, Intel, Carnegie Mellon University, or the U.S. Government.

Keywords: QoS, resource management, multi-hop ad hoc networks, Dynamic Source Routing (DSR)

Contents

1. Introduction

The need to exchange information outside the typical wired office environment is growing. For example, a class of students may need to interact during a lecture; business associates serendipitously meeting in an airport may wish to share files; or disaster recovery personnel may need to coordinate relief information after a hurricane or flood. The CMU Monarch Project's [38] research in wireless multi-hop ad hoc networks is directed at providing that connectivity. Most current protocols provide best-effort routing in ad hoc networks. However, these protocols do not optimize resources that are important to the networks, such as the battery life at the nodes or the available forwarding capacity. This proposal explains how to improve the Quality of Service that ad hoc networks can offer by solving a subproblem which we call *resource management*.

In a typical ad hoc network, potentially mobile nodes come together for a period of time to exchange information. While exchanging information, the nodes may continue to move, so the network must be prepared to adapt continually. Since networking infrastructure such as repeaters or base-stations will frequently be either undesirable or not directly reachable, the nodes must be prepared to organize themselves into a network and establish routes among themselves without any outside support.

In the simplest cases, the nodes may be able to communicate directly with each other. In general, however, ad hoc networks must also support communication between nodes that are only indirectly connected by a series of hops through other nodes. In general, an ad hoc network looks like a network in which every mobile node potentially is also a router, and all nodes run a routing protocol. Unfortunately, standard routing algorithms work poorly in a mobile environment in which network topology changes may be drastic and frequent as the individual mobile nodes move about.

The Dynamic Source Routing protocol (DSR) [21, 20, 4] is unique among the current set of routing protocols for ad hoc networks in that it uses source routes to control the forwarding of packets through the network. The key advantage of a source routing design is that intermediate nodes do not need to maintain up-to-date global routing information, since the packets themselves already contain all the routing decisions. This fact, coupled with the *on-demand* nature of the protocol, dramatically lowers the overhead of the protocol by eliminating the need for the periodic route advertisement and neighbor detection packets that are present in other protocols. Beyond this, the source route on each packet describes a path through the network. Therefore, with a cost of no additional packets, every node overhearing a source route learns a way to reach all nodes listed on the route. Simulation studies of four routing protocols [6] on a wide class of scenarios show that DSR has by far the lowest overhead in terms of routing packets sent, while simultaneously delivering the greatest percentage of data packets among the four protocols.

The remainder of this proposal is organized as follows. The next two sections describe mechanisms for explicitly managing resources in an ad hoc network using DSR, and the assumptions on which those mechanisms are based. Section 4 then presents a formalization of the resource management problem. Sections 5 and 6 end the proposal by explaining how the mechanisms can be evaluated, and discussing the related work. The three appendices provide more detailed information for the interested reader. Appendix A gives a brief overview of the current definition of the DSR protocol. Appendix B describes in practical terms the problems DSR still faces, and provides data supporting the need for explicit resource management. Finally, Appendix C presents the current design of the path-state and flow-state mechanisms.

2. Providing Explicit Resource Management

While DSR has shown itself to be an excellent routing protocol in the scenarios we have studied, it currently provides only best-effort shortest path routing. Experience has shown that simply providing best-effort routing is often not enough, and that significant performance improvement could be obtained by explicitly managing the network resources. For example, even what appears to be a light traffic load can cause congestion in parts of the network unless the network explicitly routes traffic to avoid congestion by managing available forwarding capacity. Likewise, energy consumption is a critical resource for battery-powered nodes, yet no working routing protocols currently take energy levels into account when deciding which nodes will expend energy forwarding packets on behalf of other nodes.

For these reasons, we will use battery power and available capacity as examples to motivate two complimentary approaches to the problem of resource management in ad hoc networks. The first approach is to have nodes in the network send information about their current state back to the sender as needed, which allows *the sender* to direct its packets along the routes that it believes make the best use of network resources. The second approach is to require a sender to reserve resources at the intermediate nodes before sending data, which allows *the intermediate nodes* to control how many resources they must expend.

While the two approaches allow us to address a wide variety of QoS issues, they can both be implemented by extending DSR with a single framework that dovetails elegantly into DSR's existing on-demand structure. The framework is based on the introduction of two kinds of soft-state, called *path-state* and *flow-state*, at the intermediate nodes in the network. Path-state enables the first approach to resource management, while flow-state enables the second approach.

Path-state allows intermediate nodes to forward packets according to a predetermined source route, even though most packets do not include the full source route. Conceptually, the originator of each data packet initially includes both a source route and a unique *path identifier* in each packet it sends. As intermediate nodes forward the packet, they cache the source route from the packet, indexed by the path identifier. The source can then send subsequent packets carrying only the path identifier, since intermediate nodes will be able to forward the packet based on the source route for the path that they have cached.

While path-state allows the elimination of the source route from each packet, thereby reducing the overhead of the DSR protocol, it also provides a way for sources to monitor the state of each path through the network. We envision a system where, as a packet propagates through the network, each intermediate node updates a set of *path-metrics* carried by the packet to reflect the local network conditions seen at the node. These metrics are reflected back to the sender by the destination, along with the path identifier, and enable the sender to track the value of these metrics for each of the source routes it is currently using. We are currently experimenting with metrics that are easy for nodes to measure, that require constant size to represent regardless of source route length, and that allow the sender to pick appropriate routes through the network for its packets.

Flow-state allows a source to differentiate its traffic into *flows*, and to then request better-than-best-effort handling for these flows. With the additional information provided by the flow-state, the network will be able to provide admission control and *promise* a specific Quality of Service (QoS) to each flow. Since the ad hoc network may frequently change topology, the flow-state mechanisms are directly integrated into the routing protocol to minimize their reaction time and to provide *notifications* to a flow when the network must break its promise for a specific level of QoS.

This proposal explains how the path-state and flow-state extensions can be used together to explicitly manage the resources in an ad hoc network. We propose to demonstrate the techniques on two particular resources: the battery power available at each node, and the available forwarding capacity of each node. If the techniques are successful, the evaluation will show they provide the following practical benefits:

- More efficient and fair use of battery power and available capacity,
- Reduction in congestion "hot-spots", and
- Less jitter in the interarrival rate of packets.

The combination of simulation and emulation experiments described in Section 5 will lead to understanding of how different types of resources can be managed in an ad hoc network. In particular, we expect to find that battery power is best managed by using the feedback to senders provided by path-state mechanisms, and that available forwarding capacity is best managed by admissions control using the flow-state mechanisms.

3. Assumptions

The general problem of resource management in networks is not novel, and prior approaches to solving it are described in the related work section (Section 6). However, the environment of a multi-hop ad hoc network is sufficiently different from those previously studied that the set of assumptions we must make are very different. These different assumptions have driven the design of my mechanisms for resource management, the path-state and flow-state mechanisms, in a different direction from that most others have taken. The key assumptions that drove the design of the path-state and flow-sate mechanisms are described below.

Transmitter and receiver time are the critical resources: Unlike wired networks, where the carrying capacity of the links may well outstrip the processing power of the routers, the mismatch is in the other direction for the wireless networks in which we are interested. While both wireless link capacity and CPU speed are on exponential growth curves, the CPU speed curve is significantly ahead of and with faster growth than the wireless capacity curve. Unlike wired networks, where the amount of processing per packet must be kept to minimum, in low bandwidth, high mobility wireless networks, saving a single transmission is worth a significant amount of memory or CPU cycles spent anywhere in the network. This is true even for the smallest devices imaginable. Such devices will likely be battery constrained, and while chip-making technology is reducing the power consumption of processing and storage components, the power consumption of wireless transceivers is not only decreasing more slowly — it is bounded by the physical laws of electromagnetic propagation.

Topology change may be frequent: The path-state and flow-state mechanisms must be able to quickly adapt to changes in network topology, since topology changes can be caused either by movement of the nodes or by the variability of wireless propagation. For example, our ad hoc network testbed, which has only 8 nodes, experienced a motion-induced route change more frequently than once per 30 seconds and link status changes that occur almost constantly.

Link capacity varies widely over time: Our solutions must work in environment with a shared media, so that the link capacity seen by each node's network interface is affected by transmissions or noise from external sources, and even from other nodes that are within carrier-sense range.

Traffic load is sufficiently stationary to enable time-average measurement: The load offered to each node in the network must change sufficiently slowly that the node can meaningfully measure the load by averaging samples over time. Beyond this aggregation constraint, the design of the path-state and flow-state mechanisms is made easier if we can assume that the load offered to the network will be roughly stationary, or at least will change more gradually than than topology shifts. If the characteristics of the offered traffic change at the same rate as the external forces change the carrying capacity of the network, there is no way to build a feedback system that will adapt the offered traffic to the carrying capacity. Experimentation will be required to determine how much this assumption can be relaxed before performance begins to suffer.

Sufficient media access and packet scheduling schemes are available: Path-state and flow-state are network layer mechanisms. However, they will need to be realized on top of practical link layers and packet scheduling algorithms. We believe the mechanisms are general enough to be implemented on any type of MAC protocol, with a suitable set of metrics (Section C.2.3). As described in this proposal, the mechanisms target the most difficult case of contention-based MAC protocols, such as the IEEE 802.11 DCF MAC protocol [18]. If an efficient TDMA protocol were developed for multi-hop ad hoc networks, it would greatly simplify the work of monitoring the available resources at each node to provide inputs to the network layer. It would not, however, substantially change the problems by the path-state and flow-state mechanisms, nor the need for such mechanisms to explicitly manage the resources of the ad hoc network.

4. Problem Formalization

This section describes a formalization of the resource management problem in ad hoc networks, focusing on the resources of available capacity and battery level. Currently, the formalization is useful for evaluating the effectiveness of the path-state and flow-state mechanisms for resource management, as described in Section 5. Additionally, this formalization helps to characterize the class of resources that the path-state and flow-state mechanisms can successfully manage. We expect that other network resources whose management can be formalized in the same way as battery level and available capacity will also benefit from the mechanisms.

In the most abstract frame, the resource management problem is that of optimizing a function, which we call a *resource constraint*, over a set of nodes, packet, or routes. For example, let

$$n = \text{number of nodes}$$

$$P = \text{set of data packets the network attempts to deliver}$$

$$b(i) = \text{ battery level at node number i.}$$

The goal of equalizing the consumption of battery power in the network in order to maximize the time until a node runs out of power can then be expressed as

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (b(i) - b(j))^2$$

where we assume n is limited to the set of nodes that run off battery power (nodes with effectively infinite power supplies should be treated differently). Singh *et al.* [42] lists other expressions that could be turned into resource constraints for balancing power consumption.

There are many ways to express the desire to prevent congestion of the network. One method that we expect will work well, while also improving the quality of audio connections, is to minimize jitter in the network. If

$$v(n) = \text{ variation in packet forwarding time at node n, then}$$

$$\text{minimize} \sum_{p \in P} \sum_{n \in path(p.s \rightarrow p.d)} v(n)$$

where $p.s$ is the packet's source and $p.d$ the packet's destination, and $path(p.s \rightarrow p.d)$ is the set of nodes on the path the packet uses to traverse the network .

In order to prevent the degenerate case where the functions are optimized by simply transmitting no packets, the optimization problem must also include at least two *progress constraints.* First, the network must attempt to maximize the number of packets it attempts to deliver. Second, the network must maximize the fraction of accepted packets that it then successfully delivers to the packets' destinations . In comparing the various approaches to resource management, there will clearly be trade-offs between optimizing the resource constraints and optimizing the progress constraints. For example, this means that sometimes the network may carry less traffic in order to reduce jitter.

The practical difficulty in optimizing these functions in a real network is that the information required to compute the value of the functions is distributed among all the nodes in the network. The traditional formulation of the problem of computing routes in the network is itself a similar distributed computation problem, and there are two classes of classic routing algorithms that solve the distributed routing problem: Distance Vector and Link State [46]. In the routing problem, each node has local information, namely the set of its neighbors, that must be shared with the other nodes in order to compute routes. For the networks of the size in which we are interested, either the Distance Vector or the Link State algorithms could be trivially modified to carry resource information and thereby attempt to solve the resource management problem.

However, our previous work [6] has shown that in ad hoc networks, which have relatively low bandwidth and rapidly varying topology, Distance Vector algorithms often do not even converge to form stable routes. It is extremely unlikely that they would be able to successfully manage resources in the network whose values change even more quickly than each node's set of neighbors. Likewise, we have shown that the cost of maintaining accurate neighbor sets, as a Link State algorithm must do, is very high in ad hoc networks.

This proposal primarily seeks to make practical contributions to improving the quality of service provided by ad hoc networks. However, in doing so, it will indirectly evaluate the ability of the path-state and flow-state mechanisms to approximate solutions to the underlying optimization problems in the environment of an ad hoc network.

5. Evaluation Mechanisms

The objectives of this proposal are the design of the path-state and flow-state mechanisms and the evaluation of their ability to successfully manage the resources of a mobile ad hoc network. In doing so, the research will also generate insights into the class of resources that each mechanism is best suited towards managing, while simultaneously improving the performance and usefulness of the DSR protocol in general.

We plan to evaluate the path-state and flow-state mechanisms using a combination of simulation and emulation [25, 29]. After simulating the mechanisms on simple scenarios and refining the mechanisms as directed by the data, we will study the impact and value of the path-state and flow-state mechanisms in the examples explained below.

Using our technology for emulation of ad hoc networks, we will be able to carry out the final evaluation of the mechanisms on larger networks and with significantly less difficulty than a full-scale test-bed evaluation [27, 28] would require. In addition, emulation provides excellent experimental control and repeatability, and it allows us to make alterations to any part of the network stack if it becomes necessary, including the MAC protocols and physical layers. This flexibility permits experimentation with radio interfaces that are not available today, but that will exist in the future.

5.1. Improving DSR's Performance

The first test in evaluating the path-state mechanism will be determining whether it successfully reduces byte overhead while not significantly increasing packet overhead. We expect that path-state will reduce the byte overhead of DSR by a factor of 5 to 10, since the majority of packets that now carry a 40-byte source route will instead carry a 4- to 8-byte extension header containing only a path-identifier. Using the smallest encoding of the path-identifier [5], there is the potential to completely eliminate overhead from data packets.

The reduction in byte overhead will have several benefits. First, it will reduce or eliminate the per-data packet overhead from DSR. Second, it will improve battery life by decreasing the length of time transmitters must be on. Finally, reducing byte overhead is likely to improve the performance DSR under high loads. Johansson *et al* [19] report that at high offered loads, AODV performs better than DSR in terms of delay and packet delivery ratio. They attribute the performance difference to the large number of bytes consumed by source routes in DSR.

5.2. Balancing Battery Life

In order to measure the ability of the path-state mechanisms to manage the battery resources of an ad hoc network, we must first extend our simulator to track the energy level available at each node. Work on this is already under way as part of VINT development effort [11]. This framework enables us to experiment with algorithms for retrieving routes from the route cache. The algorithms will use the path-metrics stored in the route cached by the path-state mechanisms to select routes that balance the energy levels of the nodes over time.

We will evaluate the improvement due to the explicit resource management by comparing the performance of DSR enhanced with path-state mechanism to the performance of unenhanced DSR running over the same scenarios. In order to determine how effective the path-state mechanism is at obtaining all the possible gains, we will compare its performance against the "gold-standard" of an omniscient routing protocol.

5.3. Managing Congestion in the Network

We will compare the ability of the path-state and flow-state mechanisms to manage congestion in the network via an approach similar to that used for battery life. For each scheme, we will compare the total amount of data it attempts to send to the destination and the amount that successfully reaches there. We expect to find that the flow-state mechanism is more conservative in the amount of traffic it sends (since it uses admission control), but that it is more successful at controlling congestion and so drops fewer of the admitted packets.

5.4. Providing Better-than-Best Effort Traffic

By using path-state and flow-state as resource management techniques, we expect to be able to show a point-to-point audio connection of usable quality across a multi-hop ad hoc network in the face of competing traffic and topology change. As the first step to reaching this goal, we will determine whether the network is able to provide flows with the Quality of Service it has promised via a detailed analysis of the variability in jitter and throughput experienced by packet trains traversing the network.

5.5. Providing Useful Feedback to Higher Layers

Finally, we hypothesize that the path-metrics measured by the network layer should improve the ability of protocols above the network layer to operate over multi-hop ad hoc networks. In order to evaluate this hypothesis, we propose to make the path-metrics available to higher layer protocols that have been adapted to respond to such direct feedback and then measure the difference in end-to-end performance of the protocols. The two obvious candidates for higher layer protocols to experiment with are TCP, due to its well studied nature, and applications in the Odyssey [31] suite, since they are already designed to react to changing network conditions.

To quantify the potential impact of feedback from the network layer higher layer protocols, we can use examples from TCP throughput studies. In our testbed, we have measured scenarios in which the feedback could improve throughput by 20-30% [27]. Improvements of 200-300% are reported in the literature [24].

Preliminary experimentation with DSR has already shown how the addition of network layer feedback readily suggests improvements that could be made to the higher layer protocols to improve their adaption and performance. Many other groups are working on improvements to TCP's mechanisms, and we see our role as supporting their work by delivering additional information from the network layer. Our focus is on how the single framework of path-state and flow-state can be leveraged to support numerous improvements through-out the network stack.

6. Related Work

There has been a large amount of research on improving the Quality of Service and power consumption at the link layer [9, 3, 41]. This work is largely complimentary and orthogonal to ours, as the path-state and flow-state improvements to DSR can build on top of whatever QoS support the lower layer provides. If the link layer is able to provide stronger QoS semantics, better scheduling, or better utilization metrics to the higher layer, DSR will be able to take advantage of them. However, link layer improvements alone will not reduce the need for network layer signaling that is the heart of this proposed research.

Many other groups have recognized the importance of supporting better than best-effort Quality of Service in ad hoc networks, and are exploring a wide set of possible designs. Compared to this body of work in the large, there are five ideas that distinguish our work. First, we believe we can achieve the desired QoS semantics by framing the problem in terms of the subproblem of resource management. Second, our protocols aggregate the information needed for resource management at the sending node. By avoiding the need to distribute resource information throughout the network, the protocols are completely on-demand. Third, we plan to evaluate and compare two different approaches to resource management. Forth, our proposal tests the hypothesis that QoS mechanisms can take advantage of the detailed information known by the routing protocol if the two are tightly integrated. Finally, the single set of path-state and flow-state mechanisms represent a basic foundation that supports a wide range of highly desirable improvements, and these mechanisms dovetail well with the existing DSR framework.

There is already a large class of designs for QoS in ad hoc networks follow an approach similar to RSVP [48], as proposed for providing Integrated Services in traditional wire-line networks. These designs largely separate the routing protocol from the QoS mechanisms, which operate as a signaling layer on top of routing [30, 45]. The path-state and flow-state mechanisms are integrated directly with DSR, which should enable them to react faster and with less overhead to topology and environmental changes. For example, DSDV+QoS [17], ALP [14], and MMWN [39] use the routing protocol to distribute the QoS parameters of the links, so that each node can identify the network paths that support a particular QoS. These designs use Link State or Distance Vector as routing algorithms, which DSR has been shown to outperform in highly mobile environments. However, MMWN and DSDV+QoS include clustering to aggregate information, which may enable them to scale to larger networks than DSR. Although important for some classes of applications, scalability is not a current goal of the work in this proposal. There are many applications which will benefit from QoS support where scalability is not a significant issue, and other work within the Monarch Project addresses scaling issues in a manner completely consistent with this approach to QoS support.

CEDAR [43] uses a Link State algorithm for QoS, but propagates information only about high-bandwidth, stable links. By using a source routed scheme, our design differs in that QoS and path-metric information is carried only over those paths that require the information.

The INSIGNIA system [23, 22] assumes there is a converged unicast routing topology maintained by an independent routing protocol. It then piggybacks QoS signaling messages onto the data packets in order to reduce control overhead and latency. INSIGNIA also contains an elegant scheme for encoding on each data packet information that can be used to selectively drop packets from a flow should the need arise to lower the bandwidth required by the flow while the packets are in flight. This scheme could be usefully incorporated into DSR and used in addition to the path-state and flow-state mechanisms. INSIGNIA does not provide a method for backtracking QoS setup messages, however, so it can potentially reject admissible connections as described in Section C.3.2.

AODV is a routing protocol for ad hoc networks that combines mechanisms from both DSR and DSDV [35]. However, the two protocols are fundamentally different, as AODV is based on hop-by-hop routing, rather than on source routing like DSR. AODV constructs routes on-demand using a Route Discovery mechanism like DSR's, but its hop-by-hop nature requires it to use hard-state sequence numbers (adapted from DSDV) in order to prevent routing loops. Adding path-state to DSR will reduce source route overhead while maintaining the properties of DSR, such as route shortening and support of asymmetric routes, that AODV does not contain and that rely on the existence of source routes in some form. As future work, it should also be possible to adapt the path-state and flow-state mechanisms to provide QoS support in AODV.

Holland and Vaidya [16, 2] have studied the behavior of TCP in ad hoc networks using DSR as a routing protocol, and identified the need to communicate routing information to the transport layer. Our path-state design generalizes their approach, and applies it to other transport layers, such as voice traffic and Odyssey. Liu and Singh [24] address three of the four pathologic interactions between network and transport layers we have identified, but focus on backwards compatibility. They use a shim layer just below the transport protocol that derives network information from the existing TCP acknowledgment stream and then modulates the packets seen by TCP based on this information. Since we are designing the routing protocol, it makes sense to explicitly export this information, rather than forcing a shim layer to intuit it.

Escobar, Lauer and Steenstrup [10] describe an alternative on-demand system for congestion control in packet networks. Their system works by propagating congestion information hop-by-hop backwards along the route towards the packet source. Every time a node **A** forwards a packet to node **B**, **B** sends back to **A** its estimate of **A**'s share of the bandwidth to the packets final destination. As packets flow through the system, the packets' sources eventually learn the rates at which they can send packets without causing congestion. The path-state mechanism for DSR should enable the same style of congestion control, even when unidirectional links are in use.

The Asynchronous Transfer Mode (ATM) network specification [46] deals extensively with both the use of path-identifiers and the provision of QoS [15]. However, the path-identifiers used in ATM are only unique between pairs of switches, and not globally unique as in our protocol. While it would be possible to completely adopt the ATM model and eliminate global addresses from most data packets, the savings in overhead would probably be more than offset by a loss in ability to deal with rapid topology changes. In terms of QoS, however, ATM has a great deal of guidance to offer the path-state and flow-state mechanisms. In particular, the ATM Available Bit Rate (ABR) service is similar in intent to the feedback provided by the path-state mechanism, and its lessons may be directly applicable.

7. Conclusions

In summary, we have described how the single framework of path-state and flow-state maintenance can be used to explicitly manage resources in a DSR multi-hop ad hoc network. The immediate contributions of this research will be the design of the path-state and flow-state framework, and the evaluation of how successful it is at managing the two resources of battery power and available forwarding capacity. We will simulate the protocols under a variety of scenarios, and will quantify the performance improvements and costs of the best approaches. Using emulation, we will evaluate the impact of the changes on the performance of real voice applications running on physical end-systems. Practically, we expect that managing these resources will extend battery life, reduce the congestion in the network, and ultimately enable usable quality voice traffic across the network.

References

[1] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. Internet Request For Comments RFC 2581, April 1999.

[2] Bikram S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradham. Improving the Performance of TCP over Wireless Neworks. In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS'97)*, pages 365–373, May 1997.

[3] Rusty O. Baldwin, Nathaniel J. Davis IV, and Scott F. Midkiff. A Real-time Medium Access Control Protocol for Ad Hoc Wireless Local Area Networks. *Mobile Computing and Communications Review*, 3(2):20–27, April 1999.

[4] Josh Broch, David B. Johnson, and David A. Maltz. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft-ietf-manet-dsr-02.txt, June 1999. Work in progress.

[5] Josh Broch, David B. Johnson, and David A. Maltz. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft-ietf-manet-dsr-03.txt, October 1999. Work in progress.

[6] Josh Broch, David A. Maltz, David B. Johnson, Yih-chun Hu, and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 85–97, Dallax, TX, October 1998.

[7] Kevin Brown and Suresh Singh. M-TCP: TCP for Mobile Cellular Networks. *Computer Communications Review*, 27(5), 1996. Read from http://www.ece.orst.edu/s̃ingh/papers.html.

[8] M. Scott Corson and Anthony Ephremides. A Distributed Routing Algorithm for Mobile Wireless Networks. *Wireless Networks*, 1(1):61–81, February 1995.

[9] David A. Eckhardt and Peter Steenkiste. Effort-limited Fair (ELF) Scheduling for Wireless Networks. Private communication, July 1999.

[10] J. Escobar, G. Lauer, and Martha Steenstrup. A congestion-control algorithm for receiver-directed packet-radio networks. In *Proceedings of the Tactical Communications Conference. Vol.1. Tactical Communications. Challenges of the 1990's*, pages 69–103, 1990. net24.

[11] Kevin Fall and Kannan Varadhan, editors. *ns* Notes and Documentation. The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC, January 1999. Available from http://www-mash.cs.berkeley.edu/ns/.

[12] Laura Marie Feeney. An Energy-consumption Model for Performance Analysis of Routing Protocols for Mobile Ad Hoc Networks. Private communication, July 1999.

[13] Paul Ferguson and Geoff Huston. *Quality of Service — Delivering QoS on the Internet and in Corporate Networks*. Wiley Computer Publishing, 1998.

[14] J.J. Garcia-Luna-Aceves and M. Spohn. Scalable Link-State Internet Routing. In *Proceedings Sixth International Conference on Network Protocols*, pages 52–61, oct 1998.

[15] Natalie Giroux and Sudhakar Ganti. *Quality of Service in ATM Networks: State-of-the-Art Traffic Management*. Prentice Hall, 1999.

[16] Gavin Holland and Nitin Vaidya. Analysis of TCP Performance over Mobile Ad Hoc Networks. In *The Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, pages pp. 219–230, 1999.

[17] Yu-Ching Hsu, Tzu-Chieh Tsai, and Ying-Dar Lin. QoS Routing in Multihop Packet Radio Environment. In *Proceedings Third IEEE Symposium on Computers and Communications (ISCC'98)*, pages 582–586, June 1998.

[18] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.

[19] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-Based Performance Analysis of Routing Protocols for Mobile Ad-Hoc Networks. In *The Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, pages 195–206. ACM, August 1999.

[20] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In *Mobile Computing*, edited by Tomasz Imielinski and Hank Korth, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[21] David B. Johnson and David A. Maltz. Protocols for Adaptive Wireless and Mobile Networking. *IEEE Personal Communications*, 3(1):34–42, February 1996.

[22] Seoung-Bum Lee and Andrew T. Campbell. INSIGNIA. IETF Working Draft - work in progress, draft-ietf-manet-insignia-00.txt, November 1998. Draft expired May 1999.

[23] Seoung-Bum Lee and Andrew T. Campbell. INSIGNIA: In-band Signaling Support for QOS in Mobile Ad Hoc Networks. In *Proceedings of 5th International Workshop on Mobile Multimedia Communications (MoMuC'98)*, October 1998. Read from web as http://comet.ctr.columbia.edu/insignia/insignia98.ps.

[24] Jian Liu and Suresh Singh. ATCP: TCP for Mobile Ad Hoc Networks. Available from web page??? Personal Communication, June 1999.

[25] David A. Maltz. Emulation of Ad Hoc Networks. Talk delivered at the 45th IETF in Oslo, Norway, July 1999. Slides available from http://ietf.org/proceedings/99jul/toc.htm#P229_7520.

[26] David A. Maltz, Josh Broch, Jorjeta Jetcheva, and David B. Johnson. The Effects of On-Demand Behavior in Routing Protocols for Ad Hoc Networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1439–1453, August 1999.

[27] David A. Maltz, Josh Broch, and David B. Johnson. Experiences Designing and Building a Multi-Hop Wireless Ad Hoc Network Testbed. Technical Report 99-116, School of Computer Science, Carnegie Mellon University, March 1999.

[28] David A. Maltz, Josh Broch, and David B. Johnson. Quantitative Lessons from a Full-Scale Multi-Hop Wireless Ad Hoc Network Testbed. In *Proceedings of WCNC 2000*, September 2000. To appear.

[29] David A. Maltz, Qifa Ke, and David B. Johnson. Emulation of Ad Hoc Networks. Talk delivered at the VINT Project Retreat, June 1999. Slides available from http://www.monarch.cs.cmu.edu/papers.html.

[30] Shree Murthy and J. J. Garcia-Luna-Aceves. A routing architecture for mobile integrated services networks. *Mobile Networks and Applications*, 3(4):391–407, 1999. Read from web as http://www.cse.ucsc.edu/research/ccrg/publications.html.

[31] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, Eric J. Tilton, Jason Flinn, and Kevin R. Walker. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, pages 276–287, St. Malo, France, October 1997.

[32] Vincent D. Park and M. Scott Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proceedings of INFOCOM'97*, pages 1405–1413, April 1997.

[33] Vincent D. Park and M. Scott Corson. A Performance Comparison of TORA and Ideal Link State Routing. In *Proceedings of IEEE Symposium on Computers and Communication '98*, June 1998.

[34] Vincent D. Park and M. Scott Corson. Temporally-Ordered Routing Algorithm (TORA) Version 1: Functional Specification. Internet-Draft, draft-ietf-manet-tora-spec-01.txt, August 1998. Work in progress.

[35] Charles E. Perkins and Pravin Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of the SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, August 1994. A revised version of the paper is available from http://www.cs.umd.edu/projects/mcml/pub.html.

[36] Charles E. Perkins and Elizabeth M. Royer. Ad Hoc On Demand Distance Vector (AODV) Routing. Internet-Draft, draft-ietf-manet-aodv-02.txt, November 1998. Work in progress.

[37] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc On-Demand Distance Vector Routing. In *Proceedings of WMCSA99: 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[38] The CMU Monarch Project. http://www.monarch.cs.cmu.edu/. Computer Science Department, Carnegie Mellon University.

[39] Ram Ramanathan and Martha Steenstrup. Hierarchically-organized, multihop mobile wireless networks for quality-of-service support. *Mobile Networks and Applications*, 3(1), 1998. Journal version of "MMWN Preliminary Design (Draft)" available from ftp.bbn.com/pub/ramanath/mmwn-design.ps.

[40] Srinivasan Seshan, Mark Stemm, and Randy H. Katz. SPAND: Shared Passive Network Performance Discovery. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 135–146, dec 1997.

[41] Suresh Singh and C.S. Raghavendra. PAMAS — Power Aware Multi-Access protocol with Signalling for Ad Hoc Networks. *Computer Communication Review*, 28(3):5–26, 1998.

[42] Suresh Singh, Mike Woo, and C.S. Raghavendra. Power-Aware Routing in Mobile Ad Hoc Networks. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 181–190, 1998.

[43] Raghupathy Sivakumar, Prasun Sinha, and Vaduvur Bharghavan. CEADR: A Core-Extraction Distributed Ad Hoc Routing Algorithm. *IEEE Journal on Selected Areas in Communications*, 17(8):1454–1465, aug 1999.

[44] W. Richard Stevens. *TCP/IP Illustrated, The Protocols*, volume 1. Addison-Welsley, 1994.

[45] Anup Talukdar, B. R. Badrinath, and Arup Acharya. MRSVP: A reservation protocol for an Integrated Services Packet Networks with Mobile hosts. Read from http://www.cs.rutgers.edu/ badri/papers.html. Journal version available as "Integrated services packet networks with mobile hosts: architecture and performance." Wireless Networks. 5(2) March 1999. Pages 111-124.

[46] Andrew S. Tannenbaum. *Computer Networks*. Prentice Hall, third edition, 1996.

[47] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, The Implementation*, volume 2. Addison-Welsley, 1995.

[48] L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. Internet Request For Comments RFC 2205, September 1997.

A. Overview of the Current DSR Protocol

Dynamic Source Routing (DSR) [21, 20, 4] is in a different class than most other routing protocols for multi-hop ad hoc networks in that it uses source routes supplied by a packet's originator to determine the packet's path through the network, rather than using independent hop-by-hop routing decisions made by each node that receives the packet.

In a source routing design, each packet to be routed through the network carries in its header the complete, ordered list of nodes through which the packet must pass. As mentioned in Section 1, the key advantage of a source routing design is that intermediate hops do not need to maintain fresh routing information in order to route packets they receive, since the packets themselves already contain all the routing decisions. This advantage eliminates the need for the periodic route advertisement and neighbor detection packets present in other protocols.

A.1. The Basic Design of DSR

Figure 1 shows the basic operations of the DSR protocol, which we divide into two mechanisms: Route Discovery and Route Maintenance. Route Discovery is the mechanism by which a node **S** wishing to send a packet to a destination **D** obtains a source route to **D**. To perform a Route Discovery, the source node **S** broadcasts a ROUTE REQUEST packet that is flood-filled through the network in a controlled manner and is answered by a ROUTE REPLY packet from the destination node or from another node that knows a route to the destination. Nodes keep the routes they learn in a *Route Cache*, which can store multiple routes to the same destination. Before propagating a ROUTE REQUEST, nodes check their Route Cache for a route that could be used to answer the REQUEST. Route Discovery is initiated only on-demand, meaning that nodes do not expend effort to obtain or maintain routes to nodes they are not currently communicating with.

Route Maintenance is the mechanism by which a packet's sender **S** is able to detect if the network topology has changed such that it can no longer use its route to the destination **D**. For example, a route becomes broken if two nodes listed as neighbors on the route have moved out of range of each other. When Route Maintenance indicates a source route is broken, it notifies **S** with a ROUTE ERROR packet identifying the broken hop in the route. For subsequent packets, **S** can use any other route it happens to know to **D**, or it can invoke Route Discovery again to find a new route.

Since 1994, we have iteratively refined the DSR protocol via designing, simulating, and analyzing optimizations. This work has fleshed out the exact rules required for a correct protocol (e.g., when is it safe for a node reply to a ROUTE REQUEST for another node), and led to a collection of improvements to Route Discovery [26, 20]. The optimizations and extensions to the protocol we have evaluated include:

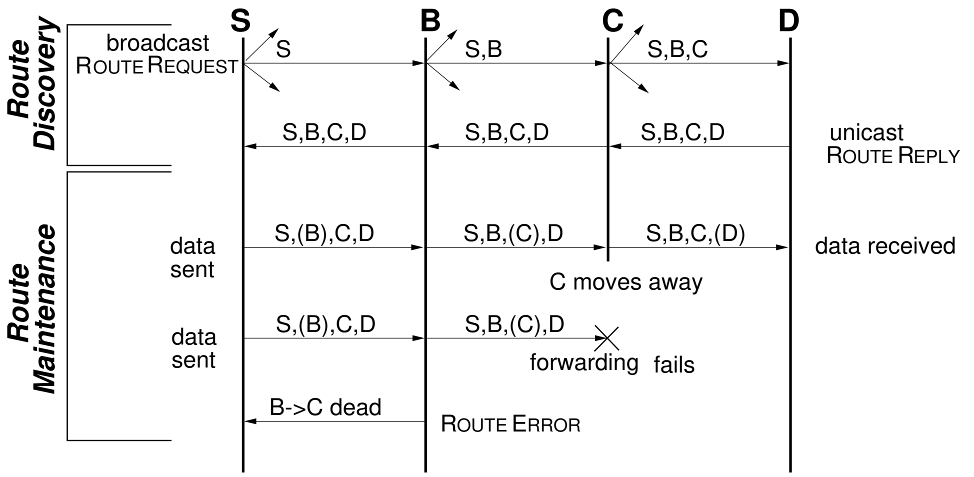


Figure 1 Basic operation of the DSR protocol showing the building of a source route during the propagation of a ROUTE REQUEST, the source route's return in a ROUTE REPLY, its use in forwarding data, and the sending of a ROUTE ERROR upon forwarding failure. The next hop is indicated by the address in parentheses.

- Allowing source routes to be cheaply shortened if communicating nodes move closer together
- Improving the speed at which stale data is removed from the nodes' caches.
- Several different route cache data structures and algorithms, and the rules that govern how the route cache can be used to limit the repropagation of Route Discoveries.
- The use of control message piggybacking to support asymmetric routes.
- Techniques for avoiding ROUTE REPLY storms.

A.2. The Performance of DSR

The combined effect of the optimizations gives DSR significantly better performance than three other ad hoc network routing protocols, when measured on three key metrics. In a detailed simulation study [6], the four routing protocols were each run on networks of 50 nodes, subjected to identical packet workloads and moving in identical patterns. The experimental design allows us to directly compare the performance of the protocols, since they were challenged in an identical fashion. The movement patterns were created by the Random Waypoint algorithm [20] we developed that tends to stress all operating modes of the routing protocols by creating many widely varying network topologies.

A brief description of each of the four routing protocols follows below. Each protocol was selected for inclusion in the study from the set of published proposals for two reasons: each represents a very different point in the design space of ad hoc network routing protocols, and each had been described in enough detail by their designers to permit simulation.

DSDV: Destination Sequenced Distance Vector [35]. A distance vector algorithm modified to guarantee loop freedom even in the face of the rapid topology changes of an ad hoc network. After experimenting with the protocol, we found that a change in the rule for sending triggered updates resulted in better performance. We report the performance of this improved variant, which we call DSDV-SQ.

TORA: Temporally-Ordered Routing Algorithm [8, 32, 33, 34]. A link reversal algorithm that emphasizes minimizing reaction to topology change.

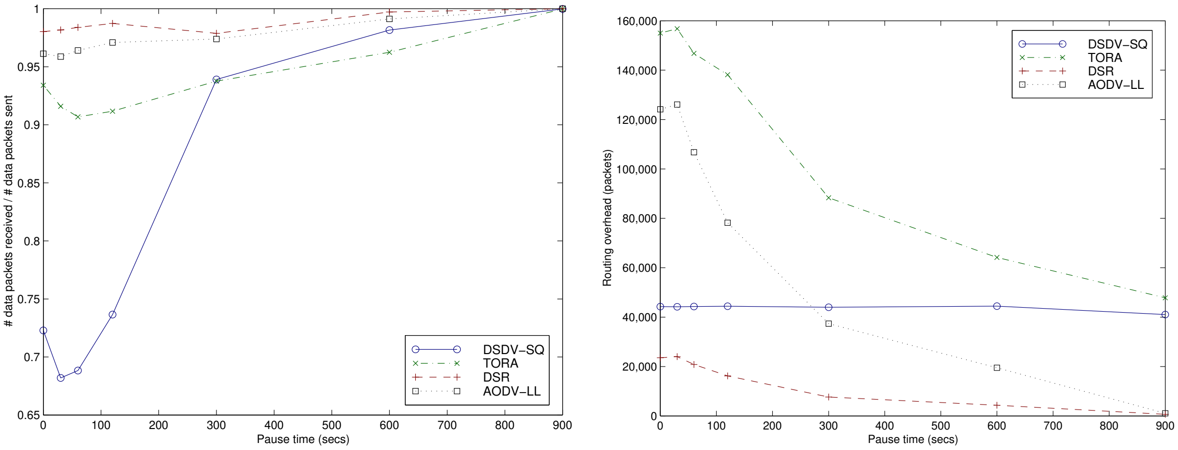
DSR: Dynamic Source Routing. Our routing protocol, as described briefly in Appendix A

AODV: Ad Hoc On Demand Distance Vector [37, 36]. A combination of DSDV and DSR, designed to use the on-demand nature of DSR's Route Discovery mechanism to adapt to ad hoc network topology changes, while using hop-by-hop routing to eliminate the need for source routes. Experimentation with AODV led us to develop an improved variant called AODV-LL that also incorporates ideas from DSR's on-demand Route Maintenance.

Figure 2(a) shows the fraction of the offered packets that were successfully delivered by the routing protocols when subjected to an identical workload of constant bit-rate (CBR) sources during a 900-second simulation of 50 nodes moving at a maximum speed of 20m/s in a site 300m $\times$ 1500m. The x-axis describes the degree of mobility in the scenario, with Pause Time 0 meaning constant motion and Pause Time 900 meaning no node motion. Each point on the curves is the average of 10 runs on different scenarios, with each protocol evaluated on the same set of 210 different scenarios.

Figure 2(b) shows a comparison of the number of routing protocol packets that were sent in order to achieve the packet delivery rates shown in Figure 2(a). The ideal routing protocol would require zero packets, meaning it introduced no overhead on the functioning of the network. Both DSR and AODV-LL have the desirable property that their overhead drops to near zero as node mobility decreases, though AODV-LL's overhead increases much more rapidly than DSR's as node motion increases.

We believe this simulation study establishes DSR as strong foundation on which to build an ad hoc network. On the set of scenarios we evaluated, DSR consistently has significantly less overhead and higher packet delivery rate than the other protocols we studied. Further, careful analysis of the trace files from these simulations established that the measured performances stemmed directly from fundamental properties of the protocols, and was not due to the specific scenarios we used, measurement fluke, or implementation error. Our results have recently been confirmed by Johansson *et al* [19].



(a) Packet Delivery Ratio

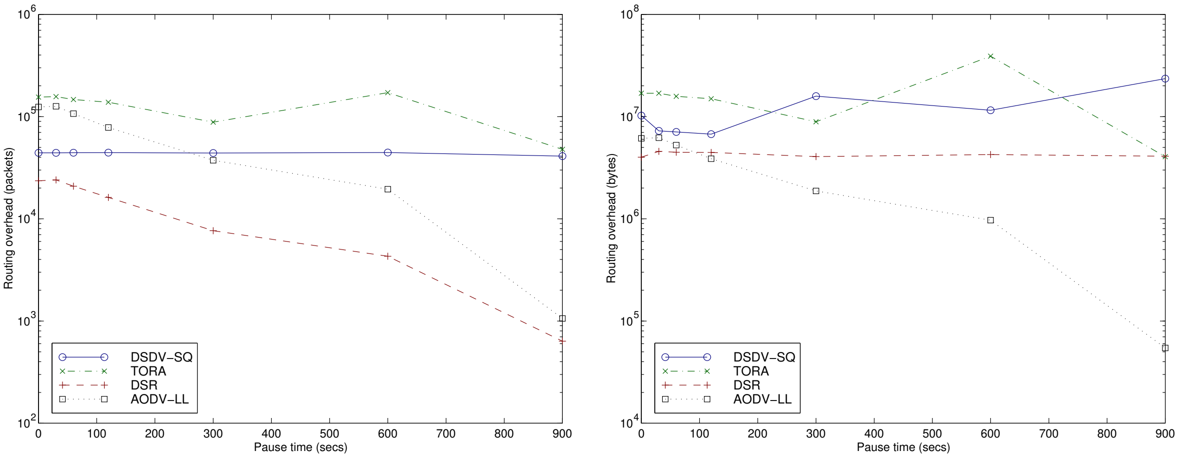
(b) Routing Protocol Overhead

Figure 2 Comparison of routing protocol metrics when the protocols are subjected identical workloads. Pause Time 0 is constant motion and Pause Time 900 is no motion.

B. Unsolved Challenges in Ad Hoc Network Performance

While our work has developed DSR to the point it is a strong foundation for ad hoc networking, it has also uncovered several challenges which we believe explicit resource management via the path-state and flow-state mechanisms will alleviate.

Based on our ongoing analysis of its constituent mechanisms [26], DSR attains its excellent performance from two fundamental factors. First, each packet carries a complete description of a path through the network that not only directs the routing of the packet, but also allows many nodes in the network to learn about the network topology. Second, each node aggressively uses the routes it knows of to limit the extent of Route Discovery, thereby reducing the protocol's overhead.



(a) Overhead in packets

(b) Overhead in bytes

Figure 3 Routing protocol overhead measured in packets and in bytes. The byte count includes all bytes occupied by source routes on data packets. (50 nodes, 20 CBR sources, 64-byte data packets)

Unfortunately each of these factors has an associated cost. In this section, we describe several problems we have observed in DSR's performance that should be remedied by the path-state and flow-state mechanisms proposed here.

B.1. Byte Overhead in DSR

While the source route on each packet distributes valuable information throughout the network, it also adds 12–40 bytes to the headers of each packet. While this is a proportionately small overhead when carried on a large packet (e.g., a MTU-size packet that is part of a bulk data transfer), it is a very large overhead on the smaller packets that are typically used for interactive real-time applications, such as voice communication.

Figure 3 demonstrates the impact of the source route size, by comparing the overhead of the four routing protocols when calculated by counting packets (Figure 3(a)) with the overhead of the protocols when calculated by counting all bytes of data that carry routing protocol information (Figure 3(b)). Both figures are on semi-log scales so the entire range of data can be fit on one axis. Although DSR clearly has the lowest overhead when counted by packets, it has roughly constant byte overhead as a function of mobility rate. This is because the total number of bytes of routing protocol data is dominated by the bytes consumed by the source route on each data packet, which depends on offered load, not on mobility. AODV-LL, which uses hop-by-hop routing rather than source routing, sends 100 times fewer bytes of overhead when nodes are stationary. AODV-LL sends more bytes of overhead than DSR when node mobility is high, however, because its lack of optimizations causes it to send many more routing packets than DSR.

DSR's byte overhead will be dramatically improved by a mechanism that reduces the cost of source routes. However, the mechanism must not prevent the DSR optimizations, which currently assume the presence of source routes, that give DSR its low packet overhead.

B.2. Load Concentration in DSR

Although the use of caching to limit Route Discovery decreases the number of routing messages sent into the network, it may also concentrate the network traffic onto a few routes through the network, since DSR does not currently have a notion of load balancing.

All current on-demand routing protocols tend to discover a single path through the cloud of mobile nodes, which then becomes a de-facto backbone for the network. While this may be efficient in terms of reducing the number of active routes in the network and using short paths for all network traffic, it has the effect of concentrating all the network load on just a few machines and the wireless spectrum around them. This load concentration has two adverse effects: it disproportionally wears down the batteries of nodes along the backbone, since they carry more traffic, and it bottlenecks the aggregate bisection bandwidth of the network, since all traffic must be squeezed down the single backbone. Feeny has independently observed this phenomenon [12].

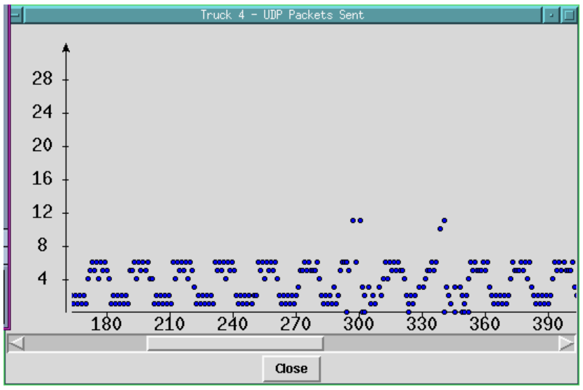
Adding resource usage awareness to the Route Discovery mechanism of DSR will allow the protocol to discover and use, when necessary, longer-than-optimal routes, but routes that balance the load across the network and enable the network to achieve higher aggregate bandwidth by routing around bottleneck points. Mobile nodes wishing to send data traffic can also use the resource usage information they learn. For example, nodes can pick routes through the network that best match the properties of traffic they wish to send. Some routes many have lower latency, while others may support higher bandwidth.

B.3. Providing Better than Best Effort Service

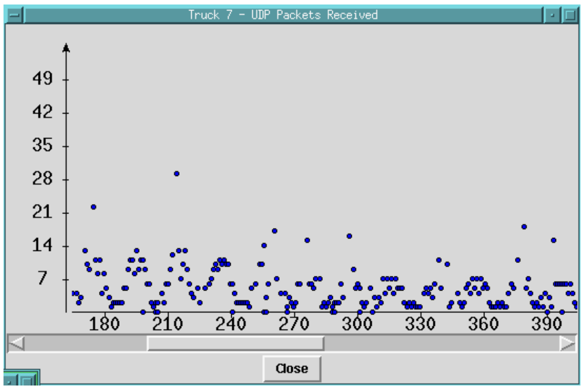
We have shown that the packet delivery rate achieved by DSR is quite high. However, the instantaneous throughput available to any particular flow varies dramatically over time. This results in severe jitter for evenly spaced streams of packets sent through the network as packet queues build and drain at the changing bottlenecks. Figure 4 demonstrates the extent of the problem in our ad hoc network testbed running with 7 nodes.

Figure 4(a) shows the rate at which node **T4** sent UDP packets, measured in terms of packets per second as a function of time. The recorded rate is the sum of two independent traffic streams: one stream at a rate of 1 to 2 packets per second sent to a central collection machine, and the second stream a synthetic audio connection with node **T7**. In order to model the "I talk, I pause" nature of human conversation, the synthetic audio connection sends packets at an average rate of 4 packets per second for 10 seconds, then is silent for 10 seconds, and then the cycle repeats.

Figure 4(b) shows the rate at which UDP packets were received at node **T7**. There is substantial jitter over the period from 180s–240s, but the "I talk, I pause" form of the connection is still visible. At times 240s and 276s, however, competing bulk data transfer TCP connections are begun, and the receiving pattern becomes unrecognizable.



(a) CBR packets sent per second



(b) CBR packets received per second

Figure 4 Effect of current DSR ad hoc network on a CBR stream of packets.

As in wired networks, we propose to alleviate this problem using admissions control and prearranged allocation of capacity, buffer space, and network interface queue priority. Providing such explicit control of the allocation of network resources is very similar to the proposed Integrated Services Quality of Service (QoS) systems [13, Chapter 7] in wired networks, such as RSVP [48], where the sender of a class of packets can negotiate with the network for the level of resources that will be spent on the class. The negotiation arranges the *QoS parameters* of a logical communications channel between the sender and destination of the packets. These QoS parameters typically include quantities such as:

- the capacity of the channel in bits/second
- the latency and jitter seen by packets traveling the channel, and
- the loss rate.

However, the *guaranteed reservation* schemes proposed for wired networks cannot be directly transfered to wireless ad hoc networks. Ad hoc networks simply cannot provide the notion of a reservation for a channel between two nodes with particular QoS parameters (e.g., capacity, latency, loss rate). Instead, each node in the network can provide only a *promise* to not deliberately oversubscribe itself such that maintaining a certain set of QoS parameters for the channel is impossible. The difference between a reservation and a promise is that a reservation is a positively stated guarantee that resources are available, while a promise is a negatively stated assurance that a node will not knowing prevent resources from being available. When a node discovers that conditions have changed such that it *is* oversubscribed, it provides *notification* to the senders to whom it must break its promises. The notification may contain the new set of currently available QoS parameters, or the sender may renegotiate them itself.

The reason ad hoc networks cannot possibly provide guaranteed reservations is because they are exactly that: ad hoc. By definition, ad hoc networks must be able to assemble out of independent nodes in arbitrary environments. For this reason, the nodes of an ad hoc network will be inherently subjected to “environmental effects” that are outside the network stack’s ability to predict or control. Since these environmental effects can dramatically change the communication capacity between any two nodes, no layer of the network stack can make any type of guarantee about the capacity available over a single link, let alone over a concatenation of several links. Example environmental effects that are potentially unpredictable and uncontrollable include:

- A node being physically moved away from its neighbors and creating partitions in the network. Even if each node’s network stack were loaded with a prediction of the node’s movement pattern, the node could still deviate from it.
- A node can be shut off or destroyed.
- A source of wireless interference could degrade the available capacity of the wireless channel around a node. This source could stem from natural sources, other human activity, or even the activity of other nodes which are

close enough to cause interference but not close enough to exchange data packets and participate in the network QoS algorithm.

Some of these effects are possible in conventional wired networks (routers do crash on occasion, and back-hoes do cut cables), but we must expect the frequency to be much, much greater in ad hoc networks. For example, where a 10-second long flow in a conventional network is very unlikely to experience one of these environmental effects, it may be an unusual 10-second period when one of these effects does not occur in an ad hoc network.

Because of unpredictable and uncontrollable environmental effects, the network layer in an hoc network cannot guarantee *any* amount of service to its upper layers. The most the nodes in the network can do is to exchange promises to maintain, if possible, a path through the network with certain QoS parameters, and to attempt to notify the sending node when the promise is broken. A promise is thus not a statement that a set of resources will be provided, *a promise is a statement by a collection of nodes that they will not intentionally oversubscribe themselves such that the promised resources cannot be provided.*

B.4. Pathologic Interactions Between Network and Transport Layers

The marked difference in characteristics between wired network environments and multi-hop ad hoc network environments can lead to poor behavior when protocols written for wired networks are run on top of ad hoc networks. For example, it is well known that TCP performs badly when run on top of networks with high packet loss or corruption rates, since TCP interprets the resulting packet loss as indicating network congestion and reduces its offered load to the network in response. We have identified at least four challenges that must be solved to enable TCP to achieve good performance over multi-hop ad hoc networks. The first three of these challenges have also been independently observed and reported by Holland and Vaidya [16], and by Liu and Singh [24].

- **Disambiguate packet loss signals from network congestion signals.** This will prevent TCP from reducing its offered load to the network when packets are lost due to wireless transmission errors or transient network routing errors.
- **Suspend packet origination when no route to the destination is available.** The problem of competitive retransmission between the link and transport layer is well known and debated. However, with on-demand routing protocols, a similar but more damaging problem exists between the network layer and the transport layer. If TCP originates packets even when the network layer has no route to the destination, these packets can be lost and cause TCP to exponentially increase the time until it next offers a packet [7]. The long spans between offered packets may cause TCP to completely miss short periods when connectivity does exist to the destination.
- **Reduce offered load during periods of network congestion.** Just like wired networks, ad hoc networks can suffer from congestion, and transport protocols must adapt when congestion exists. Congestion in wireless networks exhibits characteristics unlike those we have seen in wired networks, however. Congestion sets in particularly quickly, and not infrequently lasts for a long period of time (on the order of tens to hundreds of seconds).
- **Match characteristics of available network paths with behaviors desired by the transport layer.** As a windowed, flow-controlled protocol, TCP can achieve excellent throughput in the presence of high latency connections, but suffers greatly when even a few packets are lost. To optimize TCP throughput, a path with many shorter highly-reliable hops may be preferable to one with fewer, but error-prone, hops. Audio applications may have the opposite requirement.

For example, Figure 5 illustrates the last point particularly well, depicting a TCP connection between three stationary nodes in our ad hoc network testbed. The three nodes were positioned along a straight line, spaced maximum nominal transmission range apart, with one endpoint being the TCP source and the other endpoint being the TCP sink. Sequence numbers marked with a dot were transmitted using a two-hop route through the intermediate node, while sequence numbers marked with a '×' were transmitted directly between the endpoints.

In this example, when DSR performs Route Discovery, it often finds a one-hop route between the endpoints, due to the vagaries of the propagation environment. This one-hop route has a very high loss rate, however, and when TCP attempts to use it, so many packets are lost that TCP exponentially increases the time between offering packets to the network, resulting in the long packet gaps at times 12–15s, 18–23s, and 27–33s. Hallmarks of the first two challenges are also present in the figure, but they are obscured by the dominance of the fourth.

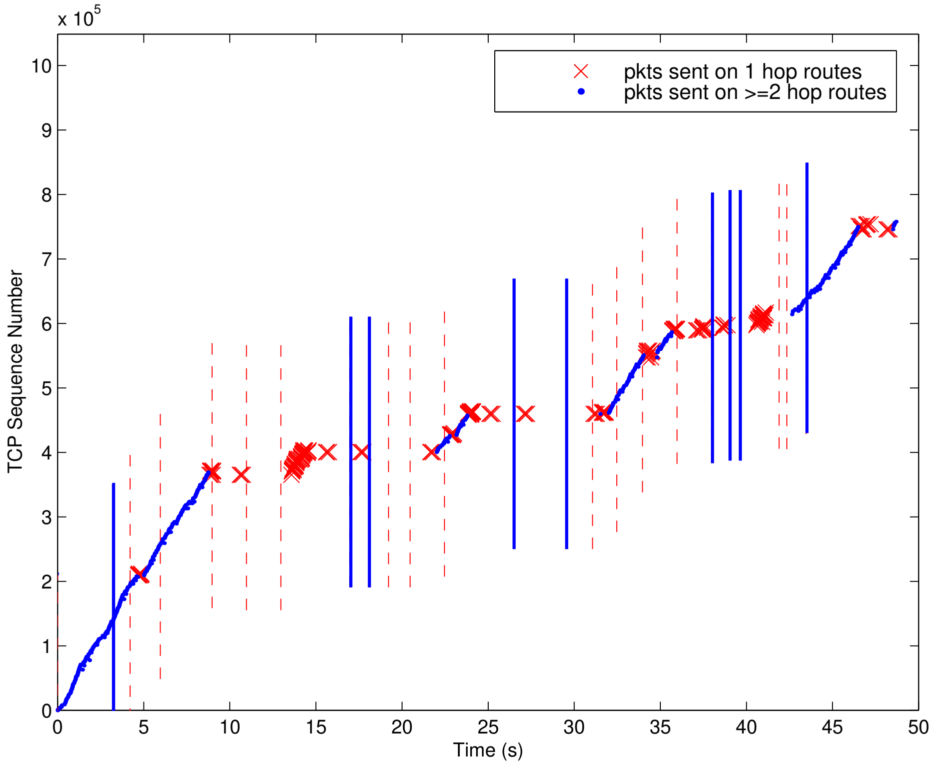


Figure 5 A TCP sequence number plot for a 1 MB transfer over a two-hop route. The vertical lines indicate the times at which the TCP source initiated Route Discovery; the dashed lines indicate the times at which only a non-propagating ROUTE REQUEST was transmitted and the solid lines indicate both a non-propagating and a propagating ROUTE REQUEST .

C. Outline of Path-State Maintenance Protocol

This appendix outlines a single protocol which attacks all of the challenges outlined in Appendix B, subject to the assumptions described Section 3. It first describes how path-state information can be created and used inside the network. It then describes how the characteristics of a path can be measured, and the measurements returned to the sender. These measurements enable the first approach for optimizing the use of resources throughout the network by providing senders with feedback on the current state of the network. It concludes by describing how senders can obtain resource promises from intermediate nodes, which supports the second form of resource management.

C.1. Path-State and Flow-State Identifiers

The metadata that intermediate nodes in the network must process can be divided into *path-state* and *flow-state*, where path-state is the fundamental unit of routing information and flow-state is the fundamental unit of Quality of Service information.

Path-state is associated with a particular set of hops through the network from some source **S** to a destination **D** (i.e., a particular source route in the network). It consists of the information needed to route packets along the path, and information about the carrying capacity of the path, such as the unused bandwidth along the path or the minimum latency of the path.

Flow-state is specific to a particular class of packets flowing between **S** and **D** that is routed over a given path. Flow-state is used to record Quality of Service promises that have been made for a particular flow, and allows packets from **S** to **D** that take the same path through the network to be treated differently by intermediate nodes. For example, all the TCP connections between **S** and **D** that take the same path will share the same path-state, but may have independent flow-state.

To represent paths and flows inside the network, we use a scheme inspired by the Virtual Path Index and Virtual Circuit Index of ATM networks [46, p. 451]. Paths are identified by the logical concatenation of the source node address and a 16-bit path identifier where the low 5 bits are 0. Flows are identified by a path identifier where the low 5 bits are used to distinguish between the different flows that use the same path. This scheme has two main advantages. First, each source node can independently generate globally unique path- and flow-identifiers. Second, the hierarchical relation of flow-identifiers to path-identifiers means that many flows from the same source node can share the same path-state, which reduces the overhead of maintaining the routing information. The drawback is that if a flow must be rerouted, its flow identifier will change. However, when a flow is rerouted the QoS metadata must be renegotiated anyway, so changing flow identifiers will not create needless additional work in the network.

C.2. Path-State Creation, Use, and Maintenance

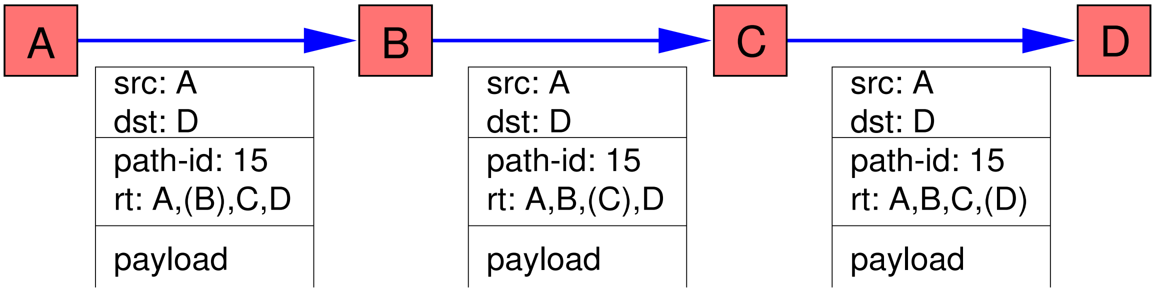
The path-state portion of the protocol has two major goals. The first goal is to ensure sufficient state exists at the nodes along a path from a source **S** to a destination **D** so that packets from **S** to **D** do not need to carry the complete source route. The second goal is to allow **S** to discover the characteristics of a particular path to **D** so that it can adapt its sending pattern to the capabilities of the path, or even choose a different path entirely.

The next sections describe how the path-state is created, how the characteristics of the path are discovered, and what metrics can be used to characterize the path.

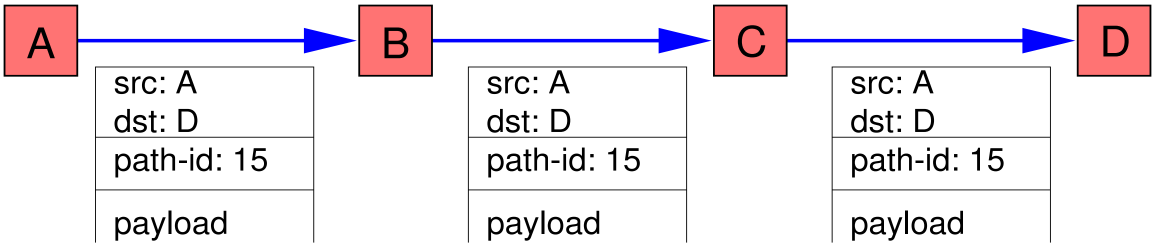
C.2.1. Creating Path-State for Routing

To create the path-state, we assume that Route Discovery proceeds as normal in DSR. Once the source node **S** has obtained a source route to the destination **D**, it begins sending data packets to **D** as normally done in DSR, with each packet carrying a full source route header. Internally, **S** assigns a path-identifier to that particular source route and stores the path-identifier in its route cache along with the source route. **S** then includes the path-identifier as part of the source route header as shown in Figure 6(a). As each intermediate node processes the source route to forward the packet, it also stores the source route in its route cache, indexed by the path-identifier.

Once **S** receives any kind of acknowledgment that one packet containing a full source route has made it through the network to **D**, it can begin sending subsequent packets to **D** without a full source route — carrying only the path-identifier as shown in Figure 6(b). Each intermediate node receiving such a packet queries its route cache to find the route the packet is supposed to take, and determines its next hop. The sender **S** could even optimistically assume that its first packet with a full source route will successfully create the needed path-state at each intermediate node, and



(a) Packet with path identifier and source route.



(b) Packet with path identifier only.

Figure 6 Path identifiers assigned to a source route by the originating node **A** enable later packets to omit the source route.

send all subsequent packets with just the path-identifier. As explained in Section C.4, **S** will receive a ROUTE ERROR if the state was not successfully created, and can then correct the situation.

C.2.2. Monitoring Characteristics of the Path

In order to support network layer services such as balancing the traffic load across the network, end-systems must have a method for determining the characteristics of the paths through the network that they could use. While many schemes have been proposed by which the end-systems themselves can measure the characteristics of a path (e.g., TCP congestion window and RTT calculations [1, 44, 47] and SPAND [40]), we hypothesize that, particularly in the in the dynamic environment of an ad hoc network, more useful, more accurate, and more timely information can be developed by enlisting the aid of the nodes along the path to measure the path characteristics.

We propose that each node can measure the activity around itself, and thereby determine information such as: the mean latency it adds to the packets it forwards and the latency variation (jitter); the number of additional packets per second it believes it can process; or the unused amount of wireless media capacity in the air around the node. Experimentation will be required to discover exactly which metrics will prove to be accurately measurable and useful, though Section C.2.3 provides several proposals. If the metrics kept by each node on a path are combined, the result should be a characterization of the path that the packet sender can use to organize or adapt its offered load.

To implement this scheme, we first define a new type of extension header for DSR than can be piggybacked onto a packet in the same way as the existing DSR headers. This new header is called the *path metrics header* (written as

`path/flow id`	`type`
`bitmap`	
`QoS variables`	

Figure 7 Conceptual layout of the MEASURE header.

MEASURE) (Figure 7) and conceptually consists of a bitmap to indicate what metrics are carried in it, the path-identifier of the path along which the metrics are measured, and the metrics themselves.

Whenever a sender **S** wishes to measure the characteristics of a path it is using, it includes the MEASURE header in any packet it sends along that path, setting the `type` of the header to **record**. As each node along the path forwards the packet, it updates the variables inside the MEASURE header with the metrics it has measured locally. When the header reaches the final destination **D**, **D** sets the `type` of the MEASURE header to **return** and piggybacks the header into any packet headed back to **S**. Since the path metrics header includes the path-identifier of the path along which it was measured, **S** can include the data into its route cache for future use, and can treat the receipt of the path metrics header as a positive acknowledgment that the path-state between **S** and **D** for the given path-identifier is correctly set up. This could lead **S** to cease including source routes in the packets it sends along the path, as described in Section C.2.1.

If we find that it is valuable to immediately provide **S** with the path metrics of every discovered route, we could alter Route Discovery slightly to generate this information. Currently, if an intermediate node has a cached route that it can use to answer a ROUTE REQUEST, it generates a ROUTE REPLY itself. Instead, we could require it to place its proposed route on the ROUTE REQUEST (turning it from a flood-fill broadcast into a unicast packet) and send the packet to the destination so it will measure the metrics of the complete path. The destination will then return the metrics to the source along with the ROUTE REPLY as described above.[1]

C.2.3. Candidate Metrics

In order to limit the additional overhead that collecting and distributing path-state metrics will place on the network, all the metrics must have property that the amount of space required to express the metric does not increase as the number of hops on the path increases. Experimentation will be required to determine which metrics are most accurately measured and most useful, but my initial set of candidates includes the following:

- Interface queue length — Our previous work [27] has shown that this is a good estimator of local congestion.
- Rate of interface queue draining — When an interface is backlogged, the rate at which packets leave the queue directly measures the usable capacity of that interface.
- Quiet time fraction — When an interface is not backlogged, the usable capacity of the interface can be estimated by promiscuously listening to the media and measuring the fraction of time during which it is not in use (though this will overestimate the capacity).
- Forwarding latency and variation — This can be measured as the time between when a packet is received and when it is acknowledged by the next hop.
- Unidirectional links — Paths containing unidirectional links are usable, but undesirable as they increase the overhead of Route Maintenance.
- Packet loss rate — Signal quality information from the interface itself, or the frequency of hop-by-hop retransmission, can be used to estimate the loss rate of each link.
- Likelihood of path breakage — Intermediate nodes may know ahead of time that they intend to shutdown or move such that paths through them will no longer work.

These metrics all have the property that they can be expressed in a single value that each node can measure locally. As a packet with a path metrics header passes through a node, the metrics in the header can be updated to reflect the node's metrics using a combination function like minimum, maximum, sum, or weighted average that produces another single value to replace the one already in the header. This updating will be done at the last possible moment before the packet is forwarded, in order to assure the packet has the most current metrics on it when it leaves.

[1] We have been intending to experiment with this alteration to Route Discovery for some time, since it offers two benefits, even without path-state metrics. It should decrease the number of broken routes returned by Route Discovery since each cached route is tested before being returned, and it should prevent DSR from jeopardizing one data packet for every bad route in an intermediate node's cache. The cost is some extra latency on Route Discovery.

C.3. Flow-State Creation, Use, and Maintenance

The flow-state portion of the protocol enables a sender to obtain promises from all nodes along a path to a destination that a certain set of resources are available along the path, and that the intermediate nodes are committed to making these resources available for the particular flow. This allows a sender obtain better-than-best-effort Quality of Service for a flow by obtaining promises from the intermediate nodes to reserve the resources needed to provide that QoS.

Unlike prior QoS work in wired networks, at this point we cannot formally characterize or bound exactly what type of services the flow-state protocol will be able to offer. The goal is to provide CBR and TCP streams with the ability to specify and obtain a minimum bandwidth and delay/jitter bound. If the environment is particularly harsh, it is possible that only best-effort service will be offerable. It is this intuition that lead us to create the system of promises and notifications. Only by experimentation can we determine how stable and effective this system will be in a multi-hop ad hoc network environment.

C.3.1. Requesting Promises along Existing Paths

Similar to the use of the path metrics header, at any time a promise can be requested or changed along any path an originator is currently using. Once an originating node has created a path-identifier for a route through the network, it can request a promise of network resources along that route by first generating a new flow-identifier to identify the promise. The originator then fills out a *flow-request header* (written as FLOW REQUEST) and inserts it into any packet sent along that path.

Figure 8 shows the conceptual layout of a FLOW REQUEST, which contains the new path-identifier assigned by the originator, the flow-identifier of the promise that this request supersedes (if any), the requested lifetime of the promise, and the QoS parameters that describe the requested promise itself. The use of the minimum and requested fields for the QoS parameters differs depending on whether the FLOW REQUEST is piggybacked on a ROUTE REQUEST or not, as described below.

When a FLOW REQUEST piggybacked on a unicast packet is received by a node, the node performs the following steps:

- If the node is the destination of the packet, it converts the FLOW REQUEST into a MEASURE with type **return** and uses the current values in the `desired` fields of the FLOW REQUEST to populate the fields of the MEASURE. It then piggybacks the MEASURE onto any packet being returned to the originator.
- Else if the intermediate node has available enough resources to meet the `minimum` requested promise in the FLOW REQUEST, it:
  - Sets aside the maximum of its available resources and the `desired` resources. The set aside resources are held in a *tentative promise* pool until the promise is confirmed, or a relatively short timeout expires.
  - Nodes can recycle resources from listed `old flow-id`
  - Updates the the `desired` fields of the FLOW REQUEST to reflect the resources set aside (there is questionable value in a down stream node allocating more resources to a flow than an upstream node can currently handle).
  - Forward the packet and piggybacked FLOW REQUEST to the next node on the path.

flow-id	old flow-id	
lifetime		
capacity	min	desired
latency	min	desired
variation	min	desired
loss	min	desired

Figure 8 Conceptual layout of the FLOW REQUEST header.

- Else, the node does not have enough resources to meet the `minimum` requested promise, so it sends the originator a ROUTE ERROR piggybacked with a MEASURE reflecting the minimum of the current values of the `desired` fields in the FLOW REQUEST and the available resources. The `type` field is set to **refused**. Such a MEASURE enables the originator to learn three things: that its requested cannot be satisfied along the given path; the identity of the bottleneck node; and the available resources up to and through the bottleneck node.

When the originating node receives a MEASURE header of type **return** for a flow on which it has an outstanding FLOW REQUEST, it accepts the promised level of service by changing the type of the MEASURE header to **confirm** and piggybacking the header on any packet going along the flow. This informs the intermediate nodes to move the set aside resources from the tentative promise pool to the allocated pool, and enables upstream nodes to free any set aside resources in excess of the capacity of a bottleneck downstream node.

The use of the `old flow-id` to recycle resources is important for two reasons. First, it enables an originator to attempt to increase or decrease the amount of a current promise without losing the resources it already has promised. Second, both packet loss and the expanding ring search of Route Discovery may result in several FLOW REQUESTS being sent for the same flow. If subsequent FLOW REQUESTS for a flow are not able to notify intermediate nodes that they can reuse resources set aside while processing earlier FLOW REQUESTS, the network will quickly reach a state where admissible flows are being needlessly rejected.

C.3.2. Requesting Promises as Part of Route Discovery

The scheme for requesting promises described in the previous section has the advantage that it enables an originator to request or update a promise for a flow along any route currently in its route cache, regardless of how it obtained the route. For the common case in which a node wishes to obtain a resource promise for a new flow to a previously unknown destination, we integrate the flow request with the Route Discovery for the destination.

Integrating the flow request with Route Discovery enables us to avoid the inefficiency of discovering routes that will not be usable by the flow due to insufficient resources. The integration of flow requests with Route Discovery also allows us to avoid a common pitfall of QoS schemes that layer a reservation signaling protocol on top of a unicast routing algorithm — schemes without tight integration will refuse admissible flows whenever the unicast routing algorithm directs the request packets into a congested area of the network, unless the signaling protocol also provides a method to backtrack the request and route around the congested area. Utilizing the same mechanisms currently used in Route Discovery, we can avoid the need for backtracking.

We call the combination of flow requests with Route Discovery *QoS-guided Route Discovery*, which originating nodes can invoke simply by piggybacking a FLOW REQUEST on the ROUTE REQUEST. Each node receiving the FLOW REQUEST uses the same algorithm described in Section C.3.1, with two exceptions:

- Nodes *silently discard* the ROUTE REQUEST if they can not meet minimum requirements
- Unless the ROUTE REQUEST indicates that replying from cache is forbidden, nodes with a cached route to destination unicast the ROUTE REQUEST along the cached route.

A node requiring a route with a QoS promise uses the following algorithm. First, it sends a ROUTE REQUEST that permits intermediate nodes to reply from cache. If the network is uncongested, this should frequently and quickly succeed in returning both a ROUTE REPLY and a MEASURE describing the available QoS along the discovered path. If after a timeout, the originating node has not received a ROUTE REPLY, it begins another Route Discovery, this time forbidding replies from cache, which will force an exploration of all feasible paths to the destination.

This scheme does risk an implosion of unicast REQUESTS at the target of the Route Discovery (e.g., if target is a popular server to which many nodes have cached routes). At the cost of additional complexity and soft-state, it would be possible to add hold-downs at the nodes surrounding the target so that only the first few REQUESTS are forwarded towards the target.

C.3.3. Providing Notifications of Changing Path Metrics

When a node detects that it must break a promise, it must notify the node to which it made that promise. Further research can study the issue of how the now reduced resources should be best distributed among the flows. Some published strategies exist, though simply picking the minimum set of promises to break that leave the other promises unchanged is probably sufficient.

The difficulty in providing notification of a changed path metric is getting this information back to the source. When promises are broken, it is not unreasonable for the breaking node to send its own ROUTE ERROR to the owner of the promise with a MEASURE informing the originator of the resources now available. The use of MEASURE headers to determine the currently available resources along a path is more problematic, however, as for every MEASURE sent by the originator, the destination must send a response containing the measured metrics.

If the traffic is TCP, the overhead of the responses are low, as they can be piggybacked on the ACK stream. For one-way CBR traffic though, introducing the overhead of a reverse stream to carry the changing metrics could be severe.

If the overhead of the responses becomes a problem, it may be possible to implement a enhanced piggyback mechanism, which we call *Optimistic Forwarding*. The approach is based on the fact that although no work has been exerted to create hop-by-hop routing information at each node, chances are good that each node can determine a next-hop for packets headed to any known destination by simply examining its route cache. By piggybacking the MEASURE header for one hop onto *any* packet that is headed to that next-hop, we can cheaply create a reverse flow of information that will eventually reach the originator of the MEASURE. Each node who receives a MEASURE with a type of **return** simply piggybacks the MEASURE for one-hop on packets that seem to be flowing the right direction back to the source. To insure the timeliness of the information, each MEASURE being returned to an originator could include a deadline by which the information is supposed to reach the originator. If it appears that hop-by-hop propagation will result in missing the deadline, the MEASURE can be unicast as a first-class packet to the originator.

C.4. Expiration of State from Intermediate Nodes

Since there is no guarantee that either the source or destination of a packet flow will be able to communicate with all of the nodes that carried the flow when they wish to terminate the flow, there must be time-based expiration mechanism by which intermediate nodes can purge the path-state and flow-state from their caches and reclaim the resources set aside to maintain it. However, if intermediate nodes were to purge the state of an active flow, the intermediate nodes would find themselves with packets to forward that do not contain a source route, but only contain a flow-identifier that references state they no longer hold. Since intermediate nodes do not necessarily know the timing with which the sender originates packets, an inactivity timer alone would have to be set very conservatively to prevent purging the path-state of low bit-rate connections.

To solve the expiration problem, we take advantage of the relatively "soft" nature of the path-state and flow-state. When the state is created, the source node specifies a time after which it should be discarded.[2] The source node can thereby estimate how often it must refresh the state, for example, by sending packets that contain a full source route on them. Should the state have somehow expired at an intermediate node when a packet labeled with a flow or path identifier arrives, the intermediate node returns a ROUTE ERROR to the source node specifying "missing state information" as the cause of the ERROR and elicit the sender to refresh the missing state.

Since all path-state information is guaranteed to have expired from the network after a bounded amount of time, nodes can safely and unambiguously reuse path and flow identifiers after that period.

[2]This time will typically be on the order of a hundred seconds