Timely, Reliable, and Cost-Effective Internet Transport Service using Structured Overlay Networks

by

Amy Babay

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

September, 2018

Abstract

Emerging applications such as remote manipulation and remote robotic surgery require communication that is both timely and reliable, but the Internet natively supports only communication that is either completely reliable with no timeliness guarantees (e.g. TCP) or timely with best-effort reliability (e.g. UDP). We present an overlay transport service that can provide highly reliable communication while meeting stringent timeliness guarantees (e.g. 130ms round-trip latency across the US) over the Internet. To enable routing schemes that can support the necessary timeliness and reliability, we introduce *dissemination graphs*, providing a unified framework for specifying routing schemes ranging from a single path, to multiple disjoint paths, to arbitrary graphs. We conduct an extensive analysis of real-world network data, finding that a routing approach using two disjoint paths performs well in most cases, and that cases where two disjoint paths do not perform well typically involve problems around a source or destination. Based on this analysis, we develop a timely dissemination-graph-based routing method that can add targeted redundancy in problematic areas of the network. We show that this approach covers nearly 99% of the performance gap between a traditional single-path approach and an optimal (but prohibitively expensive) scheme, while two dynamic disjoint paths cover about 70% of this gap, and two static disjoint paths cover about 40%. This performance improvement is obtained at an overall cost increase of less than 1% compared with using two disjoint paths. The implementation of the dissemination-graph-based Internet transport service is available in the open-source Spines overlay messaging framework (www.spines.org).

Advisor: Dr. Yair Amir
Readers: Dr. Yair Amir, Dr. Michael Dinitz, Dr. Cristina Nita-Rotaru, Dr. Xin Jin

Acknowledgments

First, I am deeply grateful to my advisor, Yair Amir for his advice, encouragement, optimism, and constant drive for excellence. Yair has been an outstanding mentor throughout my time in graduate school and has become a dear friend. He strives to improve the lives of everyone he touches, and has an impressive track record of success; his passion for helping students reach their full potential has certainly changed my life for the better. I cherish the years spent working with him pursuing solutions to exciting and important problems and growing as both a researcher and a person.

I thank Michael Dinitz, Cristina Nita-Rotaru, and Xin Jin for serving on my thesis committee, and Erica Schoenberger for serving on my GBO committee. Their support and feedback throughout this process has greatly improved the quality of the work and my skills as a researcher. In particular, I thank Michael Dinitz for advising me on the theoretical aspects of this work and on a qualifying project in the PhD program. Michael's guidance has broadened my perspective, strengthened my core CS foundation, and deepened my ability to think about problems in an algorithmic manner. I thank Cristina Nita-Rotaru for her support and contributions to my growth as a researcher throughout my time in graduate school and for always posing questions that force me to think about the work in new ways and improve it. I thank Xin Jin for contributing his expertise on software-defined networking and expanding my understanding of and vision for programmable networks. I thank Erica Schoenberger for contributing her unique perspective on the broader implications of the work.

I thank Emily Wagner for her critical role in making the work in this thesis a success. The Playback Network Simulator used to compare and evaluate dissemination-graph-based routing protocols in this thesis was originally Emily's Masters project, and her hard work and dedication to seeing our work succeed was crucial in turning the core contribution of the thesis into a high-quality conference publication. Beyond that, Emily is a wonderful friend and I admire the uncompromising quality of her work.

I thank Tom Tantillo for being an incredible colleague and friend. Collaborating closely with Tom was one of the highlights of my time in the PhD program. Tom is one of the most intelligent and technically capable people I know, and working with him on the code and algorithms behind Spire (our intrusion-tolerant SCADA

system for the power grid) was an experience I will treasure. Even when he can work faster than the rest of us, Tom never gets frustrated by people slowing him down with questions and is always willing to take the time to teach others and share his knowledge.

I thank Daniel Obenshain (Dano) for being an impressive role model when I first started in the DSN lab and for remaining a good friend throughout the years, even as he has moved on to his next stage in life. Dano is always willing to volunteer his own time to help others, and I cannot count the number of times I benefited from that generosity during my time in the DSN lab. Having Dano and Tom as role models and seeing the exciting work they were doing was a major factor in my decision to start a PhD. I thank them deeply for that initial inspiration as well as for help, stimulating technical collaborations, and friendship throughout the entire process.

I am immensely appreciative of LTN Global Communications for providing us with access to a global network infrastructure. This unique opportunity allowed us to not only collect the measurements that made the data-informed approach taken in this thesis possible, but also enabled us to deploy and test our protocols on a real globe-spanning network. I am grateful to the LTN Global Communications and Spread Concepts LLC teams for all of the effort that they put into making this possible. I especially thank Jonathan Stanton and John Lane for all of their technical help in setting up and maintaining our access to the system, as well as Nilo Rivera, Jacob Green, and John Schultz for their assistance and for insight into the practical operation of a commercial overlay. I am grateful to Malik Khan, Yousef Javadi, and Michal Miskin-Amir for their faith in and support for this work, opening up their commercial infrastructure for our experimentation, and sharing their perspectives.

I also thank John Schultz from Spread Concepts LLC for his mentorship before and during my time in the PhD program. John is one of the best engineers that I know, and the opportunity to work with him has helped me become a better programmer and learn how to write code that can have a life beyond research prototypes. As the architect of Spines, John was also instrumental in realizing the work in this thesis as part of the open-source release of Spines.

I thank my fellow members of the DSN lab throughout my time in graduate school for their collaboration and for making the lab an enjoyable place to be. I thank Marco Platania, Trevor Aron, and Samuel Beckley for their roles in making our work on intrusion-tolerant SCADA a reality. I thank Edmund (Ned) Duhaime for contributing to the structured overlay vision by exploring techniques for seamless overlay use, a key capability for extending the usefulness of structured overlays to a wide range of new Internet services. I thank Jeffrey DallaTezza, Kaiyue (Karin) Wu, and Henrik Schuh for being wonderful labmates.

In addition to my thesis research on timely and reliable communication, I also had the privilege of working on the important problem of protecting the power grid. I thank the team at Resurgo LLC, including Kevin Jordan, Dianne Jordan, Eamon Jordan, Kevin Ruddell, Ryan Ito, Kelli Goodin, Kara Knight, and Matt Troglia, for

pushing us to pursue this problem of high national importance and for helping to make that work more real and practical through a red-team experiment at Pacific Northwest National Laboratory and a test deployment at the Hawaiian Electric Company.

I thank my family for their constant love and support; without them none of this would be possible. I thank my parents, Kim and Jim Babay, for teaching me the value of working hard and never settling for less than my best. I thank them for always listening to my frustrations, celebrating my successes, and reminding me to take a step back, relax, and look at the bigger picture. They always know how to cheer me up and help me focus on what is most important. I thank my sister, Emily Babay, for being my first friend and role model. Emily has always been one of the first people I turn to when I want to share big news or when I need someone to lean on or provide advice. Her drive and accomplishments continue to inspire me to push myself in pursuit of my goals.

Finally, I thank my wonderful fiancé Chris Paxton for his endless love and support, for always being there for me (even when he couldn't be there physically), and for making my life richer. Chris has shared all of my challenges, frustrations, successes, and breakthroughs, both the tiny day-to-day ones and the big defining ones, throughout the past five years, and I could not ask for a better partner in all of this. I look forward to building the future together with him.

During my time at Hopkins, I received support from the Defense Advanced Research Projects Agency (DARPA) grant N660001-1-2-4014 to Johns Hopkins University, from the Department of Defense (DoD) Environmental Security Technology Certification Program (ESTCP) Project EW-201607 to Resurgo LLC, from the National Science Foundation (NSF) grant 1535887 to Johns Hopkins University, and from a Johns Hopkins University Department of Computer Science fellowship.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over the past five decades, the Internet has been dramatically successful, becoming an integral part of modern life. The Internet's ability to scale several orders of magnitude to reach its current level of ubiquity was made possible by its key design principle of keeping it simple in the middle and smart at the edge (or adhering to the *end-to-end argument* [2]). Under this model, the core of the network is kept as simple as possible, responsible only for best-effort packet switching, and more complex logic and services are implemented only at the network edge. The simplicity of the network core makes it extremely scalable, and the fact that all applications can be treated in the same manner by the network core has enabled an enormous range of applications to be deployed on the Internet.

Today, the Internet's ubiquity, global reach, and standardized nature are driving increasingly high-value services to use the Internet for their communication infrastructure, and new network applications that were not previously possible are emerging. These new applications bring new demands, requiring capabilities that the Internet does not natively support. For example, these applications may involve more complex communication patterns, such as multicast or anycast, require higher levels of security or system resilience, or have more demanding performance requirements. For these applications, the Internet paradigm presents a limitation, as the introduction of new network capabilities is restricted to the network edge.

One such class of emerging applications bringing new performance demands includes new highly interactive applications, such as remote robotic surgery, other remote manipulation tasks, or collaborative virtual reality. Natively, the Internet supports communication that is either completely reliable end-to-end, but with no timeliness guarantees (e.g. using TCP), or that is timely but with only best-effort reliability (e.g. using UDP). However, these emerging applications bring severe constraints on timeliness, while simultaneously requiring high reliability. For example, for remote manipulation, human perception requires feedback to be received within about 130ms to be perceived as natural. This 130ms includes both the time for the command to reach the destination and for the feedback to be returned, translating to

a latency requirement of 65ms each way. Supporting such applications on a continent-wide scale is highly demanding, as the network propagation delay constitutes a large fraction of the 65ms latency deadline. For example, the propagation delay across North America is about 35-40ms. This thesis develops an overlay transport service to support these types of demanding interactive applications over the Internet in a cost-effective manner.

In recent years, structured overlay network architectures have been developed to support applications that require both timeliness and reliability. These architectures use programmable overlay nodes in the middle of the network to enable custom overlay routing and hop-by-hop recovery protocols, rather than relying on the Internet's routing and end-to-end recovery. Overlay nodes are instantiated in general-purpose computers, providing unlimited programmability to implement custom protocols, and are strategically placed in data centers across the globe to provide sophisticated processing capabilities throughout the network, rather than only at end hosts. Applications using these structured overlay architectures include multimedia applications, such as VoIP [1] and live television [3]. A live TV service, supporting interviews from remote studios, requires a one-way latency bound of about 200ms with 99.999% of packets delivered on time. A global overlay network with 10-20 well-situated overlay nodes can support such a service by using the 160-165ms available after accounting for a 35-40ms propagation delay to allow some buffering and hop-by-hop recovery on overlay links. A service using this approach to support the TV and media industries is available today [4].

In contrast to applications that can tolerate a 200ms one-way latency, for the demanding applications targeted in this thesis, there is almost no flexibility to allow for recovery or buffering. Moreover, while techniques such as redundant sending along a single path or network coding can improve reliability, the combination of bursty loss on the Internet and the strict timeliness constraints of the target applications reduces their effectiveness. Thus, a different approach is needed.

To improve the probability that a packet is successfully transmitted to its destination without the need for repeated recovery attempts that may cause its delivery latency to exceed its deadline, we consider redundant dissemination schemes, where multiple copies of a packet are sent over more than a single path to provide multiple opportunities for it to reach its destination on time. For applications with strict timeliness and reliability requirements, flooding on the overlay topology can provide an optimally reliable solution. In this approach, each packet is sent on all possible paths, so it has the highest possible probability of reaching its destination on time. However, overlay flooding is very expensive. Since ISPs charge for each packet sent, the cost of a redundant dissemination scheme corresponds to the number times it requires each packet to be sent. Since overlay flooding sends each packet over every link in the overlay topology, it incurs an extremely high cost.

A less expensive approach is to send on multiple disjoint paths. For example, sending on two disjoint paths costs slightly more than twice the cost of the single best path

and allows a packet to reach its destination as long as it is successfully transmitted along one of the two paths. Most existing systems that send data redundantly over more than a single path to improve reliability use disjoint paths (e.g. [5], [6]).

Disjoint paths offer a coarse-grained trade-off between cost and reliability, as adding paths provides higher reliability at a higher cost. However, this approach uniformly invests resources along the paths from a source to a destination. Investing fewer resources in more reliable parts of the network and more resources in less reliable parts of the network can improve the trade-off between cost and reliability. By considering the dynamic loss and latency characteristics of network links, we aim to provide close to optimal reliability at a reasonable cost.

1.1 Solution Highlights

We present a new approach that transports packets in a timely, reliable, and cost-effective manner by constructing *dissemination graphs* based on network characteristics, application latency and reliability requirements, and cost. A dissemination graph is a connected subgraph of the overlay network topology that connects a source and destination. In our approach, each packet from a source to a destination is sent over all the links in the dissemination graph for that flow.

Ideally, we would calculate the cheapest dissemination graph that meets the application's reliability and latency constraints in order to provide a cost-effective service. However, the problem of finding such a dissemination graph is NP-hard. While we can make computing optimal dissemination graphs tractable for certain topologies, the calculation is too slow to effectively adapt to changing network conditions, even for practical topologies with a relatively small number of nodes.

Therefore, our approach is to analyze real-world network data, examine the types of problems that occur in the field, and develop methods to construct and deploy dissemination graphs that can provide the necessary reliability and timeliness during such problems. A key finding of this analysis is that a routing approach using two disjoint paths performs well in most cases, and that cases where two disjoint paths do not perform well typically involve problems around a source or destination. The grounding in real-world data and focus on applications with extremely strict timeliness and reliability requirements separates our approach from the few previous works that have considered redundant dissemination schemes beyond disjoint paths to improve performance in overlay routing (e.g. [7]).

Based on the types of problems we observe in the collected data, we develop a timely dissemination-graph-based routing method that combines the use of two disjoint paths with a limited number of precomputed dissemination graphs that address the most common types of problems we observed and switches between the different graph types as network conditions change. Specifically, the approach uses two dynamically computed disjoint paths under normal conditions, and switches to use

precomputed dissemination graphs that add targeted redundancy around a source or destination when problems are detected in that region. For network conditions involving multiple types of problems simultaneously, we use more robust graphs that add targeted redundancy around the source and destination while also including two disjoint paths. We show that this approach can cover nearly 99% of the performance gap between a traditional single-path approach and an optimal (but prohibitively expensive) scheme, compared with about 70% for two dynamic disjoint paths or about 40% for two static disjoint paths. This performance improvement is obtained at a cost increase of only 0.3% over two disjoint paths.

The primary contributions of this work are:

1. The invention of dissemination graphs, providing a unified framework for specifying routing schemes ranging from a single path, to multiple disjoint paths, to arbitrary graphs.

2. An extensive analysis of real-world network data, finding that a routing approach using two disjoint paths performs well in most cases, and that cases where two disjoint paths do not perform well typically involve problems around a source or destination.

3. A dissemination-graph-based routing approach that employs targeted redundancy based on current network conditions to provide close to the optimal reliability possible under strict timeliness constraints at a reasonable cost.

4. An evaluation via simulation of the dissemination-graph-based routing approach, showing that it can capture nearly 99% of the benefit of an optimal dissemination method at a cost similar to that of using two disjoint paths.

5. An open-source implementation of a complete dissemination-graph-based transport service over the Internet, using the Spines overlay messaging framework [8].

6. A practical evaluation of the implemented transport service that validates the simulation results in a local-area environment with emulated latency and loss, as well as an evaluation on a global overlay network.

1.2 Thesis Organization

The remainder of this chapter (Section 1.3) reviews related work. Chapter 2 presents the fundamental components of the structured overlay framework underlying our transport service, including its resilient network architecture, unlimited programmability, and flow-based processing. Chapter 3 describes several approaches to constructing disseminations graphs that build on and extend existing overlay routing schemes and discusses the trade-offs they present between cost, reliability (under strict

time constraints), and complexity. To develop a timely, reliable and cost-effective service, we analyze how the foundational dissemination-graph-construction approaches we consider would perform using network data collected in a real wide-area setting; this analysis is presented in Chapter 4. Chapter 5 presents the dissemination-graph-based transport service developed based on this analysis, and evaluates the service via simulation, using the data collected in the wide-area analysis. Chapter 6 describes the implementation of the transport service in the Spines overlay messaging framework, evaluates that implementation, and validates the simulation described in Chapter 5. Chapter 7 discusses two practical use cases for the transport service developed in the thesis and uses the collected wide-area network data to assess the transport service's ability to support these use cases. Chapter 8 concludes the thesis.

1.3 Related Work

The dissemination-graph-based transport service introduced in this thesis is built on top of a structured overlay framework and leverages prior work on overlay routing and recovery protocols, as well as redundant dissemination protocols in overlays and other network domains. It makes use of an existing family of overlay recovery protocols, but introduces dissemination graphs as a new approach to routing that is more flexible than existing methods and allows for more sophisticated protocols that can provide better performance. Our approach of constructing dissemination graphs based on reliability goals and latency constraints is related to work on the theory of reliable network design, although due to the hardness results from this domain and practical limitations, we ultimately develop an approach that provides excellent practical results but does not provide formal cost or reliability guarantees.

1.3.1 Overlay Routing and Recovery

Our dissemination-graph-based transport service builds on existing work on overlay networks. Many previous works have observed inefficiencies in Internet routing and employed overlays to make use of alternative paths with better performance characteristics. For example, the Detour framework uses an overlay to experiment with alternative routing protocols based on performance metrics [9], RON recovers from problems on a direct Internet path by sending packets through an intermediate node (selected based on loss or latency measurements) [10], and the analysis of one-hop source routing in [11] shows that many Internet path failures can be overcome using the simple approach of sending through a single randomly selected intermediate node when a problem is detected. However, these approaches were not designed to meet strict latency deadlines. In contrast, the work in [1] presents an overlay routing protocol specifically designed to support real-time communication for VoIP. That work

introduces an *expected latency* metric that considers both the loss and latency characteristics of overlay links, with the goal of selecting a path with the highest probability of delivering packets within a specific timeliness requirement.

In addition to bypassing problems on a given Internet path via overlay routing, overlays have also been used to improve reliability and latency by enabling fast recovery of lost messages over relatively short overlay hops. Fully reliable hop-by-hop protocols (e.g. [12]) can improve latency compared with end-to-end protocols but cannot support timeliness guarantees. OverQoS [13] combines at most one recovery attempt per lost packet with forward error correction (FEC) to provide a statistical bound on the loss rate experienced by an overlay link. We use a family of recovery protocols specifically designed to support applications with strict latency requirements [1,3,14], as described in Section 2.4.

While these existing works offer considerable improvements over native Internet performance, they are not sufficient for our target applications. Because of our applications' high reliability requirements and low tolerance for interruptions, rerouting on a single path after problems are detected cannot provide the required level of timeliness and reliability, even when combined with recovery protocols or FEC, as the applications' strict timeliness requirements reduce the number of successful recoveries that can be performed and the effectiveness of FEC.

1.3.2 Multipath Routing and Redundant Dissemination

Existing work has shown the benefits of redundant dissemination over multiple edge- or node-disjoint paths in the context of overlay networks (e.g. [15–18]) and wireless networks (e.g. [5,19]). Redundant dissemination is used to improve performance (e.g. [15,16,18,19]), as well as to improve security or resilience to attacks (e.g. [5,17]).

In the wireless domain, SMT [5] transmits erasure coded messages over node-disjoint paths to overcome malicious nodes. Paths are selected at the source from a given set of available routes, with the source rating paths based on received acknowledgments to discard paths deemed non-operational and automatically select a subset of paths and coding redundancy factor based on the probability that each path is operational (though how to calculate such probabilities is left as future work). The work in [19] considers the use of multipath routing to improve performance in mobile ad hoc networks and presents a path set selection algorithm called Disjoint Pathset Selection Protocol (DPSP) that aims to generate a large number of highly reliable disjoint paths. This work shows that the path sets generated by DPSP have a significantly longer lifetime (time until all paths become non-functional) than a single path or alternative approaches, but it does not attempt to propose a specific routing method for utilizing the generated paths, as it focuses on the path set selection problem.

More relevant to our work, in the overlay domain, the work in [16] presents an empirical evaluation comparing single-path routing in the style of RON (where the source can choose to send on the direct Internet path or on a single-hop overlay path through one intermediate relay), redundant dissemination on two disjoint single-hop paths, and redundant sending on the direct Internet path. The work demonstrates that both reactive overlay-level rerouting and redundant dissemination have benefits, and a combination of the two, using redundant dissemination over two dynamically selected single-hop paths (with one path minimizing loss and the other minimizing latency), offers further performance improvements. JITeR [18] shares our goal of supporting applications with strict timeliness constraints and aims to improve reliability by making use of multiple paths in a way that explicitly incorporates delivery deadlines. In JITeR, each message is sent over a base channel and may be simultaneously sent over one or more backup channels. Similarly to [16], each channel may be either a direct Internet path (with multihoming potentially allowing several such distinct paths) or a single-hop path using one relay node. Base and backup channels are selected to minimize correlation between them, and to deliver packets "just-in-time", using longer paths that are still within the deadline first, so that shorter paths may be used later for retransmissions or for packets with shorter deadlines. In [15], redundant dissemination is used to improve performance and failure resilience for reliable multicast; in particular, this work creates a mesh that ensures that each client has at least n node-disjoint paths to the root nodes (and may have other non-disjoint paths as well), allowing them to withstand the failure of any $n-1$ nodes. The work in [17] uses node-disjoint paths to support intrusion-tolerant communication, using k node-disjoint paths to ensure timely communication in the presence of up to $k-1$ malicious overlay nodes.

In Section 5.4, we show that in the context of performance, disjoint paths provide a substantial improvement in reliability compared to a single path, but more sophisticated dissemination graphs can provide considerably better performance for a similar cost. Moreover, the approach of using dissemination graphs composed of two node-disjoint paths that we evaluate in Section 5.4 represents an improvement over previous work using disjoint paths, as it leverages the structured overlay framework to make use of paths with multiple intermediate nodes between the source and destination (providing greater control over path selection and enabling effective hop-by-hop recovery over long distances) and selects these paths using a metric that considers loss, latency, and recovery possibilities (though that metric was not our invention; it was used for single-path selection in [14]). In contrast, existing work on multipath overlay routing is typically limited to the use of a single intermediate relay node (e.g. [6, 16, 18]), and does not provide the ability to select multi-hop paths based on performance characteristics.

While most existing works using redundant dissemination consider only disjoint paths, [7] proposes routing over non-disjoint paths in order to satisfy application reliability constraints in the presence of geographically correlated failures, while min-

imizing cost and considering latency constraints. The approach of [7] ranks paths based on their latencies and then determines the cheapest path-set that meets the specified reliability requirement, according to a failure model that incorporates geographic correlation. Path computation is performed by a centralized controller that is responsible for initiating the process of switching to a new path set. While our goals are similar, the extremely demanding latency and reliability requirements of our target applications require a different approach that can react quickly to changing network conditions. The path-set computation of [7] employs several heuristics to reduce running time in practice, but still computes an optimal path-set, which can be highly computationally intensive when many paths need to be considered. Because we aim to provide close to optimal reliability, it is likely that many paths will need to be considered, making the computation too expensive for timely reactions. Moreover, our approach of computing dissemination graphs at the source and including the graph to be used in each packet allows for faster, more flexible rerouting (each packet can potentially use a different graph) compared with the controller-coordinated path setup and reconstruction process in [7]. Redundant dissemination beyond disjoint paths is also used in [17], which employs overlay flooding for extreme resilience; however, this approach is only cost-effective for a small subset of critical monitoring and control traffic and is too expensive for data transmission in our applications.

Other work combines the use of multiple paths with forward error correction (FEC) or multiple description coding (MDC). For example, [20] sends video encoded via multiple state encoding over multiple paths, and PDF [6] uses FEC while sending packets over both the direct Internet path and a maximally link-disjoint backup path. SplitStream [21] distributes content over multiple multicast trees that do not share interior nodes to improve load balancing as well as resilience, and suggests combining this approach with MDC or FEC to further improve reliability. While such schemes could be used with dissemination graphs, we choose to use fully redundant dissemination to avoid the need for application-specific encoding (as in MDC), and to avoid introducing additional latency for redundant encoded packets (as in FEC), as this may be significant given our strict timeliness constraints.

1.3.3 Theory of Reliable Network Design

Without considering latency constraints or recovery, the problem of calculating the reliability of communication between a source and destination over a given dissemination graph can be formulated as the classical two-terminal reliability problem. This problem and the related all-terminal network reliability problem have been extensively studied and shown to be #P-hard (e.g. [22–26]), meaning it is likely infeasible to determine the exact reliability of arbitrary dissemination graphs. This hardness also implies that constructing dissemination graphs that meet a given reliability constraint while minimizing cost is NP-hard (see Section 3.3 for details). Moreover,

only a few works on calculating reliability consider any form of latency constraints (e.g. [27], which considers a hop-count constraint), and none of the theoretical models we are aware of incorporate recovery protocols, which can have a significant impact on reliability.

Because of the hardness of calculating reliability, prior work on designing reliable networks or graphs has used heuristics or other approximate approaches (e.g. [28–31]). However, because they do not consider latency constraints or recovery, these approaches are not directly applicable to our problem. We take a different approach of examining real network data to design a practical solution that can address most common problems.

Chapter 2

Structured Overlay Framework for Timely, Reliable Transport

The foundation of the timely and reliable Internet transport service developed in this thesis is a structured overlay framework that enables new Internet services by building a resilient network architecture and bringing unlimited programmability into the middle of the network to support services that the Internet cannot natively provide.

This thesis specifically focuses on wide-area network applications that have strict timeliness requirements, while also requiring high reliability. For such applications, packets must be delivered within the strict timeliness constraint: packets that arrive after their deadline are not useful. Within the timeliness constraint, the transport service must be as reliable as possible. Some minimal packet loss is acceptable, but the goal is to deliver the highest possible percentage of packets on time. We specifically target on-time delivery rates on the order of 99.999%. These types of applications are not well supported by the native Internet, which provides either timely delivery with no reliability guarantees (i.e. UDP) or fully reliable delivery with no timeliness guarantees (i.e. TCP).

A structured overlay framework can be used to enable these types of applications, and other emerging applications with demanding requirements, to operate in a cost-effective manner over existing Internet services. Structured overlay networks create logical networks that run on top of the Internet and support powerful capabilities through three key principles: a resilient network architecture, software overlay routers with unlimited programmability, and flow-based processing. We describe each of these three principles below and discuss how they support the timely, reliable transport service.

2.1 Resilient Network Architecture

The physical architecture of the structured overlay network supports its key capabilities of unlimited programmability, global state maintenance, fast reaction, and resilience. The foundation of the overlay's physical architecture is a resilient network architecture, illustrated in Figure 2.1.

The structured overlay network consists of overlay nodes connected to each other via overlay links. Overlay nodes are physically instantiated as general-purpose computers residing in data centers, while the overlay links correspond to Internet paths between the overlay nodes. The use of general-purpose computers provides unlimited programmability, which makes it possible to support a wide range of current and future applications with highly demanding requirements.

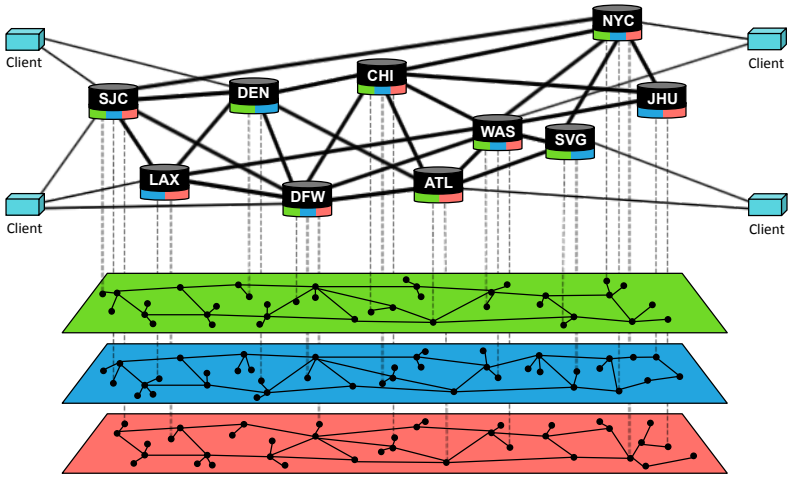


Figure 2.1: Resilient Network Architecture

A key property of structured overlay networks is that they require only a few tens of well situated overlay nodes to provide excellent global coverage. This is because, in general, placing overlay nodes about 10ms apart on the Internet enables the desired timeliness and reliability qualities, and about 150ms is sufficient to reach nearly any point on the globe from any other point. The limited number of nodes allows each overlay node to maintain global state regarding the condition of all other overlay nodes in the network and the connections between them, allowing fast reactions to changes in conditions. In contrast to the 40 seconds to minutes that BGP may take to converge during some network faults, a well designed overlay can detect and route around problems at a subsecond scale.

In addition to fast problem detection, effective rerouting also requires the ability to make use of multiple alternative routes. To support this, overlay networks exploit redundancy in the resilient network architecture. As shown in Figure 2.1, in such an architecture, each overlay node is connected to each other node through multiple

redundant paths at the overlay level, and is connected to multiple underlying ISP backbones. This redundant architecture allows the overlay to change the underlying network path used for data transmission without relying on rerouting at the Internet level. This is accomplished by selecting a different overlay-level path or by choosing a different combination of ISPs to use for a given overlay link.

For overlay-level rerouting to be effective, disjointness in the overlay paths should reflect physical disjointness in the underlying networks: if different overlay paths overlap in the underlying network, a single problem in the underlying network can affect multiple overlay paths. To exploit physical disjointness available in the underlying networks, the overlay node locations and connections are selected strategically. Overlay nodes are placed in well-provisioned data centers, as ISPs invest in such locations by laying independent fiber connections between them. The overlay topology can then be designed in accordance with the underlying network topology, based on available ISP backbone maps. Overlay links are designed to be short (on the order of 10ms) so that the Internet routing between overlay neighbors (i.e. overlay nodes connected by a direct overlay link) is relatively predictable. Short overlay links also enable improved performance and services by breaking the end-to-end principle at the overlay level and increasing the processing possibilities in the middle of the network.

Connecting each overlay node on multiple ISPs provides additional redundancy and resilience. Multihoming in this way allows the overlay to route around problems affecting a single provider and allows most traffic to avoid BGP routing by switching between providers at the overlay nodes and traversing only *on-net* links (i.e. overlay links that use the same provider at both endpoints) between them, which generally results in better performance (although any combination of the available providers may be used, if desired).

2.2 Overlay Routers with Unlimited Programmability

Because overlay nodes are physically instantiated in general purpose computers, they offer unlimited programmability in the middle of the network (as opposed to only at end hosts). The overlay software runs on each overlay node as a normal user-level program and is able to leverage general-purpose computing resources to provide sophisticated network services. For example, the overlay software can make use of the physical computer's ample memory to store sent messages for later retransmissions or to track received messages to allow de-duplication of retransmitted or redundantly transmitted messages. Similarly, the arbitrary processing possible in a general-purpose computer allows for customized network protocols and even more advanced features like cryptographic processing.

The relatively limited number of overlay nodes needed to provide global coverage

enables the key feature of state sharing among the overlay nodes. This shared state can then be used to implement sophisticated network protocols. For example, overlay nodes share information about their connections to neighboring overlay nodes, such as current loss and latency characteristics, to enable fast rerouting according to custom routing protocols (including the dissemination graphs introduced in this thesis) in response to changes in network conditions. Moreover, structured overlays can support multicast and anycast capabilities that are generally not available on the Internet by sharing multicast and anycast group state among the nodes. All of the overlay nodes share information about whether they have clients interested in a particular group, making it possible to disseminate multicast messages to all relevant nodes or to select the best target for a given anycast message (as anycast messages are delivered to exactly one member of the relevant group). The overlay's unlimited programmability allows using higher-level information to select the best anycast target based on the application's desired properties.

The flexibility of a software overlay router allows many different network protocols (e.g. routing and recovery protocols) to coexist. Each client application that connects to the overlay can select the combination of protocols that best supports their particular demands, and new protocols can be easily added to support new applications. A single overlay node can serve many clients (with the clients potentially using different combinations of protocols), and multiple overlays can even be run in parallel to provide scalability (with each overlay potentially using a different variant of the overlay software). The overlay software interface looks like a normal application to the underlying network and like a powerful network (with additional services) to the applications that use it.

2.3 Flow-Based Processing

In contrast to the Internet's stateless packet switching, the structured overlay framework employs flow-based processing. From a client's perspective, a flow consists of a source, destination, and the overlay services selected for that flow. A client can select different overlay services (e.g. routing and recovery protocols) for each application data flow.

The overlay node's access to ample memory and processing resources allows it to maintain the flow-based state needed to support basic services like reliability, as well as more advanced services like authentication. Within the overlay, application data flows may be aggregated based on their source and destination overlay nodes or the services they select, with state maintenance and processing performed on the aggregate flows.

2.4 Hop-by-Hop Recovery

A key capability enabled by the structured overlay framework is hop-by-hop recovery. In contrast to the end-to-end recovery used to provide reliability on the Internet, which requires packets to be recovered between their source and destination hosts, hop-by-hop recovery allows packets to be recovered on the specific short overlay hop on which they were lost. This greatly reduces the time needed to recover a lost packet. Specifically, since recovering a packet requires at least two propagation delays across the distance over which it is being recovered (one to request the lost packet and one to retransmit it), limiting the recovery to an overlay hop with a propagation delay on the order of 10ms enables lost packets to be recovered while still meeting strict latency constraints. For example, across North America with a propagation delay of about 40ms, the recovery time can be reduced from about 80-100ms to about 20-25ms.

Hop-by-hop recovery is made possible by the combination of the resilient network architecture, unlimited programmability, and flow-based processing discussed above. The resilient network architecture ensures that overlay nodes are strategically placed so that the propagation delay across each link is short enough to support fast recovery. The unlimited programmability and general-purpose computing resources allow intermediate overlay nodes to store packets for recovery, maintain the necessary flow-based state, and implement the recovery protocol.

While our target applications' strict timeliness requirements limit the number of recoveries that can be performed successfully for a given packet, at least one recovery on one overlay link is generally possible. Since the propagation delay of an overlay link is typically on the order of 10ms, it is feasible to use about 20-25ms to recover a lost packet on an overlay link, while still meeting a 65ms delivery deadline on the scale of a continent with 35-40ms end-to-end propagation delay.

We consider a family of recovery protocols based on the real-time recovery protocol of [1] and a later generalization [3]. These protocols are designed to operate within timeliness constraints and therefore are not 100% reliable: intermediate nodes can discard packets once their delivery deadline has passed, since recovery will not be useful after that point. The basic real-time recovery protocol of [1] allows a given packet to be requested and retransmitted at most once per overlay link. More generally, instead of issuing a single request or retransmission, multiple copies of the request and retransmission can be scheduled to be sent, separated by a short delay [3]. This short delay improves the probability of successfully bypassing the window of correlation for loss, reducing the likelihood that all copies of a request or retransmission will be lost. The transport service developed in this thesis uses the basic protocol, allowing one request and one retransmission, as extremely strict latency constraints reduce the ability to effectively space out the requests and retransmissions.

Chapter 3

Dissemination-Graph-Based Routing

The structured overlay framework described in Chapter 2 provides a foundation for a timely and reliable Internet transport service, but supporting such a service also requires effective protocols to be deployed within that framework. This chapter introduces *dissemination-graph-based routing*, a flexible source-based routing approach that enables packets to be disseminated over arbitrary subgraphs of the overlay topology. We then present several specific dissemination-graph-based routing protocols and discuss the range of trade-offs they offer in terms of their abilities to leverage the resilient network architecture's path redundancy to support stringent timeliness and reliability requirements and to provide a cost-effective service.

A *dissemination graph* is a connected subgraph of the overlay topology that connects a flow's source to its destination. The dissemination graph may consist of a single path from source to destination, multiple disjoint paths, or a more complex graph composed of an arbitrary set of overlay links. In our dissemination-graph-based routing approaches, each packet is stamped with the graph that should be used to disseminate it by its source at the time it is sent and then forwarded over the overlay links included in that dissemination graph.

This flexible source-based approach provides a unified routing framework that makes it simple to specify arbitrary graphs. Each source specifies exactly which links each of its packets should be sent over, and intermediate nodes simply forward each packet on all of their outgoing links that are included in the dissemination graph for that packet; intermediate nodes do not need any additional logic to handle different types of graphs.

During forwarding, duplicate packets are suppressed: each node only forwards the first copy it receives of a given packet. Thus, each packet normally traverses each edge of the dissemination graph exactly once; however when network problems occur, loss may prevent a packet from reaching some links in its dissemination graph or a recovery protocol may cause it to be retransmitted on some links. Note that

this deduplication distinguishes dissemination graphs from sets of independent but potentially overlapping paths, where packets may be transmitted multiple times over links shared by multiple paths.

The use of source-based routing eliminates the possibility of packets being dropped due to inconsistent network views across the overlay nodes as routing re-stabilizes after a change is detected: each node honors the dissemination graph stamped on the packet, so it follows exactly the path decided at the time it is sent, even if network conditions change while it is in flight. However, because the decision is made at the source, this approach can increase the time required to respond to certain network problems (compared with non-source-based protocols), as information about the problem must propagate to the source before routing can be updated. While non-source-based protocols may allow faster reaction times, they are much less flexible: link-state single-path routing approaches are widely used, but for disjoint paths or more complex dissemination graphs, it is not obvious how to implement an effective non-source-based routing scheme that maintains the desired graph properties in all cases (e.g. disjointness, or redundancy level).

3.1 Model

To understand what types of dissemination-graph-based routing approaches will be most effective in supporting timely, reliable, and cost-effective delivery, we first specify the model in which they are implemented and how they are evaluated.

3.1.1 Network Model

Our dissemination-graph-based routing is implemented at the overlay level. As discussed in Chapter 2, each overlay node is connected to several other overlay nodes via overlay links. Overlay links are logical links that use the Internet for their underlying communication medium. Each overlay link may have multiple ISP options available, but for simplicity we only consider there to be one logical link between each pair of neighboring overlay nodes.[1] The overlay topology (i.e. set of overlay nodes and links) is known to all of the overlay nodes, and is assumed not to change on the timescales we consider.

Our experience shows that a relatively small number of nodes (i.e. tens) are sufficient to provide excellent global coverage. Since the number of neighbors of each node (its degree) is also typically small in practice (i.e. less than ten), the number of links in the topology is not too large. This makes it practical to specify the complete set of overlay links that make up the dissemination graph to be used to transport

[1] In practice, each pair of overlay neighbors decides on the underlying ISP to use on the link between them, and this is abstracted to the rest of the network as a single logical link.

a packet in that packet's header (see Chapter 6 for implementation details). The limited number of nodes also makes it feasible for each node to maintain global state regarding the status (i.e. current loss rates and latencies) of all the overlay links in the system. Note that in practice link status information takes time to be updated and propagate through the network (on the order of tens to hundreds of milliseconds).

3.1.2 Cost Model

A key design goal of our timely reliable transport service is that it be cost-effective. While instantiating a structured overlay network incurs certain fixed costs (e.g. obtaining space in strategically located data centers, purchasing computers and networking equipment), those costs come from the framework itself and do not depend on the particular overlay protocols that are used. Therefore, in evaluating different dissemination-graph-based routing approaches we focus on the component of the cost that varies with the approach used: the total cost of the bandwidth that approach uses to transport a packet through the network.

In our model, the service provider instantiating the structured overlay network makes sufficient access bandwidth available at each overlay node to support the demands of the applications they serve. Because provisioning sufficient bandwidth is not difficult to do in practice, limiting the amount of data sent on each link to minimize congestion and best make use of limited bandwidth is not a concern. The key constraint is the cost of the bandwidth used. In normal ISP pricing models, the ISP customer (overlay service provider) purchases a certain amount of access bandwidth for each node (e.g. 10 Gbps or more). The overlay operator commits to using a certain percentage of that bandwidth (e.g. they will pay for at least 10-20% of the bandwidth regardless of actual usage). If they use more than the commitment (which is the normal case), they pay based on the total amount of bandwidth that was actually used. Therefore, our model considers the total amount of bandwidth used per packet as its cost metric. Note that while bandwidth costs for different links may vary based on the ISP used and the location, these variations are relatively small, so we assume that the cost of bandwidth is the same across all overlay links in the topology.

Analytical Measurement. Each time a packet is sent on an overlay link, it counts against the total amount of bandwidth being used. Therefore, we consider the cost C_G of a given dissemination graph G to be the number of edges (overlay links) $E(G)$ included in that dissemination graph, since this corresponds to the number of times each packet transported using that dissemination graph will need to be sent in the normal case (although loss or retransmissions may affect this somewhat in practice).

The overall cost of a dissemination-graph-based routing approach depends on the cost of the dissemination graphs that it uses and how often each dissemination graph is used. For static routing approaches, where the dissemination graph used

for a particular flow does not change based on network conditions, the cost of the routing approach is simply the cost of the single dissemination graph it uses. For dynamic routing approaches, where a different dissemination graph can potentially be selected for each packet in a flow, this overall cost can be computed as a weighted average of the cost of the dissemination graphs used, where each graph is weighted by the percent of the time it is used. However, because the amount of time that a particular graph is used in a dynamic routing approach depends on the particular network conditions experienced, we do not calculate exact costs analytically for these approaches (although we will qualitatively discuss expected relative costs); we instead measure the cost in practice by counting the number of packets actually sent.

Practical Measurement. When evaluating the cost of a dissemination-graph-based routing approach in a real or simulated deployment, we measure cost using two different metrics. The first is simply the average *dollar cost* of selecting that approach for a packet. As explained above, we abstract the dollar cost of bandwidth as the number of packets that must be sent. Therefore, the dollar cost metric is calculated as the as the total number of packets sent (including all redundant copies of a given packet transmitted by any overlay node) divided by the total number of packets introduced into the network by some source. While this metric is simple and accurately represents the cost that would be paid using a particular routing approach at a particular time, a potentially undesirable aspect is that it rewards unreliable graphs that drop packets early in transmission. That is, approaches using graphs that include many edges but result in packets being dropped before they have a chance to traverse those edges will look less expensive than approaches using graphs with the same number of edges in which packets successfully traverse all edges (but it is true that the less reliable graphs would in fact result in a lower cost in terms of real dollars spent).

Therefore, we also report a *goodput cost* metric that measures the average dollar cost of *successfully* transporting a packet from its source to destination using a given routing approach. This metric is calculated as the total number of packets sent (including all redundant copies of a given packet transmitted by any overlay node) divided by the total number of packets successfully delivered at their destination within their time constraint.[2]

Considered together, these two metrics provide an understanding of both the absolute cost of a dissemination-graph-based routing approach and the cost of the approach relative to the level of reliability it provides.

3.1.3 Reliability Model

The reliability of a dissemination graph is defined as the probability that a packet sent using that dissemination graph successfully reaches its destination. Reliability

[2]This is the metric used in the cost analysis we present in [32].

depends both on the structure of the dissemination graph that is used and on the network conditions experienced at the time a packet is sent.

Analytical Measurement. We consider a dissemination graph $G = (V, E)$, where vertices V are overlay nodes and edges E are overlay links. We have an assignment of failure probabilities (loss rates) $p : E \to [0, 1]$ to edges and an assignment of lengths (latencies) $l : E \to \mathbb{R}^+$ to edges. In terms of reliability, a dissemination graph over a network that experiences packet loss can be modeled as a random graph in which each edge $e \in E(G)$ in the underlying graph is removed with probability $p(e)$, where the failure probability $p(e)$ corresponds to the loss rate of edge e. Essentially, we view each packet as traversing a (potentially different) post-failure graph that includes all edges on which the packet can be successfully transmitted and excludes all edges on which it is lost. Then, the probability that a packet sent over the dissemination graph successfully reaches its destination within its latency constraint (i.e. the reliability of the graph) is exactly the probability that the source and destination are connected by a path whose length is within the latency constraint in the post-failure graph. Therefore, for a given source s, destination t, and latency constraint L, we define the reliability of the dissemination graph G subject to the latency constraint L as follows:

Definition 1. *Let $G_{\{p\}}$ be the probability distribution over subgraphs obtained by removing each edge $e \in E(G)$ independently with probability $p(e)$. The **reliability** of G is then the probability that there exists a path from s to t of length at most L in a subgraph H drawn from the probability distribution $G_{\{p\}}$. We denote this probability by $Rel(G, s, t, L)$.*

While we can precisely define the reliability of a dissemination graph for a given source, destination, and set of network conditions, in many cases this value is not easy to compute. In fact, without considering the latency constraint (or equivalently when L is arbitrarily large), computing the reliability of a graph as defined above is exactly the classical *two-terminal reliability* problem [23–26], which is known to be #P-hard [22], implying that there is no polynomial-time algorithm for computing reliability in the general case (there may exist a fully polynomial-time randomized approximation scheme that can approximate the answer arbitrarily well, but this is an open question; no such scheme is known today). Note that when discussing two-terminal reliability, we will use a non-latency-constrained definition of reliability $Rel(G, s, t)$, which is equivalent to Definition 1 but only requires that s and t be connected, rather than connected within distance at most L.

Of course, for certain restricted classes of graphs, computing reliability is simple. For example, for a single path, the reliability is simply the probability that the packet can be successfully transmitted across every link in the path, or $\prod_{e \in E(G)}(1 - p(e))$. However, for many of the arbitrary dissemination graphs we consider the computation may not be feasible.

Practical Measurement. In practical deployments or simulations, the reliability of a dissemination graph is simply measured as the percentage of packets that are delivered at their destination within the specified timeliness constraint. Therefore, to determine the empirical reliability, we just count the total number of packets delivered on time and divide that by the total number of packets introduced into the network.

3.2 Foundational Approaches to Dissemination Graph Construction

Building on the novel idea of dissemination-graph-based routing and our structured overlay framework for deploying such routing schemes over the Internet, we investigate several foundational approaches to constructing the dissemination graphs to be used in routing. These approaches range from a single path, to disjoint paths, to arbitrary graphs, and can all be specified using our framework. Together, these approaches present a range of trade-offs between reliability, cost, simplicity, and feasibility.

3.2.1 Dynamic Single Path

In a dynamic single path approach, each packet is sent on the shortest path from its source to its destination, as determined by its source at the time the packet is sent. While a variety of different metrics could be used in determining the shortest path, we consider a link-weight metric based on the *expected latency* metric of [1], which takes into account both the loss rate and the latency on each overlay link, with the goal of selecting the path that is most likely to reach the destination within the time constraint. Specifically, we calculate the expected latency of a link as:

$$(1-p)\cdot T + (p-2p^2+p^3)\cdot(3T+\Delta) + (2p^2-p^3)\cdot T_{max} \tag{3.1}$$

Here, p is the current loss rate of the link, T is the current latency of the link, $3T+\Delta$ is the time needed to recover a lost packet (a constant Δ to detect the loss, plus three propagation delays for the original send, request, and retransmission), and T_{max} is the maximum allowed latency, used as a penalty for packets that are lost and cannot be successfully recovered by the deadline.

This is the cheapest approach considered: the cost is just the number of overlay links in the single best path (note that the best path in terms of expected latency may not have the fewest number of overlay links, and therefore may not be the path with the lowest cost). If there is any problem on the selected path that cannot be masked by the recovery protocol being used, losses will be visible to the application at least until a new path is selected. While subsecond rerouting is possible, the time needed

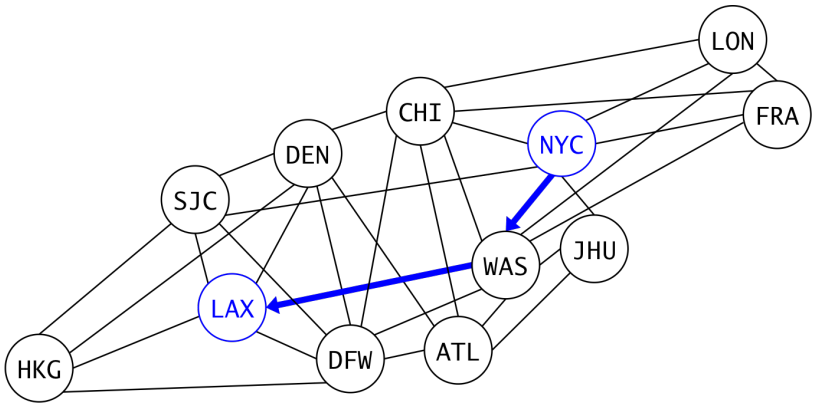


Figure 3.1: Single-best-path dissemination graph (in terms of expected latency, assuming normal-case latency and no loss) from New York to Los Angeles. Cost: 2 edges.

to react to problems can still result in interruptions of 100-200ms, which are not acceptable for the most demanding applications. An example dissemination graph for a flow from New, York to Los Angeles in a global overlay topology is shown in Figure 3.1.

3.2.2 Static Two Node-Disjoint Paths

To avoid disruptions due to problems that occur on a single path, multiple paths may be used simultaneously. When two static node-disjoint paths are used, each packet is sent over a dissemination graph consisting of two node-disjoint paths, where the paths are chosen at startup time based on their normal-case latencies.[3] The paths are not recomputed when loss rates or link latencies change, making this approach very simple to implement, as no link monitoring or rerouting is required. This approach masks the failure of any one path, at approximately twice the cost of using the single best path. However, because the paths are static, if both of the selected paths experience a problem, the application will be affected until the problem resolves. Therefore, the effectiveness of the approach depends on the type of network problems experienced; because it will continue to use its two initially selected paths even if both paths fail, it can perform even worse than a single dynamic path in cases where the two selected paths experience problems but some other better-performing path exists.

[3]Ideally, we would like to minimize the latency of the longer of the two paths, but there is not a known efficient method to compute such paths. Instead, we minimize the sum of the latencies of the two paths, which can be done efficiently, for example using Suurballe's algorithm [33].

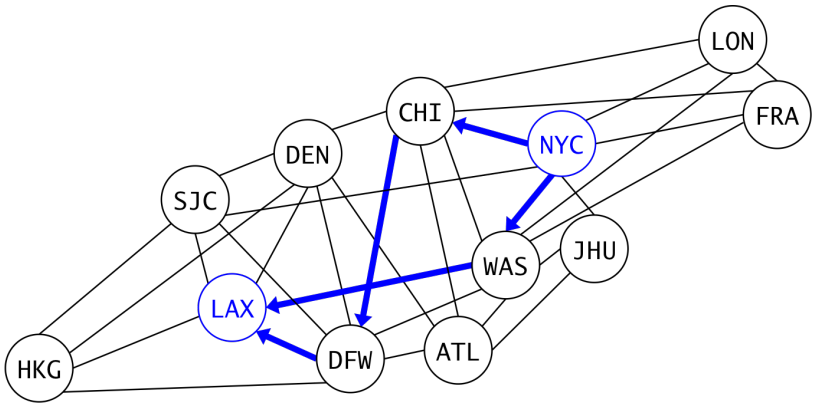


Figure 3.2: Two node-disjoint paths (chosen based on expected latency, assuming normal-case latency and no loss) from New York to Los Angeles. Cost: 5 edges.

3.2.3 Dynamic Two Node-Disjoint Paths

When two dynamic node-disjoint paths are used, each packet is sent over a dissemination graph consisting of two node-disjoint paths chosen based on their expected latencies (considering both loss and latency, according to Equation 3.1), as calculated at the packet's source at the time it is sent.

Like static two node-disjoint paths, this approach costs about twice as much as using the single best path, but it fixes the potential problem of continuing to use two failed or problematic paths when alternative good paths are available. The application will only experience a problem if both paths are affected before rerouting occurs, or if no unaffected path exists.

3.2.4 Overlay Flooding

When overlay flooding is used, each packet is sent over every link in the overlay topology. Overlay flooding is extremely expensive, but provides optimal timeliness and reliability: if there is any path that can transport a packet from its source to its destination within its deadline, it will be delivered on time.

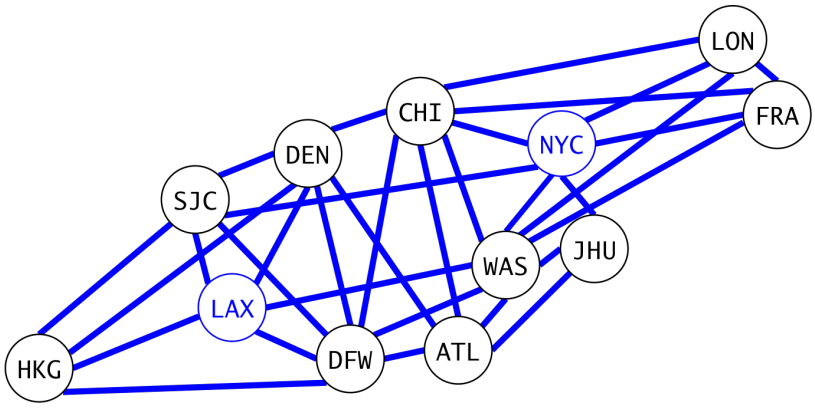


Figure 3.3: Overlay flooding from New York to Los Angeles. Cost: 64 (directed) edges (direction arrows on edges are omitted for clarity).

3.2.5 Time-Constrained Flooding

Time-constrained flooding is a novel approach that preserves the optimality of flooding at a lower cost. In time-constrained flooding, a packet is sent on every overlay link that can improve the probability that it reaches its destination on time, providing optimal reliability. Time-constrained flooding improves on the cost of overlay flooding by not sending packets to nodes from which they cannot reach their destination within the time allowed.

The time-constrained flooding dissemination graph between a source and destination for a given latency constraint is constructed as follows:

1. Run Dijkstra's algorithm from the source to mark each node with its distance (in terms of network latency) from the source.

2. Reverse all edges and run Dijkstra's algorithm from the destination to mark each node with its distance from the destination.

3. Iterate over each edge in the graph: let d_s be the distance from the source to the head vertex of the edge, d_t be the distance from the tail vertex of the edge to the destination, and l be the latency of the edge. If $d_s + d_t + l \leq$ latency constraint, include the edge.

4. Remove unnecessary loops by checking whether each included node is on at least one path from the source to the destination that does not include any cycles and removing any nodes (and their adjacent edges) that are not on such a path. The loop removal algorithm is presented in Algorithm 1.

5. Remove edges entering the source or leaving the destination (since the source will not forward the packet again after its original introduction, and there is no need to forward the packet further once it reaches its destination).

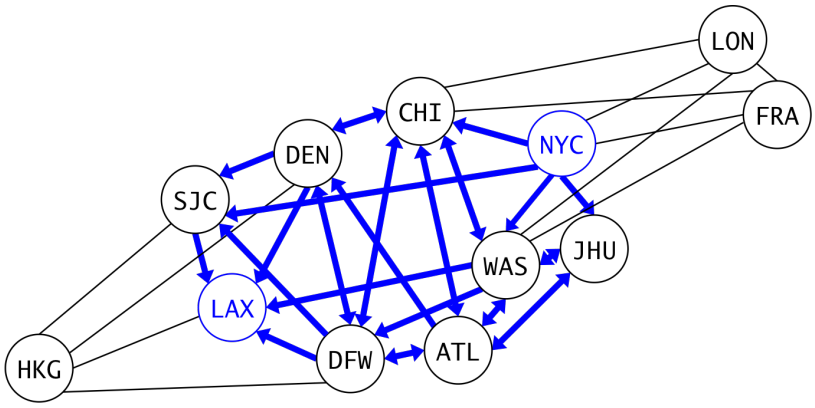


Figure 3.4: Time-constrained flooding from New York to Los Angeles, using a 65ms latency constraint. Cost: 31 edges.

The time-constrained flooding dissemination graph for a given source and destination can be computed once based on the normal-case latencies of the overlay links and does not need to be recomputed as network conditions change. While latency variations in the network may render some edges in the normal-case time-constrained flooding graph ineffective at certain times, excluding these edges would provide only minimal cost savings, so we do not consider it worth the additional complication of recomputing the graph based on changing conditions. Time-constrained flooding is optimal in terms of reliability, but it does not consider the cost of the dissemination graph beyond removing edges that do not improve reliability and therefore is still likely to be too expensive for practical use.

3.3 Optimal Dissemination Graphs

The ideal approach would be to send each packet using the cheapest dissemination graph that meets its application's reliability and latency constraints, based on current network conditions (or alternatively, the most reliable dissemination graph that can be constructed within a given budget).

From the perspective of a service provider who wants to minimize the cost of providing an agreed upon level of service, we can specify the problem of choosing the cheapest dissemination graph satisfying a given reliability constraint as follows:

Problem 1. Minimum-Cost Dissemination Graph. *Given an input graph $G = (V, E)$, an assignment of failure probabilities $p : E \to [0, 1]$ to edges, a source s, a destination t, latency constraint L, and a reliability requirement $r \in [0, 1]$, our goal is to find a subgraph H such that $Rel(H, s, t, L) \geq r$ and the number of edges $|E(H)|$ in the subgraph is minimized.*

Algorithm 1 Time-Constrained Flooding Loop Removal

Given an input graph $G = (V, E)$, source s, destination t, latency constraint L, and assignment of latencies to edges $l : E \to \mathbb{R}^+$
Let $G' = (V', E')$, where $V' \leftarrow V \cup d$ and $E' \leftarrow E \cup \{(s, d), (t, d)\}$, where d is a dummy node added to the graph and $l(s, d) = l(t, d) = 0$
Let $G_u = (V_u, E_u)$ be an undirected version of G' (collapse any edge pairs $(u, v), (v, u) \in E'$ to a single undirected edge (u, v)) ▷ This requires that all latencies are symmetric
Let *list_eval* $\leftarrow V_u \setminus \{s, t, d\}$
while $|$*list_eval*$| > 0$ **do**
    Choose some $n \in$ *list_eval*
    Let p_1, p_2 be two disjoint paths of minimal total latency from n to d in G_u (e.g. using Suurballe's algorithm)
    if $\nexists p_1, p_2$ or $l(p_1) + l(p_2) > L$ **then**
        Let $V_u \leftarrow V_u \setminus \{n\}$
        Let $E_u \leftarrow E_u \setminus \{(n, v)\} \quad \forall v : (n, v) \in E_u$
        list_eval $\leftarrow$ *list_eval* $\setminus \{n\}$
    else
        list_eval $\leftarrow$ *list_eval* $\setminus V_u(p_1) \setminus V_u(p_2)$
    end if
end while
Let $G'' \leftarrow G_u$ with directedness restored. For each edge $(u, v) \in E_u$, $(u, v) \in E'' \iff (u, v) \in E$, and $(v, u) \in E'' \iff (v, u) \in E$
return G''

Unfortunately, as discussed above, without considering latency constraints (which only make the problem harder), calculating the reliability $Rel(G, s, t, L)$ of an arbitrary graph is a #P-hard problem (it is exactly the two-terminal reliability problem). Formally, the two-terminal reliability problem is defined as follows:

Problem 2. 2-Terminal Reliability. *Given an input graph $G = (V, E)$, an assignment of failure probabilities $p : E \to [0, 1]$ to edges, a source s, and a destination t, compute the reliability $Rel(G, s, t)$.*

We can define the associated decision problem as follows:

Problem 3. 2-Terminal Reliability (Decision). *Given an input graph $G = (V, E)$, an assignment of failure probabilities $p : E \to [0, 1]$ to edges, a source s, a destination t, and a reliability requirement r, determine whether the reliability $Rel(G, s, t) \geq r$.*

The #P-hardness of 2-Terminal Reliability immediately implies that 2-Terminal Reliability (Decision) is NP-hard. Based on the hardness of the two-terminal reliability decision problem, we can show that solving the Minimum-Cost Dissemination Graph problem to calculate optimal dissemination graphs is NP-hard to even approximate:

Theorem 1. *Unless $\boldsymbol{P = NP}$, there is no polynomial-time α-approximation for* Minimum-Cost Dissemination Graph *for any α.*

Proof. We prove this by a gap reduction from 2-Terminal Reliability (Decision). The reduction is trivial: given an instance (G, p, s, t, r) of 2-Terminal Reliability (Decision), we simply reinterpret it as an instance of Minimum-Cost Dissemination Graph with arbitrarily large L (specifically, it is sufficient to set $L \geq \sum_{e \in E(G)} l(e)$ to ensure that the latency constraint will not exclude any simple path in the graph). Let ALG be an α-approximation for Minimum-Cost Dissemination Graph. If $rel(G, s, t, L) < r$ then by definition ALG must return INFEASIBLE. On the other hand, if $rel(G, s, t, L) \geq r$, then ALG will return a subgraph H where $rel(H, s, t) \geq r$ (and where $|E(H)|$ is at most α times the optimum, but the size bound is not important here). Hence, ALG lets us decide in polynomial time whether $rel(G, s, t, L) < r$ or $rel(G, s, t, L) \geq r$, and so since 2-Terminal Reliability (Decision) is NP-hard, ALG can only exist if $\mathbf{P} = \mathbf{NP}$. □

While we can make it feasible to find optimal dissemination graphs for certain practical overlay topologies using exhaustive search (using optimizations to limit the search space as we describe in [34]), such computations are too slow to permit fast reactions to network problems, taking on the order of tens of seconds to complete for overlay topologies of the size we consider.

Moreover, because detecting and reacting to problems takes time, we find that even the optimal dissemination graph for the current network conditions known at

the source is not necessarily the best choice. The formulation above assumes that the source has perfect knowledge of the network conditions (latencies and loss rates) each packet will experience. In practice, however, it takes time to detect that a loss rate or link latency has changed and to propagate that information back to the source. Therefore, it is often useful to employ additional redundancy beyond what is required to cope with current known problems to mitigate loss during the time required to react to new problems.

Since a routing scheme based on computing optimal dissemination graphs is not practical (due to the fact that it is not computationally feasible in all cases and due to the lack of perfect knowledge of network conditions), we instead take a different approach of learning about the types of network problems that occur in the field, with the goal of developing data-informed approaches to effectively cope with these types of problems. We discuss the process of collecting and analyzing real wide-area network data and our findings in Chapter 4.

Chapter 4

Analyzing Network Problems in the Field

Each foundational approach discussed in Chapter 3 presents a different set of trade-offs between reliability, cost, simplicity, and feasibility. To determine how these trade-offs interact in practice, we collect and analyze real network data to learn about the types of network problems that occur in the field and how each approach performs during such problems.

4.1 Flow Modeling with the Playback Overlay Network Simulator

To analyze the performance that different overlay routing approaches would achieve on a real global network, we developed the Playback Network Simulator. Playback is an overlay simulation tool that collects data on each link of a real overlay topology and then uses that per-link data to model the performance of one or more flows through the network. Playback can simulate a flow from any source to any destination using any dissemination-graph-based overlay routing protocol and any recovery protocol in the family we consider (i.e. recoveries in the style of [1, 3]). A complete description of the Playback Network Simulator is available in [35], but we provide an overview here that also reflects some recent changes to Playback's modeling approach.

Data Collection. To determine the network's loss and latency characteristics, Playback's data collection component collects fine-grained data by sending messages on each overlay link in the topology at short intervals. The sending frequency for these messages is configurable, but for our wide-area analysis, we send a message on each link every 10ms. The granularity may be further improved (e.g. to every 1ms) with an increased bandwidth allowance and improved logging infrastructure: the ability to store logs as we recorded data over a long period of time and processes'

ability to keep up with logging data from multiple neighbors were limiting factors in our data collection.

During data collection, each node logs every message it receives, including sequence numbers and timestamps that allow the simulator to calculate loss and latency. Loss rates and round-trip latencies can be calculated directly. To determine approximate one-way latencies, we assume that during periods with no network problems, the latency is symmetric between each pair of nodes (i.e. each one-way latency is equal to half the round-trip time). We then use these one-way latencies determined during stable periods to calculate clock skew and appropriately adjust the one-way latencies during problematic periods. This approach is more accurate than simply using half the round-trip latency, as we find that network problems that result in increased latency on an overlay link often occur in only one direction.

Flow Simulation. For a given time period, source-destination pair, sending rate, overlay routing protocol, and recovery protocol, the Playback Network Simulator simulates the end-to-end performance of that flow based on the network conditions at the specified time. For each simulated packet in the flow, it calculates whether it would have reached the destination using the given protocols, and if so, what its latency would have been. The simulated packet is propagated across the network according to its dissemination graph. For each link the packet traverses, the simulator calculates the latency and loss rate of that link by averaging over the collected data for that link in a sliding *modeling window* centered at the time that packet reaches the link. Based on the loss rate, the simulator randomizes to determine whether the packet is successfully transmitted. If the first attempt to transmit the packet across a link is unsuccessful, the simulator performs further randomizations to determine when that loss is detected, whether a request for retransmission is successful, and whether a retransmission of the packet is received, based on the specific recovery protocol used. If the packet is successfully transmitted, the latency to traverse the link is calculated as the average one-way link latency at that time, plus any time needed to perform recovery.

When static routing approaches are used, the same dissemination graph is used for each simulated packet. However, for dynamic approaches, the dissemination graphs can change over time. Dynamic routing is done in a preprocessing step of the simulation, which processes the raw log files to determine when routing updates would have been issued. In modeling dynamic reroutes, we assume that a routing update is issued as soon as the average latency or loss rate measured over a sliding *update detection window* changes by a certain threshold percentage. The size of the update detection window affects how quickly we can respond to changes in the network: short update detection windows allow for fast rerouting when problems occur, but may cause instability by rerouting in response to small changes in the network (our experience shows windows on the order of a few hundred milliseconds to be practical). Once an update is generated, we assume that it takes 65ms to propagate to the source. This is a conservative estimate, since the maximum latency between two nodes in the North

American portion of the overlay is about 50ms, and in many cases the delay will be considerably shorter. In addition, to increase routing stability, once the loss rate or latency on a link is raised, we do not allow it to be lowered again for 30 seconds to avoid repeatedly attempting to use a link that may still be experiencing intermittent problems. An increase in loss or latency will always be reported as soon as it is detected. The main phase of the simulation takes the list of reroutes as an input and uses it to change dissemination graphs at the appropriate times.[1]

Note that all modeling parameters, including the modeling window, the update detection window, and the delay for updates to propagate to the source, can be changed, and the same data can be reanalyzed with different parameters. The only parameter that cannot be changed after data collection is the collection interval itself (which was 10ms for our wide-area analysis). In our evaluation, we use a modeling window of 100ms (which corresponds to 10 packets in our data) and a update detection window of 500ms, with a 2% loss update threshold and 10% latency-change threshold.

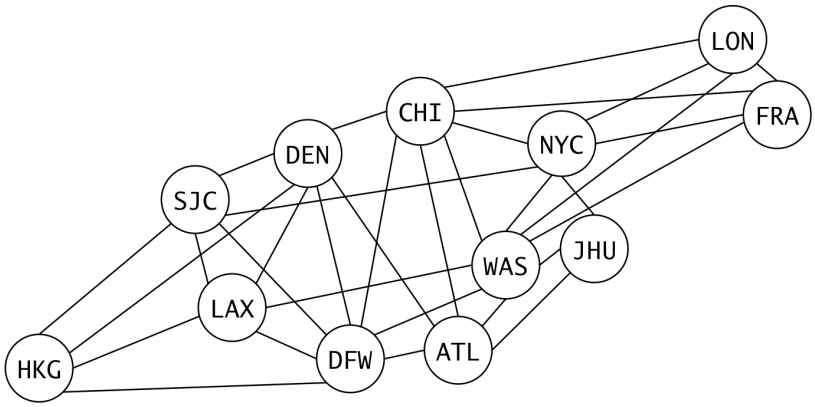


Figure 4.1: Global overlay topology spanning East Asia, North America, and Europe. Each circle represents an overlay node located in a data center.

4.2 Data Collection Environment

We collected data over a period of several months on a real global overlay network that we have access to through LTN Global Communications [4]. The specific overlay topology on which we collect data is based on the one used by LTN and uses the same underlying infrastructure. This topology includes overlay nodes in twelve data centers and spans East Asia, North America, and Europe, as shown in Figure 4.1. The data centers are located in Hong Kong (HKG), San Jose, California (SJC), Los Angeles, California (LAX), Denver, Colorado (DEN), Dallas, Texas (DFW), Chicago,

[1]Note that the process for modeling dynamic reroutes has changed from the version of the Playback Simulator used in [32, 35] to better match the behavior of a real implementation.

Illinois (CHI), Atlanta, Georgia (ATL), Washington, DC (WAS), New York, New York (NYC), Baltimore, Maryland (JHU), London, UK (LON), and Frankfurt, Germany (FRA).

While our overlay topology is based on LTN's, the topology we measure is somewhat less resilient than the full LTN topology, as we only collected data using a single ISP at each node at any given time, while the LTN topology has multiple ISP options available simultaneously. Analyzing the performance impact of multihoming and different ISP selection protocols is an interesting avenue for future work.

Over a period of four months, we collected four full weeks of data, using the Playback Network Simulator's data collection component.

4.3 Network Fault Pattern Analysis

We evaluated several of the foundational dissemination-graph construction approaches described in Chapter 3 to determine how they would have performed during our data collection period and which types of problems they successfully address. The approaches we considered were: dynamic single path, static two node-disjoint paths, dynamic two node-disjoint paths, and time-constrained flooding. Each was evaluated using no recovery protocol and using the real-time recovery protocol of [1]. The performance of each approach was evaluated for sixteen flows across North America. These flows include all transcontinental source-destination combinations of four cities on the East coast of the US (NYC, JHU, WAS, ATL) and two cities on the West coast of the US (SJC, LAX). A full analysis of the results appears in Section 5.4; here we only provide the intuition leading to our new method.

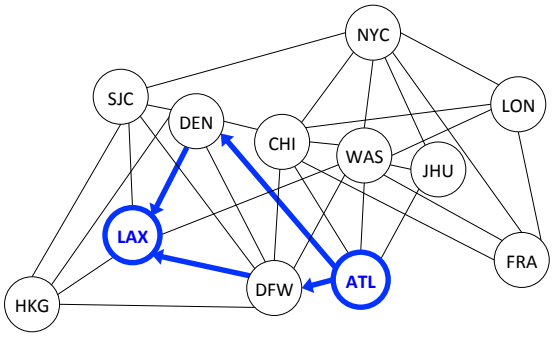
Overall, we find that two dynamic node-disjoint paths perform quite well, covering about 70% of the performance gap between a single-path approach and the optimal reliability of time-constrained flooding. Examining the data, we observed that most instances in which two node-disjoint paths did not achieve 100% reliability for a particular flow involved problems on links connected to the source or destination of that flow. We classified each interval in which two node-disjoint paths experienced problems and found that only about 3% of problems involved packets that were dropped or late due to problems on links *not* connected to the source or destination. Therefore, to close the performance gap between two disjoint paths and the optimal time-constrained flooding, we focus on problems involving the source or destination of a particular packet flow, as we find that such problems account for the vast majority of that gap.

Chapter 5

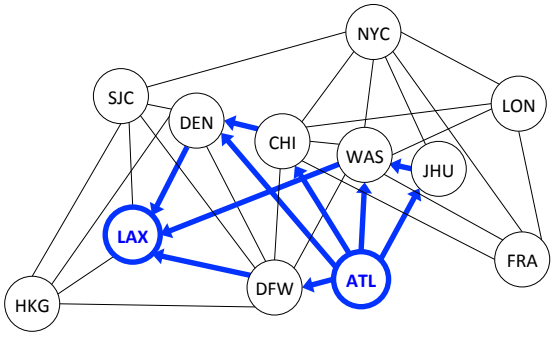
Dissemination-Graph-Based Transport Service using Targeted Redundancy

Based on the analysis described above, we design a new approach with the goal of achieving reliability close to that of time-constrained flooding (which is optimal), at a cost similar to that of two disjoint paths. Because we find that two disjoint paths generally perform well, avoiding loss incurred when a single-path approach would take time to react to and route around a problem, we use a dissemination graph consisting of two disjoint paths in most cases. To close the performance gap between two disjoint paths and the optimal time-constrained flooding, we focus on problems involving the source or destination of a particular packet flow, as we find that such problems account for the vast majority of that gap. Our approach is to use a dissemination graph consisting of two disjoint paths for each source-destination flow, except when a network problem is detected at the source or destination of the flow.

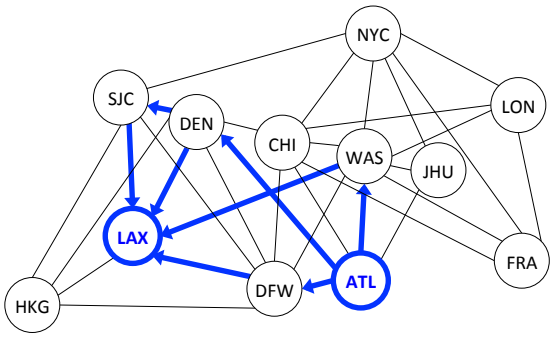
Fast reactions to network problems require both quick detection of problems and fast selection of graphs that can address those problems. Because calculating optimal dissemination graphs for arbitrary conditions is computationally intensive, our approach to enabling fast graph selection is to pre-compute a limited number of dissemination graphs that can address most common problems, converting the difficult optimization problem of computing a dissemination graph into a much simpler problem of classifying network events into a few broad categories. Based on the findings that two disjoint paths avoid many common problems and that problems that cannot be avoided using two disjoint paths generally involve a flow's source or destination, our approach is to use two dynamically computed node-disjoint paths in combination with three precomputed dissemination graphs. Offline, each source computes the following three dissemination graphs for each of its possible destinations:



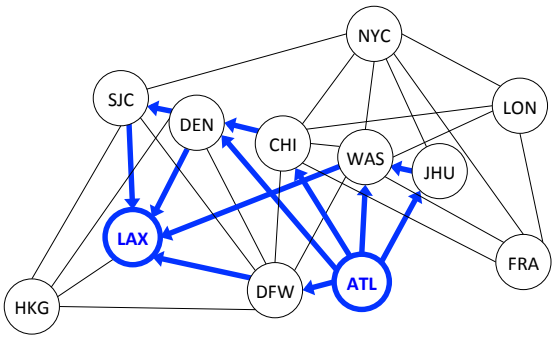
(a) Two node-disjoint paths, based on normal-case latencies (4 edges).



(b) Source-problem graph (10 edges).



(c) Destination-problem dissemination graph (8 edges).



(d) Robust source-destination-problem dissemination graph (12 edges).

Figure 5.1: Dissemination graphs for a flow from Atlanta to Los Angeles.

1. Source-problem dissemination graph
2. Destination-problem dissemination graph
3. Source-destination-problem dissemination graph

In the normal case, when no source or destination problems are detected, a source will use a dissemination graph consisting of two dynamically computed node-disjoint paths. The paths are computed based on current link latencies and loss rates, using the expected latency metric, as described in Section 3.2.3. Specifically, we use an algorithm based on the Ford-Fulkerson max-flow algorithm, as described in [36]. This approach minimizes the sum of the latencies of the two paths and works well for the practical overlay topologies we consider. Although we would prefer to select the paths such that the latency of the longer of the two paths is minimized, dynamically constructing the dissemination graph requires an efficient algorithm, and that problem is known to be NP-complete.

The source-problem and destination-problem graphs add targeted redundancy around the source or destination, respectively, while potentially reducing redundancy

in the rest of the network to keep costs as close to two-disjoint paths as possible. The robust source-destination-problem dissemination graph is more expensive, but adds redundancy around both the source and destination. In addition, when a source-destination problem graph is selected for a particular flow, it is dynamically combined with the current best two disjoint paths in order to create a highly robust graph. Figures 5.1a - 5.1d show examples of the four graph types for a flow across the US from Atlanta, Georgia to Los Angeles, California.

5.1 Constructing Dissemination Graphs with Targeted Redundancy

Unlike optimal dissemination graphs for arbitrary conditions, the three dissemination graphs that we precompute for each flow have well-defined structures, making them considerably easier to compute. Below, we describe how each of these graphs is computed.

5.1.1 Source-Problem and Destination-Problem Graphs

The source-problem and destination-problem graphs aim to maximize the number of ways out of the source or into the destination, respectively. For destination-problem graphs, we consider all overlay nodes directly connected to the destination, eliminating any that cannot be used by the source to reach the destination within the time constraint (i.e. the latency of the shortest path from the source to that node, plus the latency between that node and the destination exceeds the time constraint). We then find a tree that connects the source to all of these nodes.[1] The complete destination-problem dissemination graph consists of this tree, plus the edges connecting these nodes to the destination.

There are several possible methods for computing the tree that connects the source to the neighbors of the destination. The simplest approach is to use the shortest-path tree, as it is easy to compute and ensures the lowest possible latencies. The shortest-path tree may be a good practical choice for certain applications or large topologies, but it does not provide cost guarantees.

We currently use minimal-cost shallow-light Steiner trees [37, 38], which provide the lowest cost trees that ensure that the path from the source to each neighbor node is short enough to allow it to reach the destination on time. While finding such

[1] If the destination has many direct neighbors, the set of nodes to include can be pruned, for example, by eliminating the nodes on the highest-latency paths, or furthest from the destination (since recoveries are least likely to succeed in that case).

trees is an NP-hard problem, exact calculations of these graphs are feasible for our topology (and are likely to be feasible for many practical topologies, since they only need to be performed once, offline). The exhaustive search algorithm used in our current implementation is shown in Algorithm 2.

Algorithm 2 Shallow-light Steiner tree via exhaustive search

Given an input graph $G = (V, E)$, source s, destination t, set of target destination neighbors N and assignment of latency constraints $l : N \to \mathbb{R}^+$ to the target neighbors
Let $C \leftarrow V \setminus N \setminus \{s, t\}$
for $i \in [0, |C|]$ **do**
    for all C' such that $C' \subseteq C$ and $|C'| = i$ **do**
        feasible $\leftarrow$ true
        $V' \leftarrow C' \cup N \cup s$
        Let H be the subgraph of G composed of the vertices in V' and their adjacent edges in E
        for all $n_j \in N$ **do**
            $p(n_j) \leftarrow$ the shortest path from s to n_j in H
            if $\nexists p(n_j)$ or length $p(n_j) \geq l(n_j)$ **then**
                feasible $\leftarrow$ false
                break
            end if
        end for
        if feasible = true **then** return $\bigcup_{n_j \in N} p(n_j)$
        end if
    end for
end for

The running time of Algorithm 2 is exponential in the number of non-target vertices, excluding the source and destination (i.e. the size of the set C): specifically, its runtime is $2^{|C|} \times O(|E| + |V^2|)$ or $2^{|C|} \times O(|E| + |V| \log |V|)$, depending on the implementation of Dijkstra's algorithm used for finding shortest paths. For a small number of vertices (e.g. 15 to 20, which is sufficient to provide global coverage), this is quite practical. Moreover, using more sophisticated algorithms, exact solutions can be found in time exponential only in the number of target neighbor nodes, which is likely to be small even in larger networks (since the fanout of each node is typically bounded, and for destinations with large fanout, the target neighbor set would likely be pruned to reduce costs) [38, 39]. However, if exact calculations are not feasible, such trees can also be approximated (e.g. [37, 40]).

Note that algorithms for the shallow-light Steiner tree problem are typically specified with a single latency constraint for all targets, which does not quite fit our problem. Because we are constructing a tree that connects the source to all of the

neighbors of the destination, we must also account for the time required to get from those neighbors to the destination itself. Therefore, for each neighbor v of the destination, we have a different latency constraint $l(v) = L - l(e_{(v,t)})$, which reduces the allowed latency by the distance between the neighbor and destination. Algorithm 2 is specified to account for these different latency bounds, and other algorithms can generally be adapted to take this into account. For example, for the algorithm in [38] that provides an exact solution in time exponential only in the number of target neighbors via reduction to the Directed Steiner Tree problem, this can be done by modifying the layered graph constructed in the reduction.

While the above discussion was framed in terms of destination-problem graphs, the same approaches can be used for source-problem graphs by simply reversing the edges of the graph and treating the source as the destination. Reversing the edges in the final solution then gives a graph that connects the source to all of its neighbors that can provide an on-time path to the destination and connects all of those neighbors to the destination via a tree that respects the latency constraint.

5.1.2 Robust Source-Destination-Problem Graphs

The robust source-destination-problem graphs are more expensive than the other graphs types, but this is acceptable because they are only used rarely, in the case of multiple problem types occurring simultaneously in the network. These graphs are precomputed to add redundancy around both the source and destination and dynamically modified to ensure that there are at least two node-disjoint paths between the source and destination.

In our approach, the precomputed base graphs are created by simply taking the union of the edges in the source-problem and destination-problem graphs. At the time that the graph is used, it is combined with the current best two node-disjoint paths (based on network conditions), creating a highly robust graph, composed of the union of all three other graph types used in our approach (dynamic two paths, source-problem, and destination-problem). While this method does not provide strict cost guarantees, that is acceptable for these infrequently used graphs that must be highly robust to failures.

Developing a principled method of constructing dissemination graphs that provide the necessary robustness while also minimizing cost is a challenging problem. One method we have considered is to construct a highly robust graph that connects each neighbor of the source to each neighbor of the destination. Intuitively, this ensures that (assuming there are no problems in the middle of the network), it is sufficient for the source to successfully transmit the packet to any one of its neighbors and for the destination to be able to receive the packet from any one of it neighbors (since each neighbor of the source will forward the packet to all neighbors of the destination).

Unfortunately, we have shown that such computing such graphs is W[1]-hard with respect to the number of neighbor combinations, even in the case that all edges are of unit length and unit cost, making it unlikely to be feasible as a general approach [38].

5.2 Quick Problem Detection System

Fast rerouting is accomplished using a quick detection system in which each overlay node monitors each of its links, flooding an update to all of the other overlay nodes whenever it detects that a new problem has started on one of its links or that an existing problem has resolved. When the number of problematic incoming links for a given node exceeds a certain threshold, each source will switch to using a destination-problem graph for that destination. Similarly, if a node detects problems on a threshold number of its outgoing links, it will switch to using source-problem graphs. The source-destination-problem graphs are used when there are problems at both the source and destination. In addition, if a source-problem or destination-problem graph is selected for a given flow, and a problem is also detected on another link of that dissemination graph (not at the source or destination), the robust source-destination-problem graph will also be used.

While all specific thresholds are configurable, we currently consider a link problematic if its measured loss rate is greater than 2% (a loss rate of exactly 2% is not considered problematic) or if its latency is at least 25% above its baseline latency. A source is considered problematic if at least two of its outgoing links are classified as problematic, and a destination is considered problematic if at least two of its incoming links are classified as problematic.

This approach is scalable to large numbers of simultaneous packet flows, as it requires only a small monitoring overhead per overlay link. All simultaneous flows between a particular source-destination pair can use the same dissemination graph, so no per-flow monitoring is needed. Note that if the same overlay deployment needs to support applications with significantly different latency constraints, it may be necessary to maintain multiple graph sets for each flow (one for each application class), but in practice it is likely simpler to run separate overlay instances in parallel, with applications distributed across them based on their timeliness constraints.

5.3 Potential Optimization: Faster Reaction

One potential drawback of our source-based routing approach compared with link-state routing is that information about changes in network conditions must be propagated back to the source before a flow can be rerouted. We currently consider this

delay acceptable, because it is not clear that a non-source-based scheme can guarantee that the desired graph properties are maintained, and because the propagation delay is typically relatively small. We target applications with demanding latency constraints (on the order of 65ms), so clearly for such applications, the propagation delay cannot be more than the required latency constraint (i.e. the service is not feasible unless the propagation delay from source to destination is less than 65ms). In general, the propagation delay between any two points on the globe (e.g. Europe to Asia, passing through North America, which is a distance greater than half the globe) is within about 150ms.

However, the flexibility of dissemination graphs offers an avenue for optimizing the reaction time of the protocol by allowing intermediate nodes to add edges to the dissemination graph on a packet. We do not want to give intermediate nodes full power to determine the graph, since intermediate nodes may have different views of the current network conditions, and it is not clear how to ensure that the packet traverses, for example two paths that are actually node disjoint in that situation. However, *adding* edges to a graph can only improve reliability (assuming sufficient node processing power and total bandwidth, which the overlay service provider provisions). Therefore, one approach is to allow intermediate nodes in a dissemination graph to increase the redundancy of the dissemination graph if they detect problems that are not accounted for in the dissemination graph on a received packet. For example, if the source is not yet aware of a problem at the destination (and so uses a dissemination graph consisting of two disjoint paths), nodes closer to the destination that are already aware of the problem could modify that graph, for example by simply adding the edges in the destination-problem graph to increase redundancy.

While such an approach can improve reaction times, it complicates the protocol, as intermediate nodes need more involved protocol-specific logic, rather than simply forwarding each packet based solely on the edges in the dissemination graph stamped on it. Considering this trade-off, we currently choose to favor protocol simplicity and do not attempt to optimize reaction time in this way.

5.4 Evaluation via Simulation

To assess the performance of our dissemination-graph-based routing approach that adds targeted redundancy during problems involving a flow's source or destination, we use our Playback Network Simulator, as described in Section 4.1. We analyze how the targeted redundancy approach would perform over the four weeks of data we collected over several months (from July to October 2016) and compare it to the initial dissemination graph construction approaches we considered in Section 4.3. For the single-path dissemination-graph approach, we additionally consider a *redundant single path* scheme, in which each message is originally transmitted twice, separated by 0.5ms, as this represents the simplest approach to adding redundancy in packet

transmission.

We consider the same sixteen transcontinental flows as in Section 4.3, modeling a sending rate of one packet per millisecond for each flow. This rate corresponds well to the applications we target (described in Chapter 7). All results consider a 65ms one-way latency deadline, since we aim to support highly interactive remote manipulation applications. While the complete transport service uses the recovery protocol of [1], here we present results both with and without recoveries to assess the benefit provided by both the routing and recovery protocol components.

5.4.1 Overall Performance

Table 5.1 presents overall reliability and availability results for each of the dissemination-graph-based routing approaches we consider, aggregated over all sixteen flows across the US and over all four weeks of data collection. We say that a flow is *unavailable* if the loss rate on that flow exceeds 50% over a period of at least one second. As seen in Table 5.1, over the course of a week, the average unavailability for a flow using a single-path approach is about 35 seconds, or about 25-29 seconds using any of the other approaches. This translates to an overall availability of about 99.994% for a single path and 99.995-99.996% for the other approaches.

For the time that a flow is available, we calculate its *reliability*, or the percentage of packets delivered within their latency deadline. From Table 5.1, we see that both time-constrained flooding and our targeted redundancy approach reach nearly 99.9999%. This translates to about 1.4-1.5 packets per million that do not arrive within their deadline using these approaches, compared to 9-23 packets per million that do not arrive on time for the other approaches, representing close to an order of magnitude improvement.

Routing Approach	Availability (%)	Unavailability (seconds per flow per week)	Reliability (%)	Reliability (packets lost or late per million)
Time-Constrained Flooding	99.995883%	24.90	99.999863%	1.37
Targeted Redundancy (via Dissemination Graphs)	99.995864%	25.02	99.999849%	1.51
Dynamic Two Disjoint Paths	99.995676%	26.15	99.999103%	8.97
Static Two Disjoint Paths	99.995266%	28.63	99.998438%	15.62
Redundant Single Path	99.995223%	28.89	99.998715%	12.85
Single Path	99.994286%	34.56	99.997710%	22.90

Table 5.1: Aggregate availability and reliability with 65ms latency constraint, over four weeks and sixteen transcontinental flows (using the recovery protocol of [1]).

Table 5.2 shows the same availability and reliability metrics as Table 5.1 but without using any recovery protocol to retransmit lost packets. These results show a very similar overall pattern but with lower overall reliabilities. The use of the

Routing Approach	Availability (%)	Unavailability (seconds per flow per week)	Reliability (%)	Reliability (packets lost or late per million)
Time-Constrained Flooding	99.995883%	24.90	99.999702%	2.98
Targeted Redundancy (via Dissemination Graphs)	99.995863%	25.02	99.999687%	3.13
Dynamic Two Disjoint Paths	99.995676%	26.15	99.998736%	12.64
Static Two Disjoint Paths	99.995264%	28.64	99.998156%	18.44
Redundant Single Path	99.995222%	28.90	99.998519%	14.81
Single Path	99.993974%	36.44	99.997190%	28.10

Table 5.2: Aggregate availability and reliability with 65ms latency constraint, over four weeks and sixteen transcontinental flows (no recovery protocol).

recovery protocol does not significantly affect availability, as periods of unavailability are generally caused by complete disconnections, which cannot be overcome through recoveries.

5.4.2 Comparison of Approaches

As time-constrained flooding sends packets over every link that can possibly improve reliability, it provides an upper bound on the performance that any dissemination graph or path can achieve (using the same recovery protocol). Since single-path approaches are commonly deployed today, we use our single-path approach as a baseline. We consider the performance gap between time-constrained flooding and a single path as the scale for measuring the performance of other approaches.

Specifically, for each week, we calculate this performance gap as the difference between the total number of packets delivered by their 65ms deadline using the single-path approach and using time-constrained flooding. Across all sixteen flows, with each flow sending at a rate of one packet per millisecond, time-constrained flooding would deliver 68,112 more packets on time than a single path in Week 1, 480,806 more packets in Week 2, 115,152 more packets in Week 3, and 615,397 more packets in Week 4, for a total of 1,279,467 more packets delivered on-time across all four weeks (out of over 38.7 billion total packets). Table 5.3 shows what percent of this performance gap each approach covers, aggregated over all sixteen flows for each of the four weeks we consider.

These results show that our dissemination graph approach with targeted redundancy achieves close to optimal reliability, covering 98.97% of the gap between time-constrained flooding and single-path routing (for a total of 1,266,286 additional packets delivered on time compared to single-path routing). While two disjoint paths offer a substantial improvement over a single path, they do not reach the same level of reliability, covering about 70% of that gap if dynamic reroutes are used and about 40% if the paths are static.

In addition, the "Scaled Cost" column of Table 5.3 shows that our approach is

Routing Approach	Week 1 2016-07-12	Week 2 2016-08-08	Week 3 2016-09-01	Week 4 2016-10-13	Overall	Scaled Cost (Dollar Cost)
Time-Constrained Flooding	100.00%	100.00%	100.00%	100.00%	100.00%	14.350
Targeted Redundancy (via Dissemination Graphs)	94.19%	99.19%	98.00%	99.50%	98.97%	2.203
Dynamic Two Disjoint Paths	80.91%	71.34%	47.73%	73.46%	70.74%	2.197
Static Two Disjoint Paths	-76.72%	50.89%	53.58%	40.79%	39.50%	2.194
Redundant Single Path	54.12%	37.25%	4.89%	59.10%	45.75%	2.000
Single Path	0.00%	0.00%	0.00%	0.00%	0.00%	1.000

Table 5.3: Percent of the benefit of time-constrained flooding obtained by each approach and scaled cost (baseline is single-path).

able to provide this performance improvement at a cost increase of less than 0.3% compared with two disjoint paths, with two disjoint paths costing a little less than 10% more than twice the cost of the single best path. The source-problem and destination-problem graphs used by our approach may include about twice as many edges as a graph consisting of two disjoint paths (as shown in Figures 5.1b and 5.1c), and the more expensive source-destination-problem graphs can include three times as many edges (as in Figure 5.1d). However, the more expensive graphs are used infrequently: in our analysis over four weeks, our targeted redundancy approach uses the (dynamically computed) two node-disjoint paths dissemination graph 99.670% of the time, the source-problem graph 0.117% of the time, the destination-problem graph 0.159% of the time, and the source-destination-problem graph 0.054% of the time. This means that over the course of a week for a given flow, the source-problem graph is used for a little less than 12 minutes, the destination-problem graph is used for about 16 minutes, and the source-destination-problem graph is used for about 5.5 minutes. Therefore, the approach incurs a small total overhead compared with two disjoint paths. In contrast, time-constrained flooding has an overhead of more than 6.5 times the cost of two disjoint paths (or over 14 times the cost of a single path), making it too expensive for practical use. Note that while we only list the dollar cost metric in Table 5.3, the goodput cost metric is nearly identical: for the precision we consider, the only change is that time-constrained flooding's scaled cost would be reduced to 14.349 from 14.350.

While the overall pattern of results is fairly consistent across the four weeks, Week 1 and Week 3 show a few interesting differences. The Week 1 results illustrate the major drawback of using two static disjoint paths: such an approach will continue to use the same paths even if they suffer a complete disconnection or other severe problem, which can cause it to perform much worse than even a single-path approach that is able to route around problems. In Week 1, such incidents occur several times, causing the static two disjoint paths approach to perform considerably worse than the dynamic single path approach. This is the reason for the -76.72% figure for static two disjoint paths in Week 1: in this week, the static approach does not cover any of the performance gap between the single path approach and time constrained flooding.

Instead, it actually *underperforms* the single path approach by 76.72% of that gap.

For a simple example of how this can happen, we can consider an event on July 15, 2016, where the outgoing links from San Jose to Dallas and from San Jose to Denver were both disconnected nearly simultaneously. During this time, the outgoing links to Los Angeles and New York continued to work without problems, allowing time-constrained flooding to operate without any interruption and allowing all of the dynamic routing approaches to reroute within about 500ms. In contrast, the static two disjoint paths approach continues to attempt to use the same two disconnected paths, and is not able to resume successful delivery until the disconnection ends over 1 second later. More dramatically, substantially elevated latency on those same links later that same day results in a full 25 seconds during which packets cannot be delivered on time on the San Jose to Baltimore flow, while the other routing approaches experience negligible interruptions. This effect contributes to the redundant single-path approach outperforming two static disjoint paths overall, although we note that our independent loss modeling also provides the best-case simulated performance for redundant sending (as it assumes that the loss probability of the second copy of a packet is independent of the outcome of the first packet).

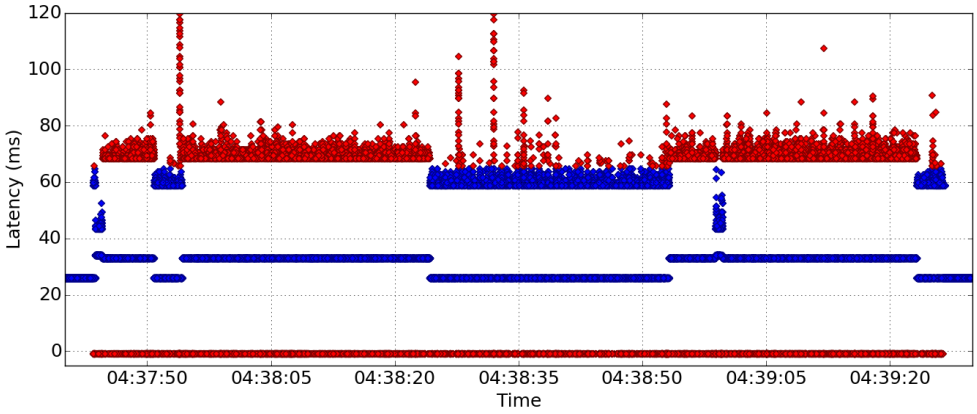
In Week 3, however, we see another unusual pattern, where the redundant single-path approach provides only a minimal benefit compared to the basic single-path approach (covering only 4.89% of the gap between it and time-constrained flooding, compared to the 35-60% it normally covers). This is because many of the problems in Week 3 stem from link disconnections that lead to suboptimal routing decisions (i.e. choosing paths that may be loss-free but cannot deliver packets on time; see Section 6.5.2.3 for a detailed examination of one such incident). Because the redundant single-path approach uses the same routing algorithm, it is limited to essentially the same performance as the single-path approach: sending packets multiple times on a path that cannot reach the destination on time does not provide any higher reliability than sending them only once on that path.

5.4.3 Case Study

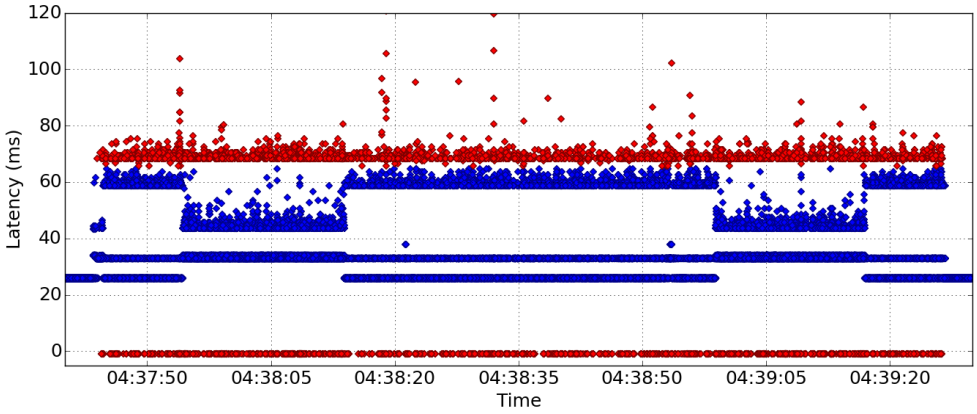
While our dissemination graph approach performs comparably to two disjoint paths during periods when the network is largely loss-free, it can provide a dramatic improvement during problematic periods. Table 5.1 shows very low loss when averaged over the course of several weeks, but this loss is not uniformly distributed over time: the service should provide acceptable performance even during periods of highly concentrated loss. Figures 5.2a and 5.2c show the performance of the flow from Atlanta to Los Angeles during one such period, occurring on August 15, 2016 (during Week 2). In these figures, blue dots represent packets delivered within the 65ms latency constraint, while red dots represent packets delivered after 65ms or dropped (dropped packets are shown at 0ms latency). During this 110-second interval, all of

the Los Angeles node's incoming links were experiencing loss ranging from about 20% to 70%. Using dynamic single path routing with real-time hop-by-hop recovery results in an effective rate of lost or late packets of about 24.9%, while using two disjoint paths plus recovery reduces it to about 4.6%. In this case, our approach provides more than an order of magnitude improvement compared with two disjoint paths, reducing the rate of packets not delivered within 65ms to about 0.3% and matches the (optimal) performance of time-constrained flooding.

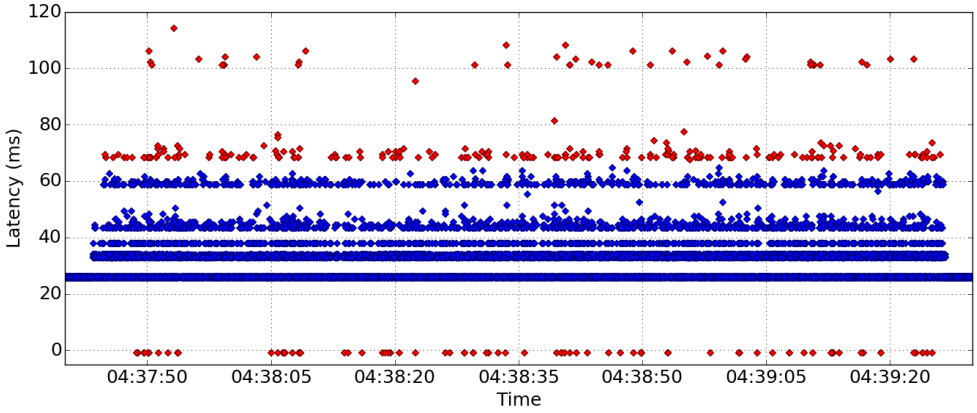
This improvement stems from the fact that the destination-problem dissemination graph from Atlanta to Los Angeles provides additional ways into the Los Angeles node, and additional opportunities for lost packets to be recovered. This can be seen in the distinct "stripes" in Figures 5.2b and 5.2c. Using two paths (Figure 5.2b), one path generally uses the Dallas overlay node as its last hop before reaching Los Angeles and has a latency of about 26ms (lowest blue stripe), while the other goes through Denver and has a latency of about 32ms (second blue stripe). Packets that are initially lost but subsequently recovered on the shorter path generally arrive with a latency of about 60ms (blue stripe at 60ms), while packets recovered on the longer path often arrive too late, with latencies around 70ms (highest red stripe). For two substantial periods, a path with San Jose as the next-to-last hop is used instead, resulting in base latency around 33ms, but faster recoveries (around 43ms). Figure 5.2c illustrates the impact of the destination-problem graph, including several additional blue stripes that represent the additional paths and recovery opportunities available between the source and destination by making use of all of the Los Angeles node's relevant neighbors.



(a) Single path (27,353 lost or late packets, 5 packets with latency over 120ms not shown).



(b) Two node-disjoint paths (5,100 lost or late packets, 15 packets with latency over 120ms not shown).



(c) Novel dissemination-graph-based approach, adding targeted redundancy at the destination (338 lost or late packets).

Figure 5.2: (Simulated) packets received and dropped over a 110-second interval on August 15, 2016 from Atlanta to Los Angeles.

5.5 Dissemination Graph Work Evolution

We first introduced the concept of a timely, reliable, and cost-effective transport service using dissemination-graph-based routing with targeted redundancy in [32]. That work included a dissemination-graph-based routing protocol similar to the one presented in this chapter and evaluated its effectiveness using the same four weeks of network data that we analyze here. However, the protocol and analysis in this chapter differ from [32] in several important ways.

First, the dissemination-graph-based algorithm we present here is an improvement on the algorithm presented in [32]. That work used a simpler approach, in which the two disjoint paths used by the protocol were static, so all four dissemination graph types could be precomputed, and the protocol only needed to switch between them based on network conditions. While the simplicity of this approach is appealing, and it was shown to provide excellent results using our collected data, relying solely on static graphs is risky for practical deployments that require high availability. Although the graphs used by the approach provide high reliability overall, it is still possible (if rare) to encounter a situation in which even the most robust graph used by the approach does not include a currently functional means of reaching the destination on time, even though some such path exists (and could be found by a dynamic approach). Therefore, the new protocol presented in this thesis uses two dynamic node-disjoint paths in the normal case, and incorporates them into the robust source-destination problem graphs as well.

Note that an alternative approach using only static graphs would be to fall back to overlay flooding or time-constrained flooding in the case of a problem that is not handled by the robust source-destination graph. However, such an approach requires significantly higher overprovisioning of the network bandwidth to ensure that the network can support the worst case of all flows using time-constrained flooding and therefore may not be feasible. In contrast, our dynamic approach actually tends to reduce costs overall, as we use the source-problem, destination-problem, and source-destination problem dissemination graphs less often than in the protocol from [32]. Because the previous protocol used two static paths, it shifted to a more robust graph as soon as one source or destination problem was detected if that problem fell on one of the edges included in the two static paths. With two dynamic paths, this is not necessary: rerouting the two paths effectively addresses this problem, so we never switch to a more robust problem-type graph unless at least two problems of the relevant type are detected.

Second, the analysis in this chapter uses an updated version of the Playback Network Simulator, with routing and disconnection modeling that more closely match the behavior that would be found in a real overlay implementation. Therefore, the results presented in this chapter are not directly comparable with the results from [32], as both the protocol and simulation have been modified.

Chapter 6

Implementation

The dissemination-graph-based transport service developed in this thesis is implemented in the Spines overlay messaging framework and is available as open source as part of Spines [8]. The implementation builds on the existing capabilities in Spines to deploy overlay networks, dynamically monitor network characteristics, and perform overlay-level routing and hop-by-hop recovery [41]. It augments those existing capabilities with a generic framework for dissemination-graph-based routing, an ability to route based on which links are classified as problematic (in terms of their current loss and latency), and an implementation of the specific dissemination-graph-based routing protocol described in Chapter 5.

6.1 Generic Dissemination-Graph-Based Routing

We implemented dissemination-graph-based routing in Spines using a bitmask with one bit per (directed) link in the overlay topology to compactly represent the graph. When sending a packet, the source specifies the dissemination graph to use for that packet by setting exactly the bits in the bitmask that correspond to the links that should be included in that packet's dissemination graph. When an intermediate node receives a packet to forward, it loops through all of its neighbors and checks whether the edge from itself to that neighbor is set in the bitmask on the packet; if it is set, it simply forwards the packet to that neighbor.

While a bitmask large enough to represent all the directed edges in the overlay topology must be included in the header of each packet, this small overhead is not a limiting factor for the approach: we currently use exactly 64 bits (one word) to represent all the overlay links in a globe-spanning topology and expect 128 bits (two words) to be sufficient to represent most overlay topologies of the size needed to support our target applications, although any size bitmask is supported.

De-duplication. When dissemination graphs are used, packets may be disseminated over graphs that are more complex than a single path or fully disjoint paths. In this case, an intermediate node may receive copies of the same packet from several of its neighbors. In such cases, the intermediate node should not forward the packet each time it receives one of the copies: it instead performs de-duplication, only forwarding the packet the first time it is received. This ability to perform de-duplication in the middle of the network is enabled by the structured overlay's ability to maintain flow-based state at each overlay node.

To allow de-duplication, each source maintains a sequence number for the packets that it sends using source-based routing. Each time that source sends a packet using source-based routing, it increments this sequence number and stamps it on the packet. Each overlay node maintains a fixed-size history for each other node in the network; this history temporarily stores the sequence numbers of packets that this node has received from those sources. The history is implemented as a simple circular buffer. Upon receiving a packet, the node looks up its sequence number in its buffer for that packet's source (by simply using the sequence number modulo the buffer size to get the correct index in the buffer). If the received packet is newer (i.e. has a higher sequence number) than the stored sequence number, the node forwards the packet and replaces the stored sequence number with the new one. If the packet's sequence number is identical to or older than the stored one, it is discarded.

Note that if there is a large amount of packet reordering, it is possible for a node to incorrectly reject a packet that it never forwarded as a duplicate, because it may have already received a newer packet for the relevant buffer slot. However, by correctly setting the size of the buffer, we can ensure that nodes will never discard a packet that may still be useful. For protocols that require timeliness and may not be 100% reliable (like the ones used for our timely, reliable transport service), this simply means setting the buffer large enough that any packet that falls outside the current history window is already past its delivery deadline (and therefore no longer useful). Specifically, the buffer should be able to hold at least a number of packets equal to the maximum sending rate of any source (in packets per millisecond) multiplied by the longest application delivery deadline being served (in milliseconds). This size can be configured for the overlay based on the applications that it needs to support. For example, we currently use a history of 100,000 sequence numbers. Targeting flows with sending rates of 1 packet per millisecond and deadlines of about 65 milliseconds, this configuration can support over 1500 such flows per source simultaneously. Of course, if more flows or higher sending rates are needed, this value can easily be increased: storing 100,000 8-byte sequence numbers for each of about 20 sources requires about 16 MB of memory, which is not a problem today. For fully reliable protocols, the history should be at least the size of the windows used for reliability (although the combination of dissemination-graph-based routing and a reliable link protocol is not yet implemented in Spines).

Finally, note that to work across overlay node restarts, each source sequence con-

sists of two parts: a 32-bit sequence number and a 32-bit incarnation. The sequence number part is incremented each time the source introduces a new message, as described above. The incarnation part is set to the current time (in epoch seconds) when the Spines daemon starts up. A packet is considered new if its incarnation is later than the stored incarnation, or the incarnation is equal and the sequence number is higher. Without time-based incarnations, a source that went down, came back up, and restarted its sequence number at one would have its messages discarded by the other nodes until it eventually reached a sequence number higher than the highest one it had sent prior to restarting. Of course, if a node starts up, goes down, and restarts within one second (causing it to choose the same incarnation on restart), its initial messages may still be discarded, but since the number of messages it could have sent in the fraction of a second that it was up is likely limited, this is not a serious problem in practice. This approach also handles the range limitation of the 32-bit sequence number: once the sequence part rolls over, the source simply increments the incarnation part and allows the sequence part to restart (of course, these can be increased to 64-bit numbers to mitigate this issue as well, at the cost of using larger headers and more memory to store the history).

6.2 Problem-Type Routing

The dissemination-graph-based transport service described in Chapter 5 depends on the ability to classify links as problematic. To implement this in Spines, we add both a new flag to propagate information about problematic links throughout the network and a concept of a baseline weight to determine if a link is experiencing higher than normal latency.

Baseline weights are specified in the Spines configuration file that each daemon reads when it starts up. This configuration file specifies all of the nodes and links in the overlay topology, as well as a baseline latency for each link. This baseline latency corresponds to the roundtrip latency of that link under normal conditions and can be measured prior to instantiating the overlay network by simply pinging between the relevant overlay nodes. If baseline latencies are not available, Spines could be set up to ignore latency for the purpose of problem classification and only detect problems based on loss, but this may result in some problems not being identified or being identified later than would have been possible otherwise.

As part of the normal operation of Spines, each node monitors its connections to its overlay neighbors, tracking the status (connected/disconnected), loss rate, and latency on each of its links. Based on this information, the node calculates a link weight for each of its outgoing links (which varies based on the link-weight metric used), and when the weight for a link changes, it floods that information throughout the overlay network. We augment the information in the flooded update with a flag indicating whether the link is currently judged to be problematic, based on the

thresholds set up in Spines. In principle, this flag can be used to allow problem-type routing to coexist with any of the link-weight metrics in Spines, although the current implementation only supports its use with the expected latency metric designed for applications that require both timeliness and reliability (described in Section 3.2.1). The calculated link weight can be used normally to compute shortest paths or disjoint paths, while the flag can be used to switch between the graph classes (two disjoint paths, source-problem, destination-problem, and robust source-destination-problem graphs) in our approach (and could be used by new routing protocols as well).

Note that in order to avoid flooding updates for minor link weight variations, updates are only sent if the change in link weight crosses a significance threshold. A change is considered significant if it changes the state of the problem flag (i.e. if a problem starts or is resolved) or if the value of the metric changes by more than a threshold percentage (with a default of 10% in Spines today). To improve stability, we also enforce that a link weight cannot decrease for at least 30 seconds after it was last increased. The default problem threshold settings in Spines are set such that a link will be considered problematic if its loss rate exceeds 2% or if its measured latency is 25% higher than its baseline latency. A problematic link will be restored to normal status once its loss rate returns to 0.5% or less, and its latency is within 15% of its baseline value.

6.3 Timely, Reliable Transport Service

We use the generic dissemination-graph-based routing and problem-type routing capabilities described above to implement the timely, reliable transport service described in Chapter 5.

Our transport service uses the existing real-time link protocol in Spines to provide timely hop-by-hop recovery. As explained in Section 2.4, the real-time recovery protocol allows a packet lost on a given overlay link to be requested and retransmitted at most once, and only if it can be recovered within its timeliness constraint. Each node stores packets for potential retransmissions until they are requested (since after a packet is requested once, it will never be asked for again) or until its deadline has passed (since it is no longer useful after that point) [1].

At startup, each node computes the three static dissemination graphs options (source-problem, destination-problem, and source-destination-problem) that it will use in routing flows from itself to each of the other node in the network. It generates and stores the bitmask representing each of these graphs.

To perform dissemination-graph-based routing using the approach described in Chapter 5, each node keeps track of which type of dissemination graph it is currently using for each other node in the overlay topology. When a node has a new packet to introduce into the network for a particular destination, it looks up what graph type is currently being used for that destination. If a source-problem or destination-problem

graph is currently being used, the node simply copies the appropriate precomputed bitmask into the packet header and forwards the packet on all of its outgoing links included in the bitmask. If a two node-disjoint paths graph is currently being used, the source checks whether it already has the relevant two paths graph cached; if so, it can simply copy it, otherwise it computes it as described in Chapter 5, copies it into the packet header and adds it to the cache (the full cache of computed paths is cleared each time a link weight is updated). If a robust source-destination-problem graph is being used, the source determines the dynamic two paths component of the graph as in the two paths graph case and then performs a bitwise "or" operation to combine it with the precomputed source-destination-problem graph before stamping it on the packet.

The current graph type used for a destination is updated in response to link weight updates. Recall that each node is responsible for sending out updates about changes on its outgoing links and receives flooded updates from the other nodes regarding their outgoing links. As discussed above in Section 6.2, links are classified as problematic when they experience loss or elevated latency. A source will switch to using a destination-problem graph for a particular destination if a threshold number of that destination's incoming links are currently classified as problematic. It will switch to using a source-problem graph if a threshold number of its own outgoing links are classified as problematic. We currently use a threshold of two problematic links, as a dynamic two-disjoint-paths graph can successfully address any single problem by relying on the second path to deliver packets until the problem is detected and then routing away from the problematic link once it is detected.

The robust source-destination-problem graph will be used for a flow when multiple problem types occur simultaneously. If both a source problem and a destination problem are detected for a given flow, the source-destination-problem graph will be used. Moreover, if either a source- or destination-problem graph is being used and a new problem is detected on a link that is in the active dissemination graph but is not part of the known source or destination problem (i.e. that is not an incoming link for the destination in the case of a destination-problem graph or that is not an outgoing link for the source in the case of source-problem graph), the source will switch to using the robust source-destination-problem graph.

If the current network conditions for a given destination do not fall into any of the problem classes described above, the dynamically computed dissemination graph consisting of two node-disjoint paths (chosen to minimize the sum of their expected latencies based on current conditions) is used.

6.4 Practical Considerations and Optimizations

We have presented a dissemination-graph-based routing approach that aims to be simple, cost-effective, and reliable. While the core service we presented selects one particular trade-off between the factors of simplicity, cost, and reliability, in practice, we find that for high-value applications it can be desirable to even further optimize reliability (at the expense of slightly increased cost or reduced algorithmic simplicity). Here we describe several modifications to the core service that we use in practice or believe can provide even better service for high-value applications.

Shortest-Path-Tree Dissemination Graphs. Section 5.1.1 presents a method of computing source-problem and destination-problem dissemination graphs based on shallow-light Steiner trees that minimize the cost of connecting the source (or destination) to all of the relevant neighbors of the destination (or source). While the method described ensures that all paths from the source to the destination through the target neighbors are within the specified latency constraint, they may not be the lowest possible latency paths, as we optimize for cost over latency.

In practice, we often prefer to optimize for latency over cost. Lower latency paths can provide higher reliability in practice, as they reduce the likelihood of latency fluctuations causing packets to miss their deadlines and make it more likely for recoveries of lost packets to be successfully completed within the time constraint. While using shortest path trees does not provide any formal guarantee that the cost of the dissemination graph is close to the minimal possible cost, in practice the overall increase in cost is small, especially since these graphs are only used when problems occur (less than 1% of the time, as reported in Section 5.4). For these reasons, shortest-path trees are currently used in constructing dissemination graphs in our Spines implementation.

Robust Source-Problem and Destination-Problem Graphs. In addition to improving the latency provided by the source-problem and destination-problem graphs, we can also improve their reliability, again at the expense of a small increase in cost. The robust source-destination-problem graphs described in Section 5.1.2 ensure that there are at least two disjoint paths through the network, but the source-problem and destination-problem graphs, as described in Section 5.1.1, do not. To improve reliability, we can also ensure that the source-problem and destination-problem graphs, in addition to including all of the relevant source or destination neighbors, include at least two disjoint paths. This approach can provide better robustness to problems that occur in the middle of the network while a source or destination problem is ongoing. While the approach specified in Chapter 5 will switch to the robust source-destination-problem graph if a problem starts in the middle of the network while a source-problem or destination-problem graph is being used, loss may occur during the time it takes to detect the problem and switch graphs. Ensuring the that source-problem and destination-problem graphs have two disjoint paths already could

allow them to handle such problems without the need to switch to the robust source-destination-problem graph and without incurring any loss. While this optimization is not implemented today, it is useful to consider for high-value applications.

6.5 Evaluation

To evaluate the implementation of our dissemination-graph-based transport service, we conduct several sets of experiments. First, we evaluate the implementation using several small overlay topologies deployed in a cluster with emulated latencies between overlay nodes and specific loss rates applied to selected edges in the topology. These experiments are designed to validate the simulation results, showing that the Playback Network Simulator accurately captures the salient performance characteristics needed to guide overlay routing protocol design.

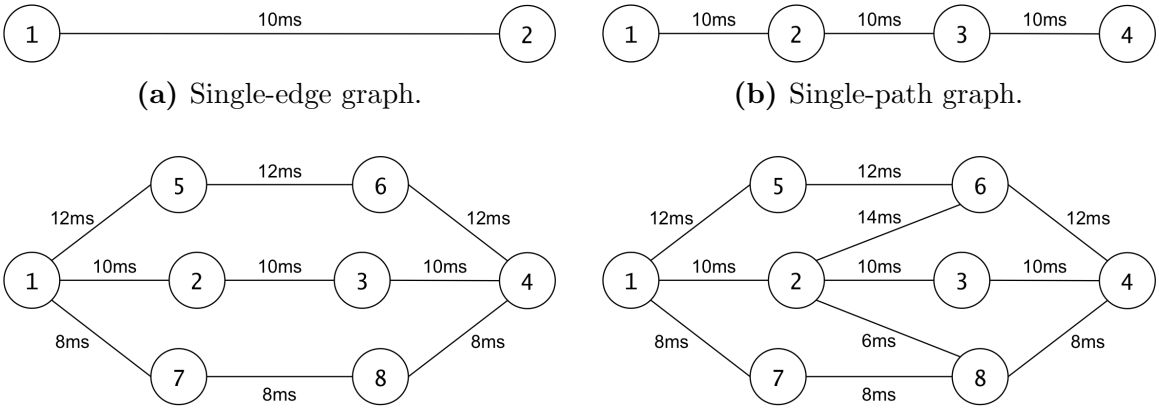
After validating the simulation results, we evaluate the implementation in a more realistic setting, using case studies drawn from the network data collected in July through October 2016. The case studies are evaluated both in a local-area cluster environment with emulated latencies and loss rates that we can fully control, and on a real global overlay network, with real latencies and emulated loss. These results demonstrate that, as shown in the simulation results, our novel dissemination-graph-based routing protocol significantly improves on the performance of protocols using a single path or two disjoint paths, and the transport service is able to provide the required timeliness and reliability for our target applications even during severe network problems.

6.5.1 Controlled Evaluation of Implementation with Simulation Validation

To validate the simulation results, we run our Spines implementation in parallel with the Playback Network Simulator's data collection component. We then use the collected data to simulate the same flows that we run in Spines and compare the results. Spines and the data collection are run at the same time, but use different ports and each have their own stream of packets.

6.5.1.1 Setup

These experiments are done using a series of relatively simple overlay topologies to isolate the various factors affecting overall performance. These topologies are shown in Figure 6.1. The latencies shown in the figures are emulated using the Linux NetEm utility. For experiments evaluating performance under various loss conditions, loss is also applied using NetEm across all ports. Each experiment is repeated five times, and



(a) Single-edge graph.

(b) Single-path graph.

(c) Three-path graph.

(d) Three-path graph, plus two edges.

Figure 6.1: Simple overlay topologies for simulation validation experiments

unless otherwise specified, results are averaged over the five runs. For the simulation results, each of the five runs is analyzed using ten different random seeds, and the results are averaged over those ten seeds and five runs. Each run lasts 5 minutes.

A key factor affecting the accuracy of Playback's simulation is the granularity of data collection: the higher the sending rate of the data collection component, the finer-grained the data will be, and the more accurate the simulation can be. Therefore, each experiment is run using two different data collection intervals: 1ms and 10ms. A data collection interval of 1ms means that Playback's data collection program sends a message on each link of the topology every millisecond. This matches the sending rate that we are simulating and provides very fine-grained data for the simulation. A data collection interval of 10ms means that Playback's data collection program sends a message on each link of topology every 10 milliseconds. This produces less fine-grained data, but is often more practical to deploy for long-running data collection experiments since it consumes less bandwidth and results in smaller log files. The modeling window used in the simulation (discussed in Chapter 4) is always set to include 10 packets; for the 1ms data collection interval this corresponds to a window of 10ms, while for the 10ms data collection interval this corresponds to a window of 100ms.

Because the two different data collection intervals are run as two different sets of experiments, there are also two corresponding sets of results for the real implementation in Spines, so any given simulation can be compared to what happened in the real implementation that was running at the same time and experiencing the same network conditions. However, the data collection interval of the simulator has no effect on Spines: any differences between the two sets of implementation data (typically denoted by "Spines (1ms)" and "Spines (10ms)") are simply due to random variations in the experienced network loss. We use NetEm to emulate the same link loss rates for both sets of experiments, but the exact sequence of lost packets is of

course not identical.

6.5.1.2 Results

The first experiment uses the single-edge topology shown in Figure 6.1a. This is the simplest possible overlay topology, consisting of only two overlay nodes with a single link between them. This allows us to evaluate the operation of the real-time hop-by-hop recovery protocol in Spines and compare it with Playback's modeling of loss and the recovery protocol without considering other factors, such as routing. Since there is only a single link, the routing protocol does not have any impact, so we simply use single-path routing in both Spines and the simulation. For this experiment, we apply loss rates ranging from 0-20% in both directions on the single edge (1,2) and measure the overall loss rate experienced by Spines and reported by the simulator. The results are shown in Table 6.1.

Applied Loss Rate on Edge (1,2)	**Expected Effective Loss Rate**	**Spines (1ms)**	**Playback (1ms)**	**Spines (10ms)**	**Playback (10ms)**
0%	0.0000%	0.0000%	0.0000%	0.0000%	0.0000%
2%	0.0792%	0.0804%	0.0796%	0.0765%	0.2223%
5%	0.4875%	0.4877%	0.4871%	0.4947%	0.8327%
10%	1.9000%	1.8955%	1.8953%	1.9100%	2.4941%
20%	7.2000%	7.1955%	7.2022%	7.1851%	8.1582%

Table 6.1: Effective loss rate under applied loss rates ranging from 0-20% for single-edge topology in Figure 6.1a.

For each applied loss rate, the expected effective loss rate using the real-time hop-by-hop recovery protocol is calculated as $p(p+p-p^2) = 2p^2-p^3$, where p is the applied loss rate. This is because for a packet to fail to make it to the destination, it must be lost on the first transmission attempt *and* either the retransmission request for that packet must be lost or the retransmission attempt must be lost. Table 6.1 shows that the loss rate experienced by Spines is in fact nearly identical to this expected loss rate, and with a 1ms data collection interval the loss rates reported by the Playback simulation are also nearly identical to Spines and the expected loss rates.

With a 10ms data collection interval, the Playback simulation tends to overestimate the actual loss rate slightly. This effect is due to correlation that the simulator introduces by averaging over the modeling window to determine the loss probability for any given packet: packets that fall in the same window or in largely overlapping windows will be subject to similar loss probabilities. Because an original transmission of a packet and a retransmission of that packet will be close to each other in time (separated by about one roundtrip time to allow for the request and retransmission),

they fall into similar modeling windows when a data collection interval of 10ms is used with a modeling window of 100ms (in the single edge topology, a roundtrip is 20ms). This added correlation reduces the effectiveness of the recovery protocol as modeled by the simulator when the underlying loss is uniform (as it is in our experiments with artificially applied loss rates).

For example, consider an underlying uniform loss rate of 2%. In any given 100ms window, there are 10 packets, so the loss rate may be 0%, 10%, 20%, ..., 100%. If a packet is lost, that means it fell in a window with at least one missing packet, so the average loss rate for that window is at least 10%. Since a recovery attempt will fall in a largely overlapping window, this makes it very likely that the retransmission also falls into a window with at least a 10% loss rate. Therefore, while on average the probability of a packet's original transmission being lost is only 2%, the probability of a retransmission of a lost packet also being lost is closer to 10% instead of 2%. In fact, if we calculate the expected effective loss rate accounting for this correlation, by letting the original probability of loss be 2% and the probability of a retransmission being lost be 10%, we get $0.02 \times (0.02 + 0.1 - (0.02 \times 0.1)) = 0.00236$, for an expected overall loss rate of 0.236%, instead of $0.02 \times (0.02 + 0.02 - 0.02^2) = 0.000792$ as shown in Table 6.1.[1] This expected loss rate of 0.236% is close to the 0.223% loss rate reported by Playback for the 10ms data collection interval with 2% applied loss in Table 6.1.

The size of the modeling window affects the correlation introduced by the simulator. Smaller windows add stronger correlation between packets, but over a shorter time window, while longer windows add weaker correlation with longer duration. Longer windows also have the effect of smearing the edges of complete disconnections, as packets will begin to be lost with some probability as soon as the beginning of the disconnection enters the window and will continue to be lost with some probability until the window has completely passed the end of the disconnection. While a window longer than 100ms would be able to more accurately model the uniform loss that we apply in these controlled experiments, we chose a window of 100ms as a reasonable compromise for real networks, where we do see disconnections and bursty loss.

Note that while the simulator does overestimate the loss rate in this case, this effect is consistent across all routing protocols, so it does not have a serious impact on the usefulness of the simulator as a tool for comparing protocols. Moreover, its effect on the overall loss is solely due to its reduction of the effectiveness of the recovery protocol. If the simulation is run without recoveries, its results are all within 0.1% of the expected value (specifically, 0%, 1.956%, 4.977%, 9.903%, and 19.936% for applied/expected loss rates of 0%, 2%, 5%, 10%, and 20%, respectively).

The second experiment is similar to the first, but instead of a topology consisting

[1]Note that the probability of a retransmission request being lost remains 2%, since the simulator does not introduce any correlation between loss in one direction of a link and loss in the opposite direction.

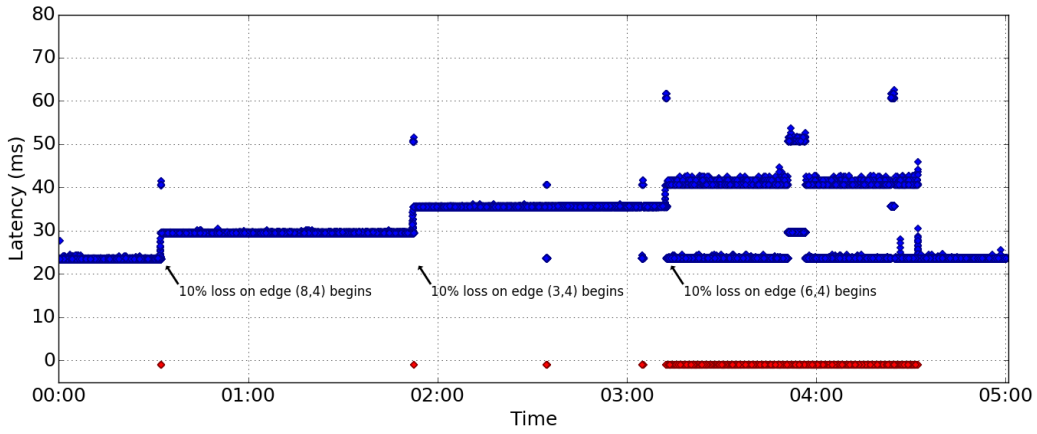
Applied Loss Rate on Edges (2,3) and (3,4)	Expected Effective Loss Rate	Spines (1ms)	Playback (1ms)	Spines (10ms)	Playback (10ms)
0%	0.000000%	0.0000%	0.0000%	0.0000%	0.0000%
2%	0.195232%	0.1975%	0.1950%	0.1933%	0.4830%
5%	1.176250%	1.1649%	1.1845%	1.1811%	1.8478%
10%	4.420000%	4.4391%	4.2140%	4.4011%	5.6306%
20%	15.52000%	15.4877%	15.5042%	15.5130%	17.0735%

Table 6.2: Effective rate of lost/late packets under applied loss rates ranging from 0-20% in both directions on edges (2,3) and (3,4) for the single-path topology in Figure 6.1b.

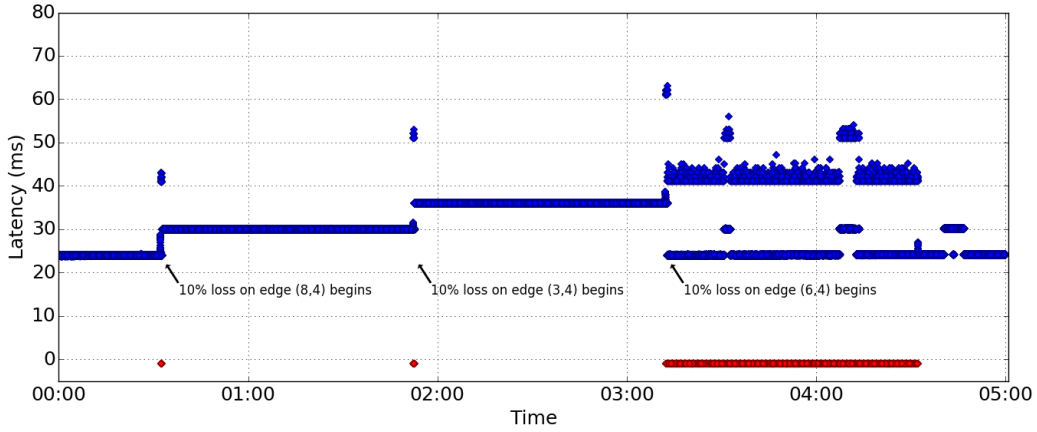
of only a single edge, it consists of a three-hop path between a source and destination. This topology is shown in Figure 6.1b and is referred to as the single-path topology. For this experiment, we applied loss in both directions to the last two links of the path, edges (2,3) and (3,4). Since packets can be dropped on either of these two links, the expected loss rate can be calculated as $2(2p^2 - p^3) - (2p^2 - p^3)^2$. However, with this topology, we also need to account for packets that can arrive late. With 10ms oneway latencies across each link, a packet that must be recovered on *both* edges (2,3) and (3,4) will take about 70ms to reach the destination node 4 (since each recovery adds about one roundtrip time on the link over which it is recovered, which for this topology is 20ms). Therefore, the total expected rate of lost or late packets is $2(2p^2 - p^3) - (2p^2 - p^3)^2 + (p(1-p)^2)^2$ for this experiment. The results in Table 6.2 show a similar pattern to those for the single-edge topology, with the rate of lost or late packets for Spines and for Playback with a 1ms data collection interval being almost identical to the expected loss rate and to one another. As before, the loss rate reported by Playback with a 10ms data collection interval shows a similar overall loss pattern to the Spines and 1ms Playback results, but slightly overestimates the absolute loss rate, particularly at lower levels of loss.

The third set of experiments evaluates basic routing capabilities by using a slightly more complex topology composed of three disjoint paths between the source (node 1) and destination (node 4), as shown in Figure 6.1c. For this experiment, we added loss to different edges in the topology one by one to trigger reroutes. Specifically, we start with no loss; after 30 seconds we add 10% loss to edge (8,4) (the last link of the shortest source-destination path in the topology); 80 seconds later we add 10% loss to edge (3,4) (the last link of the next shortest path); 80 seconds later we add 10% loss to edge (6,4) (the last link of the longest path); and 80 seconds later we remove all loss. In all cases, loss is added in both directions on each edge (i.e. to the edge and its reverse edge). Here we only consider basic single path routing (more sophisticated

routing protocols will be evaluated in the next set of experiments).



(a) Spines: 1555 packets lost or late.

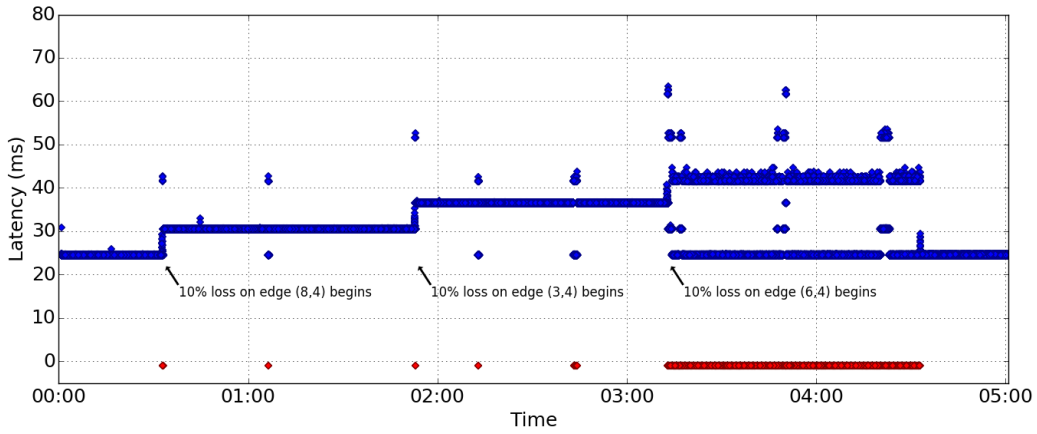


(b) Simulation (1ms data collection, random seed = 1): 1496 packets lost or late.

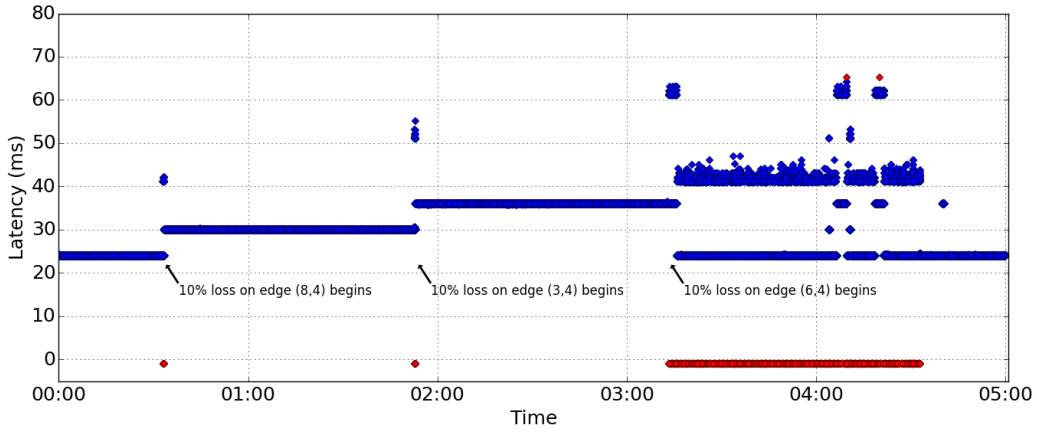
Figure 6.2: Single-path results for one experimental run with a 1ms data collection interval using the 3-path topology shown in Figure 6.1c, with 10% loss added (in both directions), to the link (8,4), then (3,4), then (6,4).

The results of one specific experimental run are illustrated in Figure 6.2 (results for the other four runs are similar). Each dot in the plots represents a packet, with the x-axis representing the time at which it was received and the y-axis representing its latency for reaching the destination. Dropped packets are shown in red at 0 on the y-axis. The protocol's rerouting can clearly be seen in the latency patterns represented in the plots. Initially, the lowest latency path (from node 1 to 7 to 8 to 4) is used. However, once 10% loss is applied to edge (8,4), the routing protocol takes that loss into account and reroutes to the next shortest path (from node 1 to 2 to 3 to 4). When loss is applied to edge (3,4), the protocol reroutes again, to the longest path (from node 1 to 5 to 6 to 4). Once loss is applied to edge (6,4), all three available

paths are equally lossy, so the protocol chooses to return to the lowest latency path – while it suffers loss, there is no way to avoid that under the given conditions.



(a) Spines: 1584 packets lost or late.



(b) Simulation (10ms data collection, random seed = 1): 1902 packets lost or late.

Figure 6.3: Single-path results for one experimental run with a 10ms data collection interval using the 3-path topology shown in Figure 6.1c, with 10% loss added (in both directions), to the link (8,4), then (3,4), then (6,4).

Recall that the expected latency metric used here attempts to optimize the probability of a packet arriving successfully at the destination by considering both loss and latency, and generally results in preferring paths with lower overall loss rates. While we apply equal loss rates on all three lossy edges in this experiment, random fluctuations in the loss pattern can cause the protocol to judge certain paths as more or less lossy at a particular time. This is the reason that we see the protocol move off of the shortest path for short periods during the time that all paths are experiencing 10% loss (e.g. a little before the 4:00 mark in Figure 6.2a). Also note that while the protocol is able to reroute quickly (in less than a second), it does suffer small amounts

of loss each time loss is applied to a new link before it can route away from that link: this shows up in the plots as dropped and recovered (higher latency) packets at each point that loss starts on a new link.

Figure 6.3 shows another experimental run with the same topology and loss pattern as in Figure 6.2, but with a 10ms data collection interval. The routing pattern shown matches the Spines implementation and the 1ms data collection results, although as expected, the overall loss rate for the simulation is slightly higher in this case.

Expected Lost/Late Packets	**Spines (1ms)**	**Playback (1ms)**	**Spines (10ms)**	**Playback (10ms)**
1520	1541.6 (SD=54.00)	1551.74 (SD=38.57)	1553.6 (SD=26.60)	2115.5 (SD=130.77)

Table 6.3: Average lost packets for 3 disjoint paths topology shown in Figure 6.1c out of 300,000 total packets.

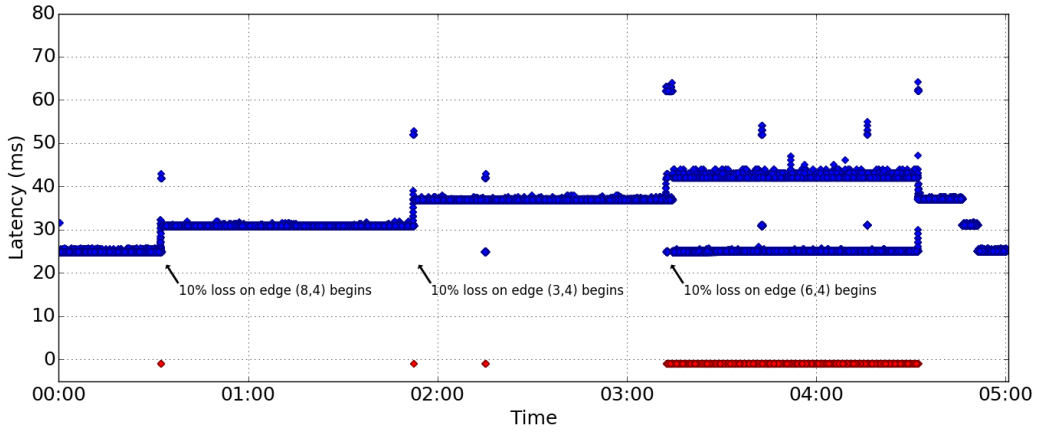
Table 6.3 shows the total number of lost or late packets averaged over 5 runs (as well as standard deviations) during this experiment in both the Spines implementation and the Playback simulation. Because of the ability to reroute, we only expect loss during the 80 seconds (out of 5 minutes) that all paths experience loss (discounting the minor loss that occurs prior to each reroute). During this time, we apply 10% loss, for an expected effective loss rate of 1.9%, or 1520 out of 80,000 packets.

Next, we evaluate the full implementation of our dissemination-graph-based transport service, comparing it to single-path routing and two-disjoint-paths routing. To do this, we use the overlay topology shown in Figure 6.1d. This graph consists of the same three disjoint paths as in Figure 6.1c, but with two additional edges, allowing for dissemination graphs that do not consist of only disjoint paths.

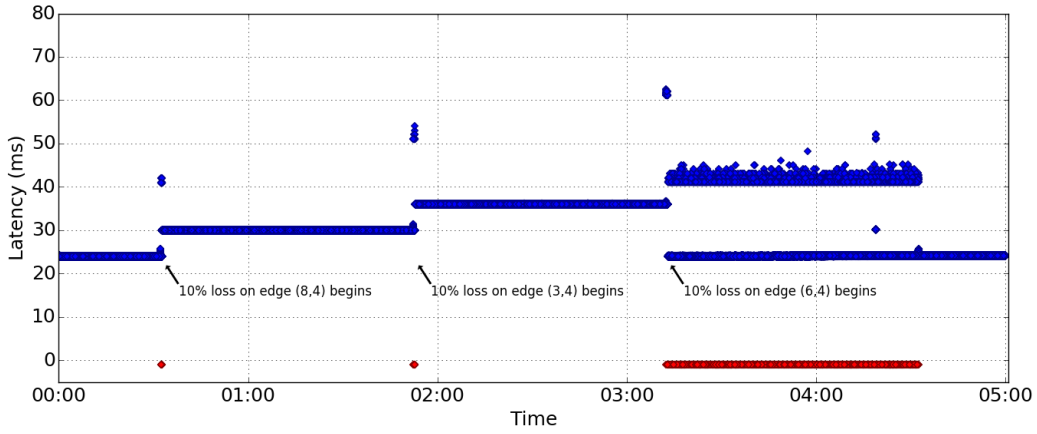
For this set of experiments, we apply loss in the same pattern as in the experiments using the topology of Figure 6.1c (10% loss on edge (8,4), then (3,4), then (6,4)), but using the extended topology and repeating the experiment with two-disjoint-paths routing and our full transport service in addition to single-path routing.

The results for a single experimental run using single-path routing are shown in Figure 6.4. These results are very similar to the results for the 3-disjoint-paths topology described above; the addition of the two extra edges does not provide any alternative paths of lower latency or loss for the specific scenario we consider, so the routing pattern is essentially the same, resulting in very similar loss rates and delivery latencies.

The results for a single experimental run using two-disjoint-paths routing are shown in Figure 6.5. This figure clearly illustrates the benefits of using two-disjoint paths instead of a single path. First, because two paths are always being used simulta-



(a) Spines: 1507 packets lost or late.



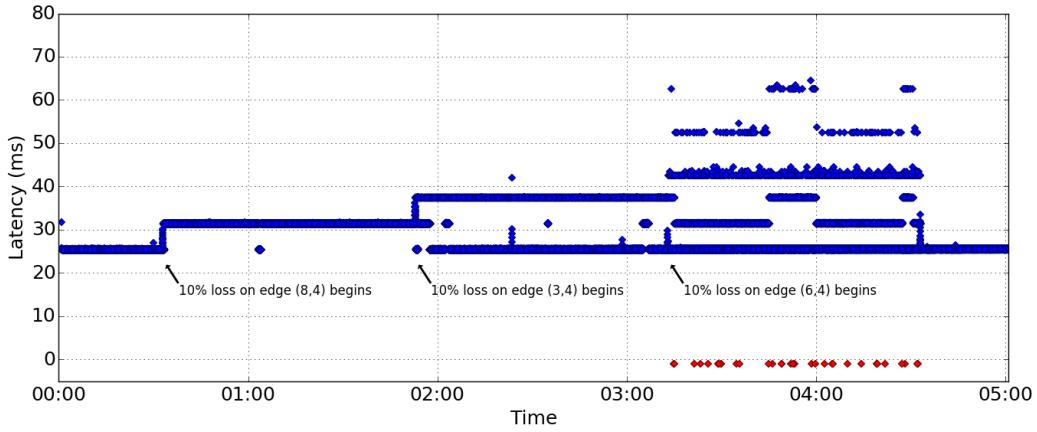
(b) Simulation (1ms data collection, random seed = 1): 1500 packets lost or late.

Figure 6.4: Single-path results for topology shown in Figure 6.1d, with 10% loss added (in both directions), to the link (8,4), then (3,4), then (6,4).

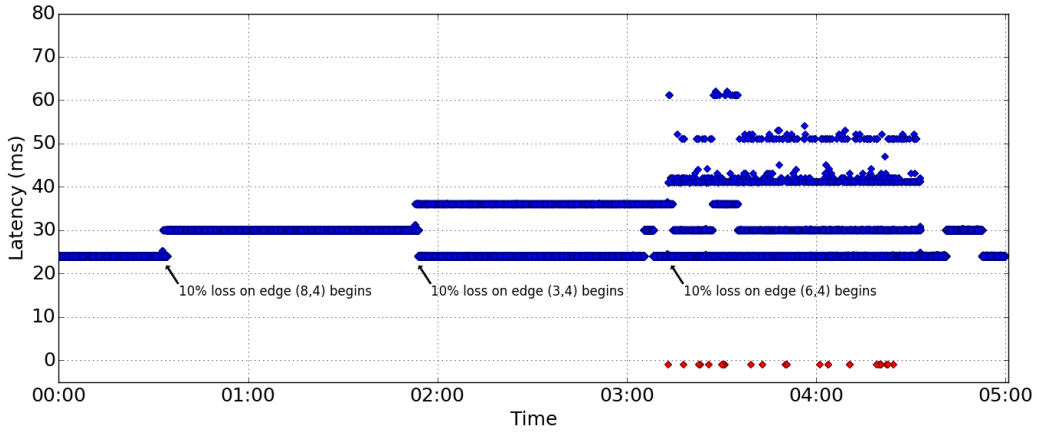
neously, this approach does not experience any loss during the time it takes to detect that a link has become lossy and reroute: the second path is able to deliver packets that would be lost in a single-path approach.[2] In addition, during the time that all paths are experiencing loss, the simultaneous use of two paths considerably lowers the loss rate by providing more opportunities for packets to arrive at the destination on time: for a packet to fail to reach the destination, both its original transmission and its recovery attempt must fail on *both* of the two paths in use.

Finally, results for a single experimental run using our dissemination-graph-based routing protocol with targeted redundancy are shown in Figure 6.6. These plots

[2]Note that this is true because we add loss to the links one at a time; if both of the two paths being used simultaneously started to experience loss or became disconnected, a two-path approach would incur loss. However, this is a rarer scenario.



(a) Spines: 32 packets lost or late.

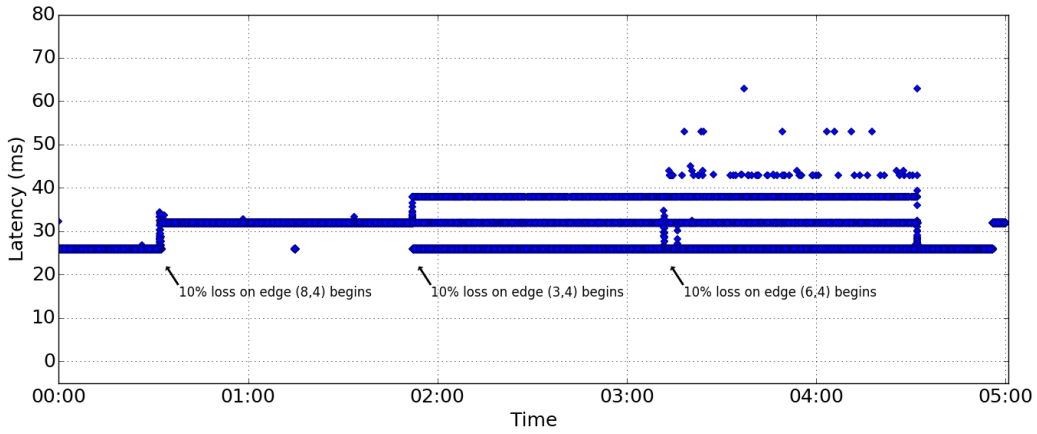


(b) Simulation (1ms data collection, random seed = 1): 24 packets lost or late.

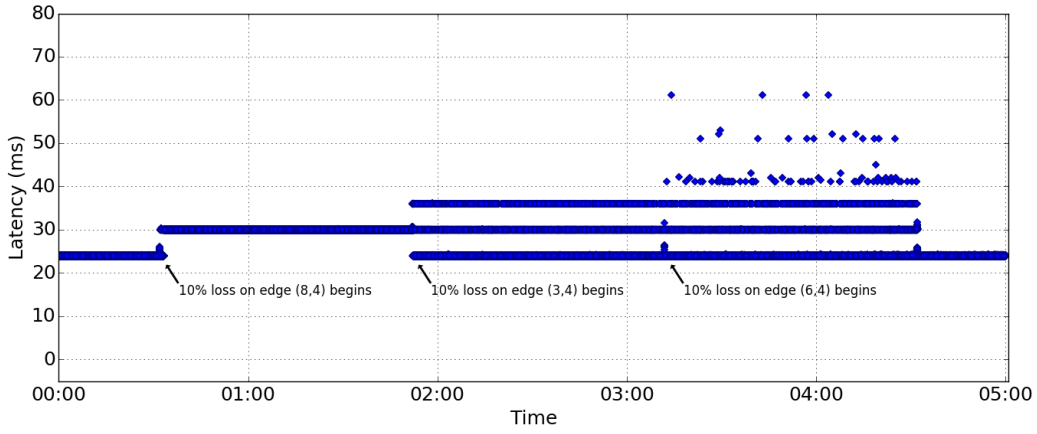
Figure 6.5: Two-disjoint-paths results for topology shown in Figure 6.1d, with 10% loss added (in both directions), to the link (8,4), then (3,4), then (6,4).

illustrate the benefit of employing targeted redundancy compared with using two paths. Because we use two disjoint paths in the normal case, no loss is incurred during the time it takes for newly begun loss to be detected (just like in the two-disjoint-paths approach). The main benefit of our approach can be seen during the time that all paths are experiencing loss. Once the first two paths begin to experience loss on their last hop (i.e. the link entering the destination), the situation is classified as a destination problem, so the protocol begins to make use of all useful incoming links for the destination. In this case, this provides three entry points instead of only two, and for the specific example run shown in Figure 6.6, results in all packets being delivered on time in the Spines implementation (Figure 6.6a), which also matches the Playback simulation for that time period (Figure 6.6b). Note that the protocol does not guarantee 100% delivery in this case but does reduce the loss rate to a nearly

negligible level: across all experimental runs we see 0-2 packets per run that are not delivered on time in the Spines implementation.



(a) Spines: 0 packets lost or late.



(b) Simulation (1ms data collection, random seed = 1): 0 packets lost or late.

Figure 6.6: Dissemination-graph results for topology shown in Figure 6.1d, with 10% loss added (in both directions), to the link (8,4), then (3,4), then (6,4).

The above plots for specific experimental runs illustrate the behavior and benefits of the three routing approaches qualitatively. Table 6.4 quantifies the effect of the protocols by reporting the average number of lost or late packets across all 5 runs for each experiment type (i.e. 1ms vs 10ms data collection interval) and all 10 random seeds for the simulation. As in the experiments with the 3-disjoint-paths topology (Figure 6.1c), we only expect loss during the 80 seconds that there is no loss-free path available (minus small hits during detection for the single-path approach). For approaches using multiple incoming links to the destination, a packet only fails to reach the destination if it is lost on *all* of the available incoming links. Therefore the loss rates for the single-path, two-disjoint-paths, and dissemination-graph approaches

in this case are calculated as $2p^2 - p^3$, $(2p^2 - p^3)^2$, and $(2p^2 - p^3)^3$, respectively. The expected and actual numbers of lost or late packets for all three approaches are shown in Table 6.4. This table shows the benefit that can be obtained by increasing the level of redundancy: using two disjoint paths reduces the number of lost packets by over 50 times compared with a single-path approach, and our dissemination graphs further reduce the number by about 50 times. Note that while the Playback results with a 10ms data collection interval continue to slightly overestimate the absolute loss rate, this overestimation occurs consistently across all protocols, with the relative pattern of improvement across the protocols remaining very similar. Therefore, while a 1ms data collection interval is recommended to achieve as accurate a simulation as possible, a 10ms interval (which is often more practical) is very good for judging differences between protocols and provides a reasonably accurate but somewhat pessimistic view of overall loss.

Routing	**Expected Lost/Late Packets**	**Spines (1ms)**	**Playback (1ms)**	**Spines (10ms)**	**Playback (10ms)**
Single Path	1520	1527 (SD=44.90)	1535.62 (SD=38.81)	1545.2 (SD=15.04)	2110.3 (SD=57.91)
Two Disjoint Paths	28.88	29.2 (SD=1.72)	29.48 (SD=4.61)	24.4 (SD=1.62)	48.0 (SD=7.00)
Dissemination Graphs	0.54872	0.6 (SD=0.80)	0.42 (SD=0.60)	0.6 (SD=0.49)	1.36 (SD=1.14)

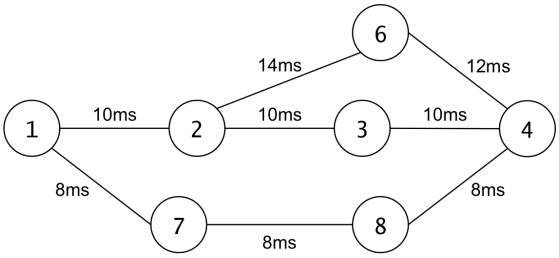
Table 6.4: Average number of lost or late packets for topology shown in Figure 6.1d out of 300,000 total packets.

Routing	**Spines (1ms)**		**Playback (1ms)**		**Spines (10ms)**		**Playback (10ms)**	
	Dollar	Goodput	Dollar	Goodput	Dollar	Goodput	Dollar	Goodput
Single Path	3.0242	3.0397	3.0243	3.0398	3.0242	3.0398	3.0246	3.0460
Two Disjoint Paths	6.0722	6.0727	6.0727	6.0733	6.0722	6.0727	6.0727	6.0736
Dissemination Graphs	7.3145	7.3145	6.7141	6.7141	7.3029	7.3029	6.7074	6.7074

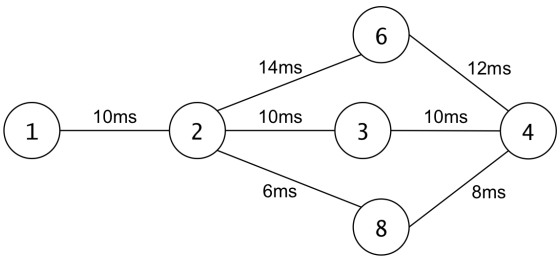
Table 6.5: Average dollar cost (total packets sent / packets introduced) and goodput cost (packets sent / packets delivered on time) for the topology in Figure 6.1d.

In addition to reliability, cost-effectiveness is a key consideration for our protocols. Table 6.5 shows the average cost for the three different routing approaches in each set of experiments with the topology in Figure 6.1d. Using two disjoint paths costs about twice as much as a single path (2.0079 times as much in terms of dollar cost on average for the "Spines (1ms)" runs). Using dissemination graphs with targeted redundancy costs about 20.5% more than using two disjoint paths in Spines, although the cost reported by the Playback simulation is only about 10.5% more than the cost

for two disjoint paths. In this case, the difference in cost between using two disjoint paths and dissemination graphs with targeted redundancy is higher than reported in the evaluation over four weeks of data reported in Section 5.4. This is simply because we are focusing on a short period with significant network problems for this controlled evaluation: the destination-problem dissemination graph is used for over half of the five-minute period examined. In practice, network problems are less frequent, so the cost of the more expensive dissemination graphs is amortized over much longer periods of time.



(a) Destination-problem graph used in Spines (8 edges).



(b) Destination-problem graph used in Playback Simulation (7 edges).

Figure 6.7: Destination-problem dissemination graphs used in Spines and simulation for topology shown in Figure 6.1d.

The reason for the discrepancy in cost reported by Spines and by the simulation is that the destination-problem graph used by Spines is slightly different than the one used in the simulation. In the simulation, we use shallow-light Steiner trees to construct the destination-problem graphs, which connect all the neighbors of the destination to the source in a tree that minimizes the number of edges while meeting the latency requirements. However, in the Spines implementation, we choose to optimize for latency over cost and use shortest-path trees. Because of this, the simulation uses the lowest cost destination-problem graph shown in Figure 6.7b with seven edges, while Spines uses the eight-edge graph shown in Figure 6.7a. Note that because of the way the latencies are set up in this topology, the lowest cost graph in Figure 6.7b is actually a valid shortest-path tree as well (the latency from the source node 1 to each of the neighbor destination nodes 3, 6, and 8 is the same in both Figure 6.7b and Figure 6.7a). Therefore, it would be valid for Spines to use this graph as well, in which case it would achieve a cost similar to that reported by the simulation; however, because of the order in which Spines processes edges when constructing its shortest-path tree, it selects a different valid (though more expensive) solution. A future optimization would be to allow Spines to select the least expensive graph among valid shortest-path trees.

6.5.2 Evaluation via Case Studies

The controlled evaluation in the previous section allows us to closely examine the factors affecting the reliability and cost of single-path routing, two-disjoint-paths routing, and our full dissemination-graph-based transport service, as implemented in Spines, and as modeled by the simulator on simple overlay topologies designed specifically to evaluate different aspects of the protocols. In this section, we evaluate our implementation of the dissemination-graph-based transport service in Spines using a real practical overlay topology and realistic network conditions.

Specifically, we use the same overlay topology as in our data collection and evaluation via simulation in Section 5.4 (Figure 4.1), and evaluate the implementation's performance on case studies drawn from network problems that we observed during the four weeks of data collection from Section 5.4. For each case study, we emulate the network conditions that were observed at a specific point in time.

First, to give us full control over the latencies and loss rates, these experiments are conducted on a cluster of twelve computers on a local area network, using NetEm to set latencies and loss rates to match those observed in the collected data. Disconnections are emulated by applying 100% loss to the relevant links. As in the controlled experiments in Section 6.5.1, each case study is run in two different conditions, one with a data collection interval of 1ms for the simulation and one with a data collection interval of 10ms. For each condition, the case study is run five times and the results are reported as averages. In these case studies, the three routing protocols being evaluated (single-path, two disjoint paths, and dissemination graphs with targeted redundancy) are always run in parallel, so we can compare the performance that each provides at the same time and under exactly the same conditions.

In addition to the local-area evaluation with emulated latencies and loss rates, we also evaluate each case study on a real global overlay network. In this setting, we do not emulate latency, as the overlay nodes are distributed across North America, Europe, and Asia (in the same topology as in the local-area emulation) and have real network latency between them. However, to recreate the scenario for each case study, we emulate loss using the *setlink* tool in Spines (since we do not have permissions to use NetEm to modify machine-level settings). Because we are only able to apply loss to Spines traffic, rather than all traffic, we do not run the simulator's data collection in this setting. Otherwise, the procedure is the same as in the local-area evaluation.

6.5.2.1 August 15, 2016 Case Study

The first case study is inspired by an event that occurred on August 15, 2016, in which all of the Los Angeles node's incoming links were experiencing varying degrees of loss over a period of a little less than two minutes. For this case study, we focus on the flow from Atlanta, Georgia to Los Angeles, California at that time. To recreate a similar scenario for our case study, we analyzed the raw log data from that time

Routing	Spines (1ms)	Playback (1ms)	Spines (10ms)	Playback (10ms)	Spines Wide-Area
Single Path	7145.60 (SD=137.72)	7066.25 (SD=111.80)	7495.60 (SD=789.92)	11677.80 (SD=1306.00)	7051.80 (SD=108.23)
Two Disjoint Paths	1047.2 (SD=210.42)	1005.75 (SD=58.71)	1036.40 (SD=203.57)	1753.6 (SD=122.63)	866.00 (SD=93.03)
Dissemination Graphs	194.0 (SD=16.52)	179.75 (SD=14.13)	196.20 (SD=15.30)	229.20 (SD=20.73)	125.00 (SD=16.59)

Table 6.6: Average lost or late packets for August 15, 2016 case study (out of 140,000 packets).

period, calculating the average loss rate and latency on each of these incoming links for every 10-second interval in that period.

To run the experiment in the local-area cluster setting, we first set the latency of all of the edges in the overlay topology to match their latency in the real wide-area topology (using NetEm). After allowing the flow to run for 10 seconds, we begin applying the calculated loss rates and latencies from the analyzed data, updating the values for each link every ten seconds. The loss rates varied between 45.2% and 68.3% for the Washington DC to Los Angeles link, 21.6% to 35.5% for the San Jose to Los Angeles link, 18.9% to 30.1% for the Dallas to Los Angeles link, 18.7% to 32.2% for the Hong Kong to Los Angeles link, and 25.4% to 38% for the Denver to Los Angeles link. None of the links experienced significantly elevated latency during this time.

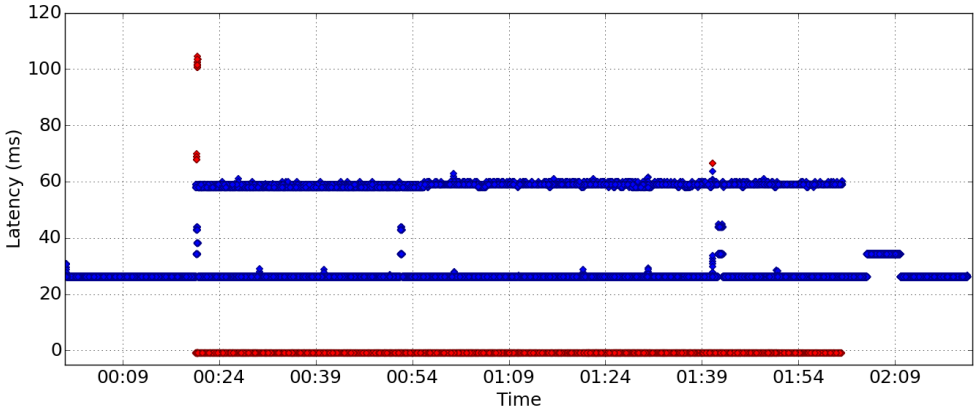
Table 6.6 shows the average number of packets lost (out of 140,000 total packets) for each routing approach. The results for Spines (1ms) show that using two disjoint paths reduces the loss rate for this scenario from about 5.1% with a single-path approach to about 0.75%. Using dissemination graphs with targeted redundancy further reduces the loss rate to 0.14%. From this, we see that our implemented dissemination-graph-based transport service is able to support good performance (99.86% on-time delivery), even during a severe network problem (with single-link loss rates of 18-68%) and significantly improves performance compared to simpler routing approaches using the same infrastructure.

Figure 6.8 shows the latency for each packet sent during one experimental run of this case study using each of the three routing protocols we evaluate in Spines. Based on the latencies shown in these plots, we can see that in general, the single-path approach uses the shortest path from Atlanta to Dallas to Los Angeles (about 26ms latency). The two-disjoint-paths approach uses that path plus the longer path from Atlanta to Denver to San Jose to Los Angeles (about 34ms), although it briefly shifts to use a path from Atlanta to Denver directly to Los Angeles, which has a similar latency overall but does not allow successful recoveries on the Denver - Los Angeles link (which is longer than the San Jose - Los Angeles link). The dissemination graph approach makes use of all incoming links to the Los Angeles node, providing more

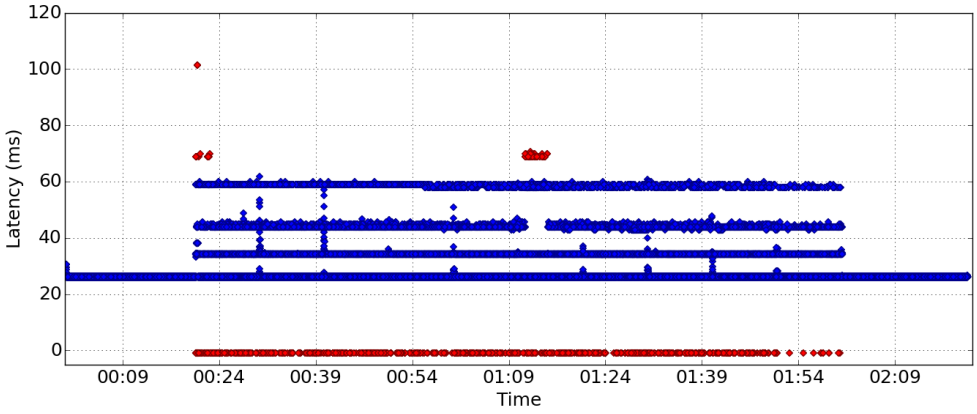
opportunities for packets to arrive on time and to be recovered successfully.

Figure 6.9 shows the results of the Playback simulator using the data collected at the same time as the run shown in Figure 6.8 (with a 1ms data collection interval), which display a similar overall pattern.

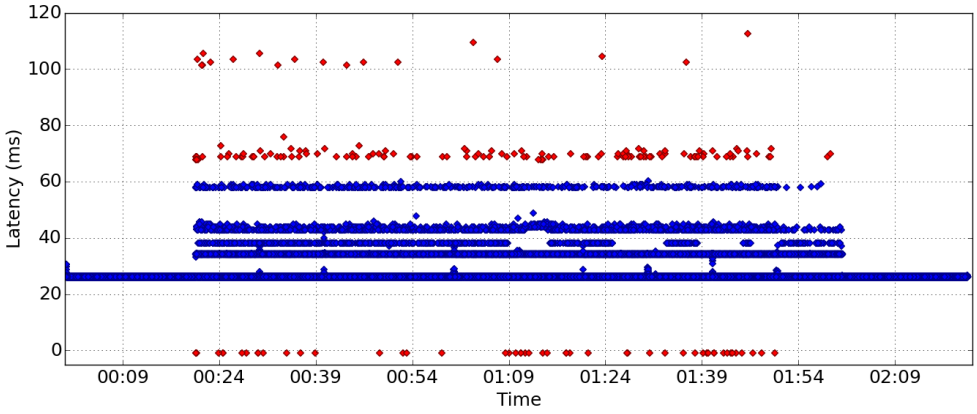
Figure 6.10 shows the results of the same case study run on the global wide-area network. Overall, the performance closely matches the results of the local-area emulation. Comparing the "Spines Wide-Area" column of Table 6.6 to the "Spines (1ms)" and "Spines (10ms)" columns shows that the number of lost packets is very similar, and Figure 6.10 shows that the overall routing pattern also matches well (although routing varies somewhat across the five runs for both the local-area and wide-area experiments). However, the fact that we do not fully control the latency of each link in the wide-area setting leads to some small differences between the two cases: for example, most recoveries on the Denver - Los Angeles link are able to reach the destination on time in the wide-area setting (during the brief reroute around the 00:54 mark for the two-disjoint-paths approach), although they arrived just after the 65ms latency deadline in the local-area emulated setting.



(a) Spines 1 path: 7064 packets lost or late.

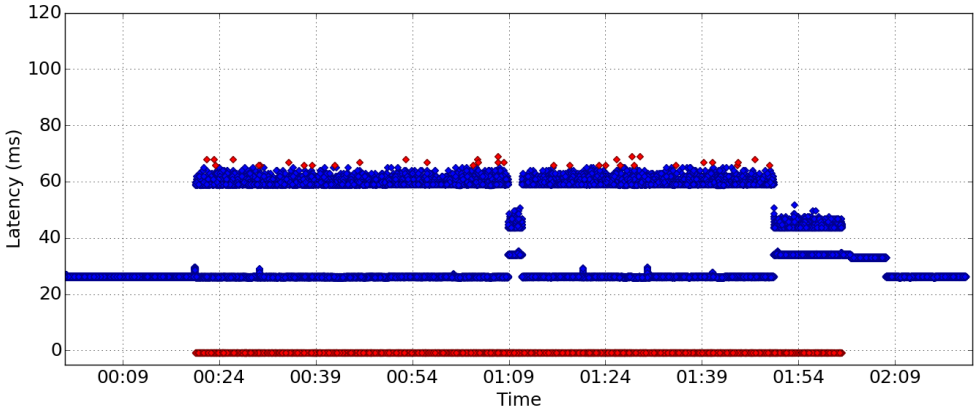


(b) Spines 2 paths: 785 packets lost or late.

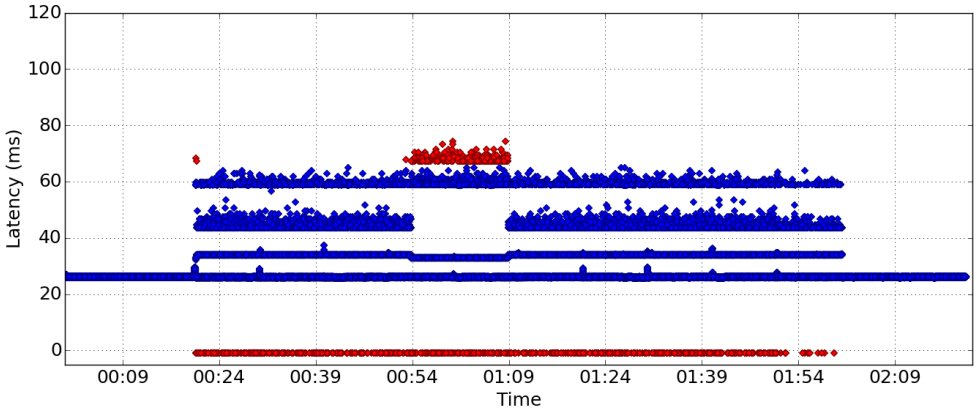


(c) Spines Dissemination Graphs: 195 packets lost or late.

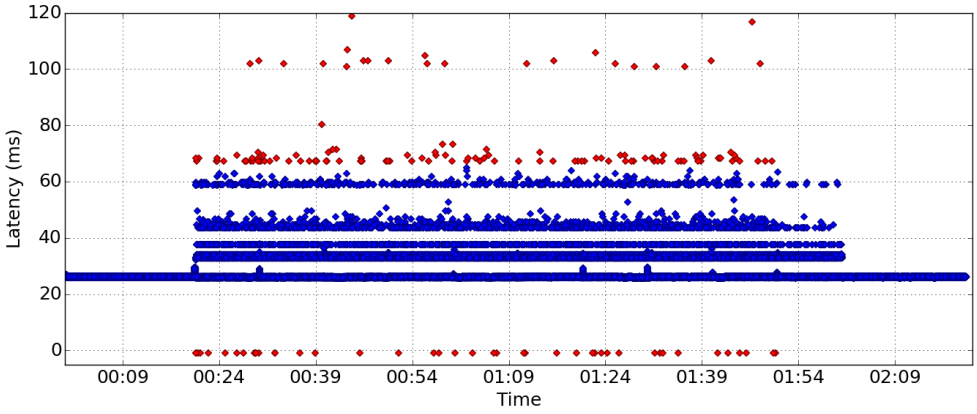
Figure 6.8: Spines results for case-study inspired by event on August 15, 2016, with loss on all of Los Angeles node's incoming links for flow from Atlanta to Los Angeles. Emulated loss and latency in local-area cluster environment.



(a) Simulation 1 path: 7154 packets lost or late.

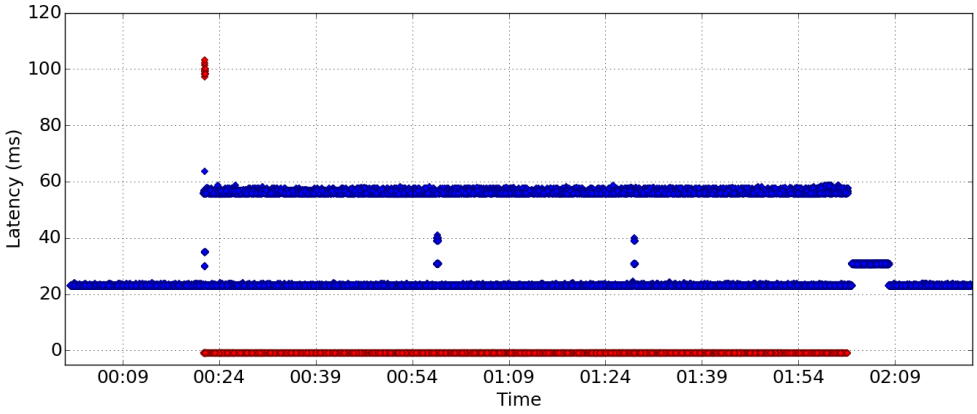


(b) Simulation 2 paths: 976 packets lost or late.

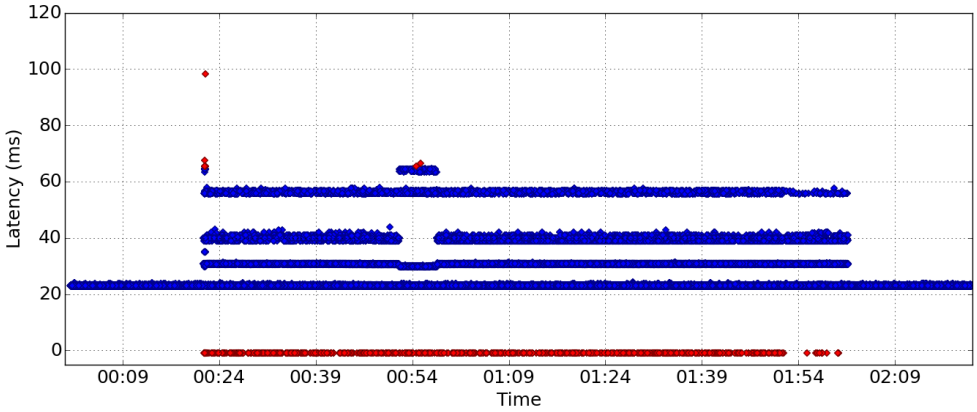


(c) Simulation Dissemination Graphs: 170 packets lost or late.

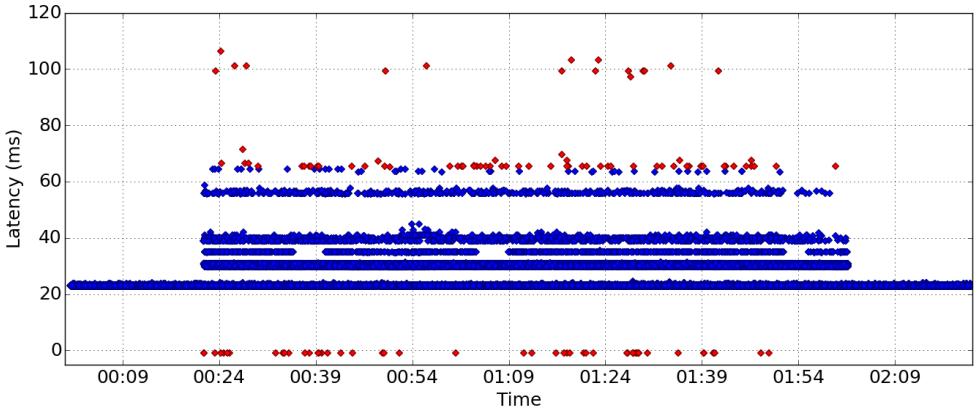
Figure 6.9: Playback simulation results for case-study inspired by event on August 15, 2016, with loss on all of Los Angeles node's incoming links for flow from Atlanta to Los Angeles. 1ms data collection interval.



(a) Spines 1 path: 6983 packets lost or late.



(b) Spines 2 paths: 735 packets lost or late.



(c) Spines Dissemination Graphs: 124 packets lost or late.

Figure 6.10: Spines results for case-study inspired by event on August 15, 2016, with loss on all of Los Angeles node's incoming links for flow from Atlanta, Georgia to Los Angeles, California. Real latencies and emulated loss rates in a global wide-area environment.

6.5.2.2 October 17, 2016 Case Study

The second case study is inspired by an event that occurred on October 17, 2016. Similarly to the previous case study, all of the Los Angeles node's incoming links were experiencing loss for a period of about one minute. For this case study, we focus on the flow from New York to Los Angeles at that time. As in the previous case study, we recreate the scenario (to the degree possible without being able to reproduce the exact loss characteristics, e.g. burstiness) by averaging the loss rate and latency over each 10-second interval and updating the applied loss rates and latencies via NetEm every 10 seconds in the local-area setting, and updating the applied loss rates via Spines setlink every 10 seconds in the wide-area setting.

In this case, the loss rates varied from 25-32.2% on the Washington DC to Los Angeles link, 24.6-30.2% on the San Jose to Los Angeles link, 22.8-28.6% on the Dallas to Los Angeles link, 32.9-44.7% on the Hong Kong to Los Angeles link, and 50.8-56.9% on the Denver to Los Angeles link. None of the link latencies were significantly higher than normal at this time.

Table 6.7 shows the average number of packets lost for each routing protocol (and experimental condition). From this table, we see that our dissemination-graph-based transport service is able to reduce the loss rate to about 0.18%, while two disjoint paths experience about 1.18% loss and a single-path approach experiences over 10% loss (these numbers are based on the Spines (1ms) column).

Routing	Spines (1ms)	Playback (1ms)	Spines (10ms)	Playback (10ms)	Spines Wide-Area
Single Path	10464.60 (SD=1950.92)	11065.75 (SD=238.72)	11568.60 (SD=2732.70)	12292.60 (SD=2852.62)	10964.40 (SD=3645.28)
Two Disjoint Paths	1182.60 (SD=101.72)	1138.00 (SD=90.37)	1949.00 (SD=858.10)	2504.00 (SD=813.86)	1655.60 (SD=599.79)
Dissemination Graphs	179.20 (SD=9.41)	158.50 (SD=6.89)	175.40 (SD=10.59)	218.00 (SD=9.86)	174.40 (SD=10.33)

Table 6.7: Average lost or late packets for October 17, 2016 case study (out of 100,000 total packets).

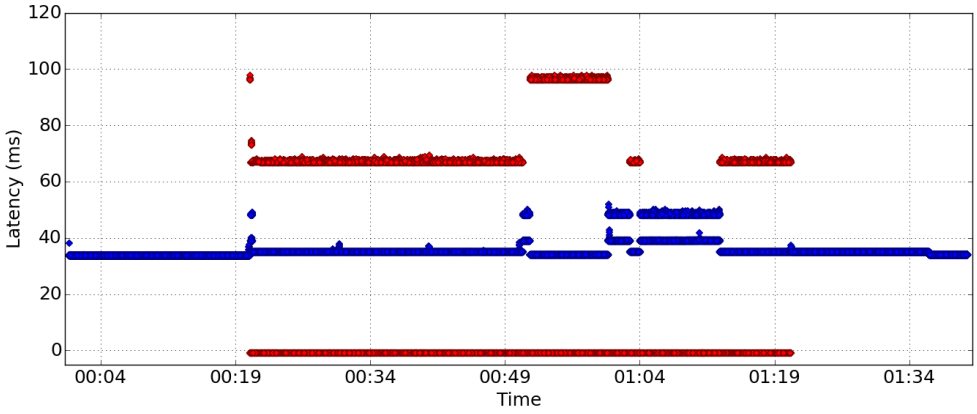
Figure 6.11 shows the per-packet delivery latencies for one of the five experimental runs for this case study for each of the three routing protocols evaluated in Spines. Using single-path routing, initially, before any loss is applied, Spines uses the lowest-latency path from New York to Washington DC to Los Angeles (about 34ms). However, once loss begins it reroutes to the slightly longer (about 35.5ms) path that uses Dallas as the last hop before Los Angeles, as that path provides lower expected latency under the applied loss conditions. A little after the 00:49 mark, as loss rates fluctuate, it briefly reroutes back to the path through Washington, and then to a slightly longer path (about 39ms) through San Jose, before going back to the 35.5ms path through Dallas and finally returning to the 34ms path through Washington after the loss ends. Interestingly, although the path through San Jose path is longer than

the others, it actually provides the best performance in this particular case, as it allows for recovery of lost packets within the time constraint (due to the fact that the last hop from San Jose to Los Angeles is shorter than the other options).

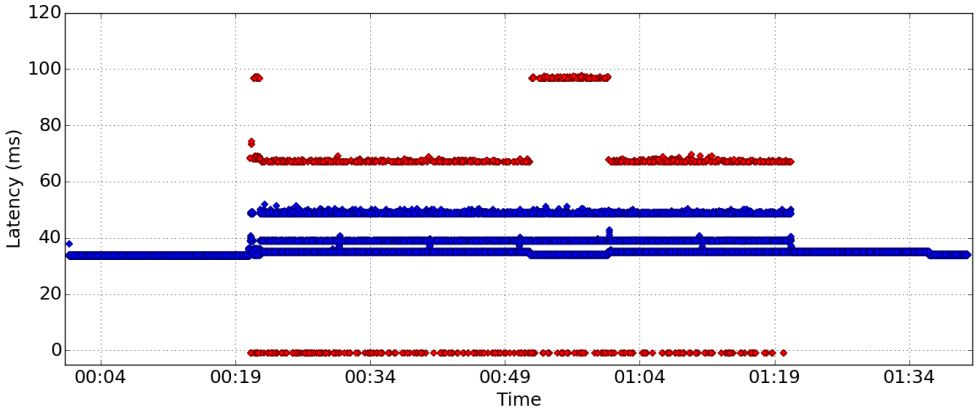
The two-disjoint-paths approach fairly consistently chooses paths through San Jose and Dallas, although it briefly trades the path using Dallas for the New York - Washington - Los Angeles path (between 00:49 and 01:04). The dissemination-graph-based transport service with targeted redundancy reroutes to a destination-problem dissemination graph as soon as it detects loss on two of the Los Angeles node's incoming edges, making use of all available entry points to the Los Angeles node and substantially reducing the overall loss rate.

The results for Playback's simulation using data collected at 1ms intervals during the same time period as the Spines run in Figure 6.11 are shown in Figure 6.12. The overall pattern of these results is quite similar, although reroutes do not line up exactly, largely due to random variation in how the two programs experience the same applied loss rate: although the data collection and Spines implementation are running in parallel and experiencing the same conditions, they will not see exactly the same pattern of lost packets, which can cause their routing calculations to differ somewhat.

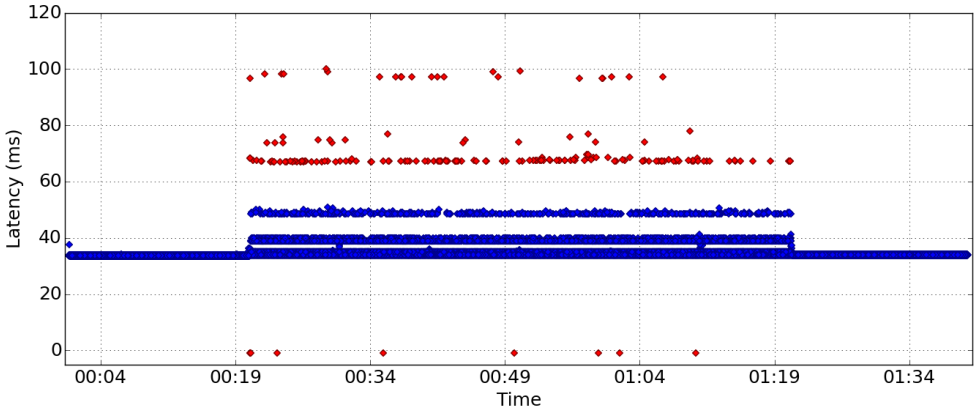
Finally, the results for one run of the same case study on the global overlay network are shown in Figure 6.13. Both the average numbers of lost packets and the overall routing patterns for each approach are very similar to the local-area Spines and simulation results, further validating the usefulness of our dissemination-graph-based approach in successfully addressing this type of network problem.



(a) Spines 1 path: 13034 packets lost or late.

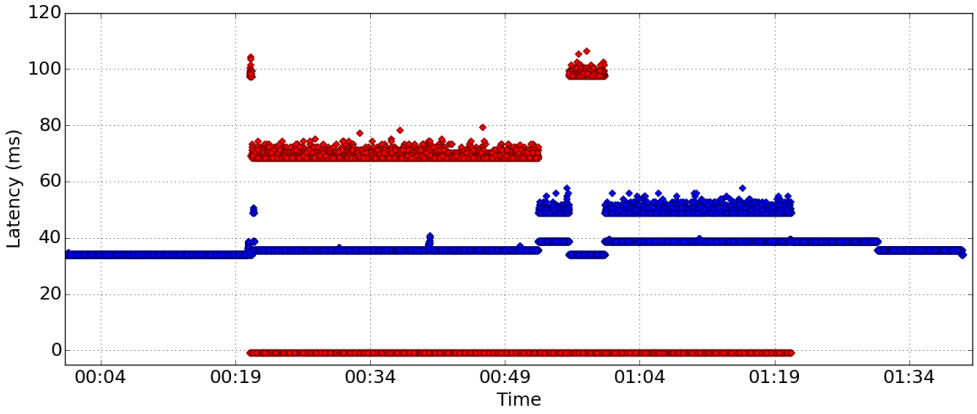


(b) Spines 2 paths: 1072 packets lost or late.

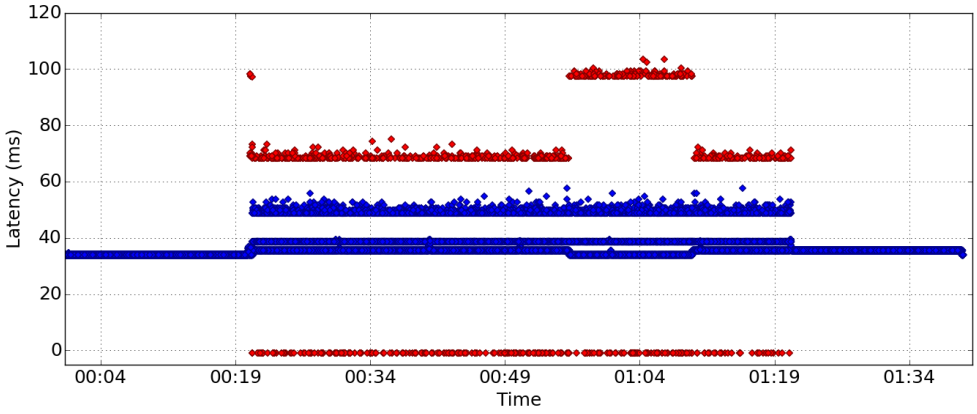


(c) Spines Dissemination Graphs: 186 packets lost or late.

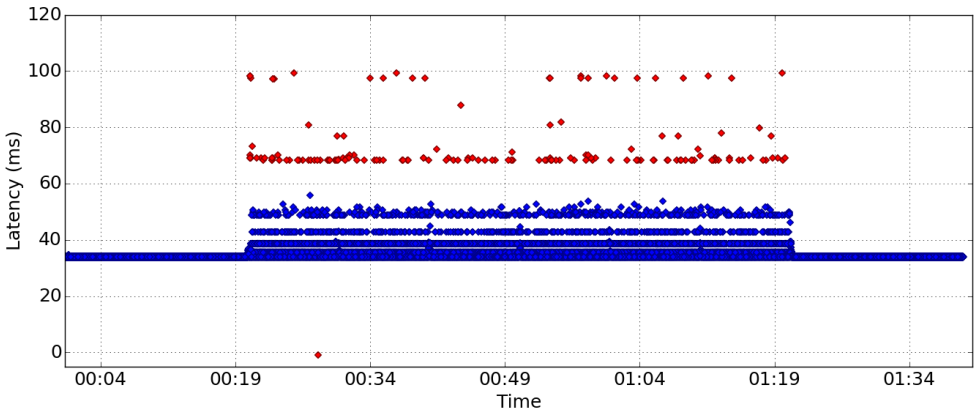
Figure 6.11: Spines results for case-study inspired by event on October 17, 2016, with loss on all of Los Angeles node's incoming links for flow from New York to Los Angeles. Emulated latencies and loss rates in local-area cluster environment.



(a) Simulation 1 path: 11017 packets lost or late.

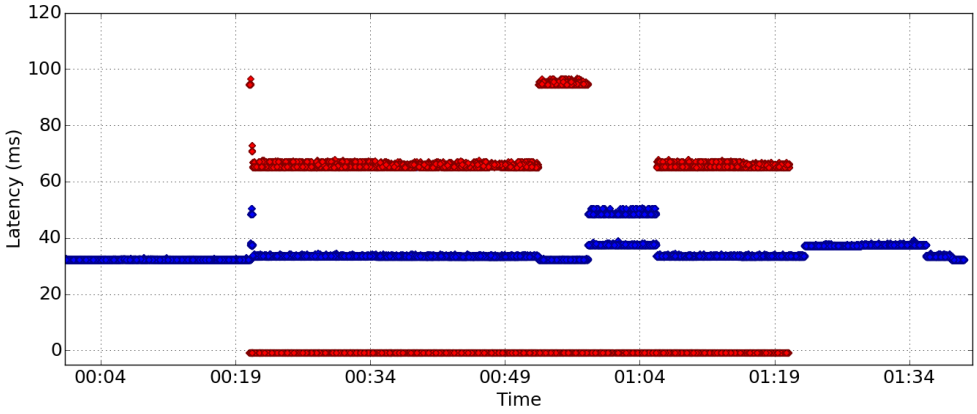


(b) Simulation 2 paths: 1109 packets lost or late.

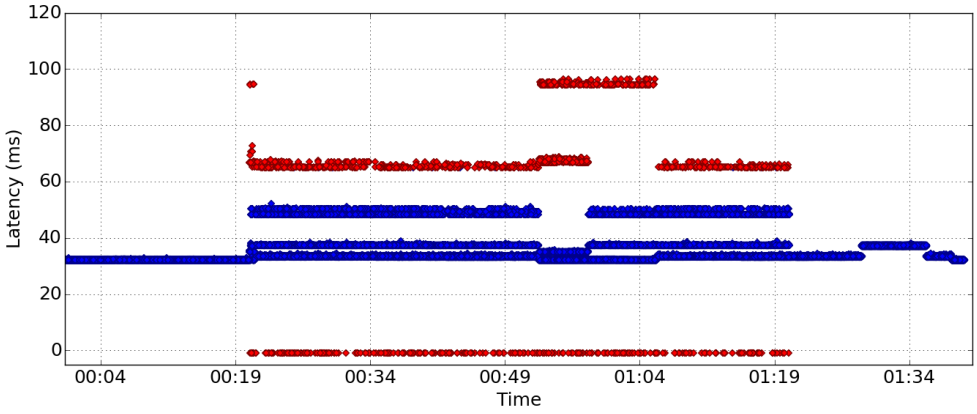


(c) Simulation Dissemination Graphs: 157 packets lost or late.

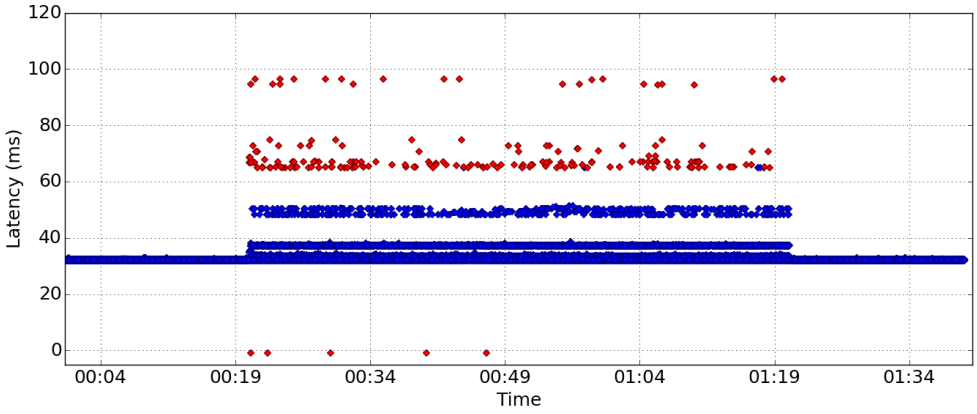
Figure 6.12: Playback simulation results for case-study inspired by event on October 17, 2016, with loss on all of Los Angeles node's incoming links for flow from New York to Los Angeles. 1ms data collection interval.



(a) Spines 1 path: 13591 packets lost or late.



(b) Spines 2 paths: 1384 packets lost or late.



(c) Spines Dissemination Graphs: 192 packets lost or late.

Figure 6.13: Spines results for case-study inspired by event on October 17, 2016, with loss on all of Los Angeles node's incoming links for flow from New York to Los Angeles. Real latencies and emulated loss rates in global wide-area environment.

6.5.2.3 September 8, 2016 Case Study

The third case study investigates a different type of problem than the previous two. Instead of random loss, this case study analyzes a complete disconnection of several links, inspired by an event that occurred on September 8, 2016. Here we consider the flow from Los Angeles to New York, at a time when several of the New York node's links experienced disconnections.

The pattern of disconnection events can be seen in Figure 6.14 (for Spines in the emulated, local-area setting), Figure 6.15 (for the corresponding simulation), and Figure 6.16 (for Spines in the real wide-area setting). Initially, the single-path approach uses the shortest path, from Los Angeles, to Washington, to New York, and both the two-paths approach and dissemination-graph-based approach use that same path plus a second path from Los Angeles to Dallas to Atlanta to Baltimore to New York. The first event, at about 00:15 is a simultaneous disconnection of both the Washington - New York link and the Baltimore - New York link. Because these are the paths that are actively being used in all three approaches, they all experience a short outage. This outage shows up as a series of dropped (red) packets, lasting about 1 second in Spines and 500ms in the simulation. This brief outage represents the time needed to detect the disconnection and reroute to a new dissemination graph. Spines has a somewhat heavier link creation and destruction process that results in it being more conservative than what is modeled in the simulator, leading to the longer outage. Once the disconnections are detected, all protocols are able to reroute, making use of alternative paths that continue to work and experiencing no loss for the remainder of the 2-second disconnection.

Notice that the dissemination-graph-based approach is able to resume taking advantage of the lower latency on the shortest path as soon as the disconnection ends, as it uses a destination-problem graph that includes that path as well as alternative ways into the destination during that period. In contrast, the single-path and two-disjoint-paths approaches continue to use the longer path. This is due to the design choice of not allowing link weights to be lowered for at least 30 seconds after a weight increase is detected. In general, this is a very useful practice, as it makes routing more stable and helps avoid switching back to problematic links before the problem has fully resolved. However, in rare cases, it can have negative consequences: one such example is seen in the next event just after 00:54.

In the event around 00:54, the Washington - New York and Baltimore - New York links are again simultaneously disconnected, followed 2 seconds later by the disconnection of the Chicago - New York link, and the disconnection of the San Jose - New York link half a second after that. This set of four disconnected links eliminates all of the possible paths from Los Angeles that can reach New York within 65ms. This situation lasts for 2 full seconds, until the San Jose - New York link is restored (followed shortly by the Washington, Baltimore, and Chicago links). While our dissemination-graph-based approach is able to resume timely service immediately

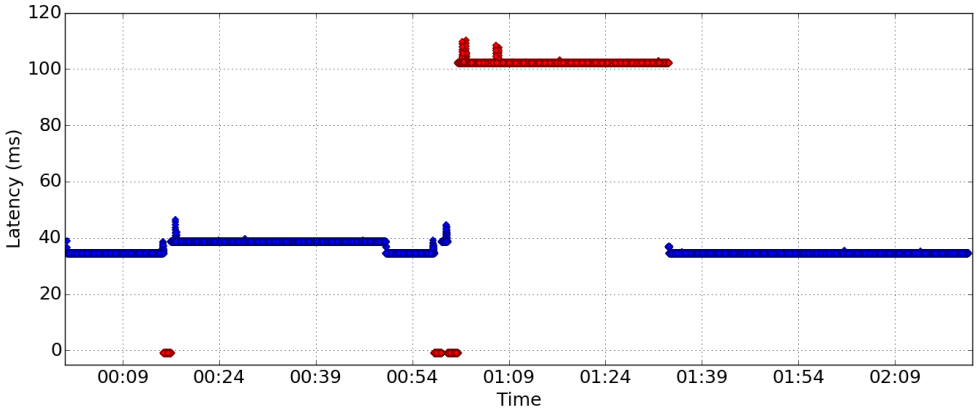
after this unavoidable 2-second interruption ends, the other approaches suffer a 30-second period of late delivery, due to the fact that they reroute the flow through Europe when the disconnection is detected and are not willing to return to any of the links that were disconnected until 30 seconds has elapsed.

This is a limitation of the expected-latency-based routing protocol used (and of most protocols used today) in the context of applications with strict timeliness constraints: because the routing protocol does not explicitly enforce the deadline, it can select paths that are not actually able to meet the requirements. One modification to the protocol that could help address this would be to restrict the operation of the routing protocol to the time-constrained flooding dissemination graph, to avoid including edges that cannot reach the destination on time (although this still does not strictly guarantee that the selected *paths* would be within the latency constraint). The benefits of dissemination-graph-based routing are first that it explicitly incorporates latency constraints, avoiding this issue, and further that it employs a higher degree of redundancy to avoid the need to attempt to select the *best* path(s) in the first place; it simply uses all viable options simultaneously in these problem cases.

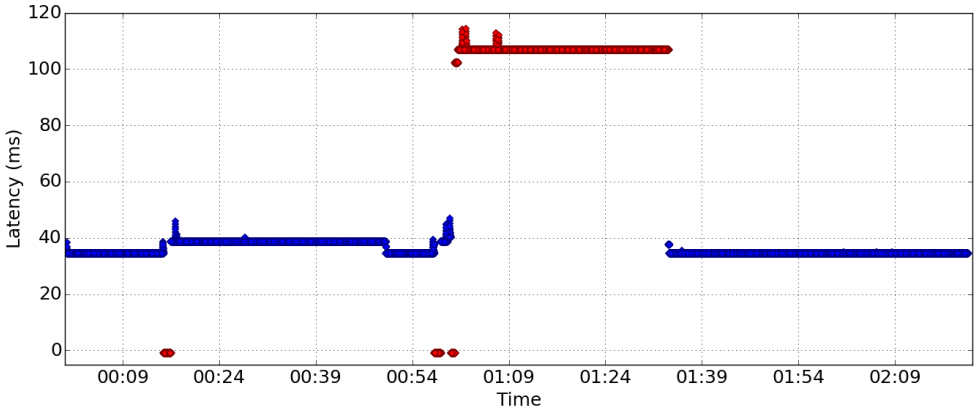
Routing	**Spines (1ms)**	**Playback (1ms)**	**Spines (10ms)**	**Playback (10ms)**	**Spines Wide-Area**
Single Path	36883.80 (SD=343.83)	29813.50 (SD=95.52)	36678.80 (SD=85.05)	29738.80 (SD=124.76)	35783.60 (SD=76.10)
Two Disjoint Paths	36006.00 (SD=171.86)	29081.00 (SD=164.05)	35997.80 (SD=85.49)	29098.20 (SD=64.28)	35670.40 (SD=89.51)
Dissemination Graphs	5920.80 (SD=170.98)	3098.75 (SD=169.24)	5913.40 (SD=85.14)	3123.20 (SD=68.31)	5563.40 (SD=86.90)

Table 6.8: Average lost or late packets for September 8, 2016 case study.

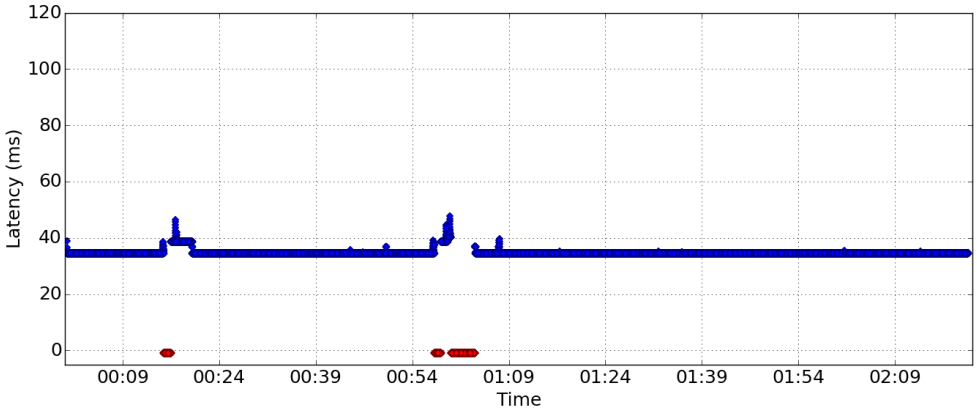
Table 6.8 shows the average number of lost or late packets over the five runs, as well as standard deviations. From the table, we can clearly see that the dissemination-graph-based protocol avoids the additional 30 seconds of failed delivery incurred by the other approaches, although it does suffer from the unavoidable disconnection, as well as the time it takes to detect and reroute to overcome the avoidable disconnections. Note that the overall loss in Spines is higher than reported by the simulator due to its link set up and tear down process: it takes somewhat longer both to react to disconnections and to restore links after a disconnection ends. The number of lost packets in the wide-area environment is slightly but consistently lower than in the local-area environment (by about 300+ packets). This is simply an artifact of the different loss/disconnection emulation methods used in the two environments. The way we change loss rates via NetEm in local-area environment has a slight delay that effectively makes the unavoidable disconnection last a few hundred milliseconds longer in the local-area setting.



(a) Spines 1 path: 36695 packets lost or late.

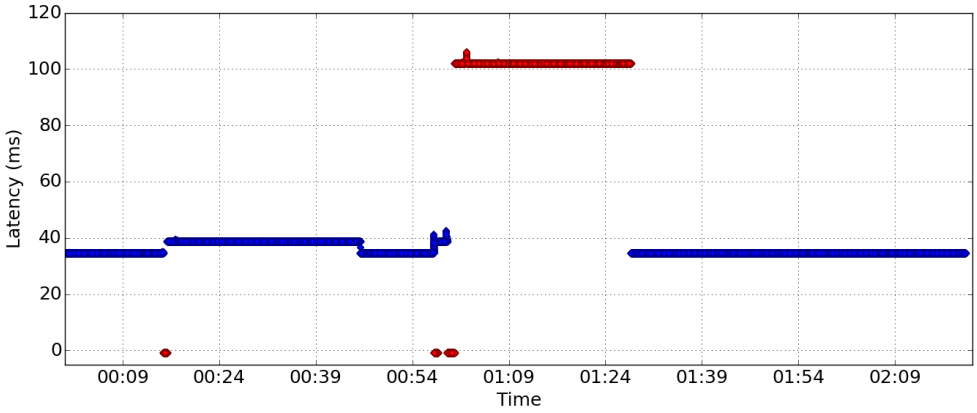


(b) Spines 2 paths: 36008 packets lost or late.

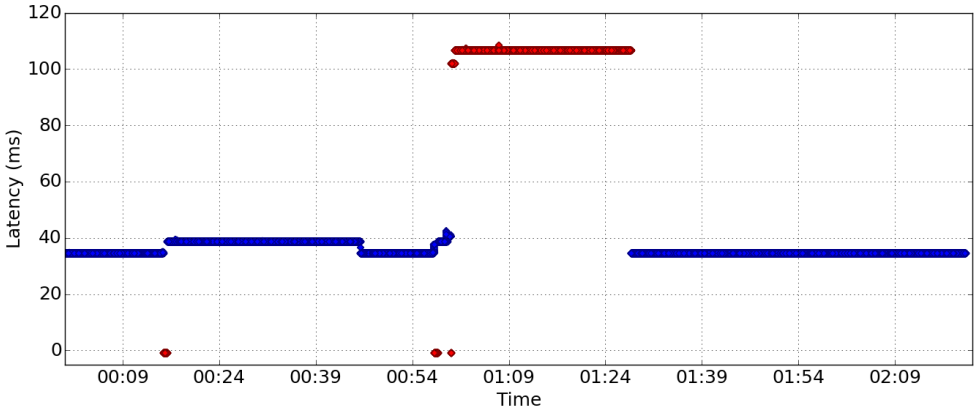


(c) Spines Dissemination Graphs: 5922 packets lost or late.

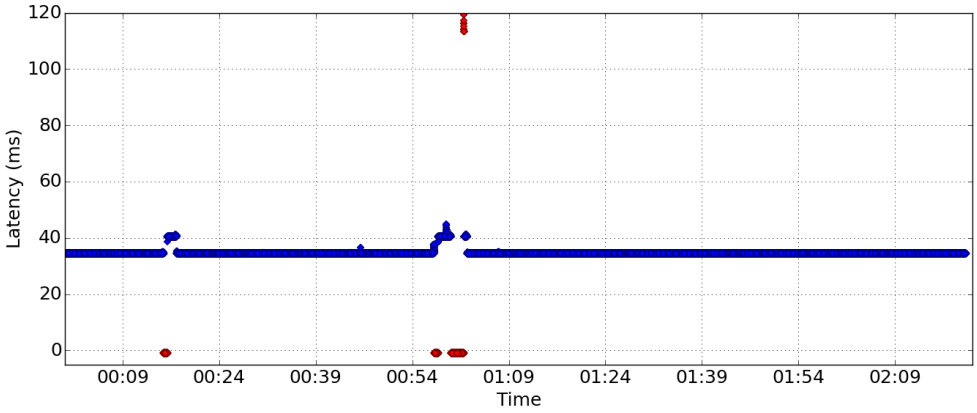
Figure 6.14: Spines results for New York to Los Angeles flow during a case-study inspired by an event on September 8, 2016, with the New York node experiencing several disconnections. Emulated latencies and loss rates in a local-area cluster environment.



(a) Simulation 1 path (1ms data collection, random seed = 1): 29763 packets lost or late.

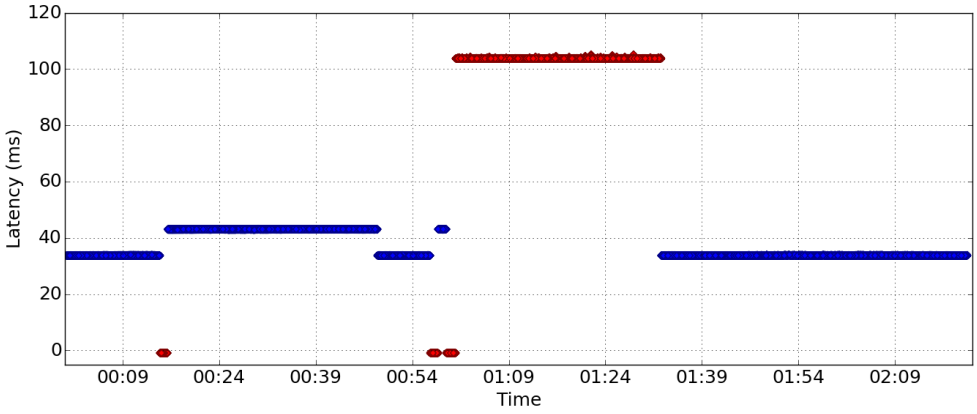


(b) Simulation 2 paths (1ms data collection, random seed = 1): 29182 packets lost or late.

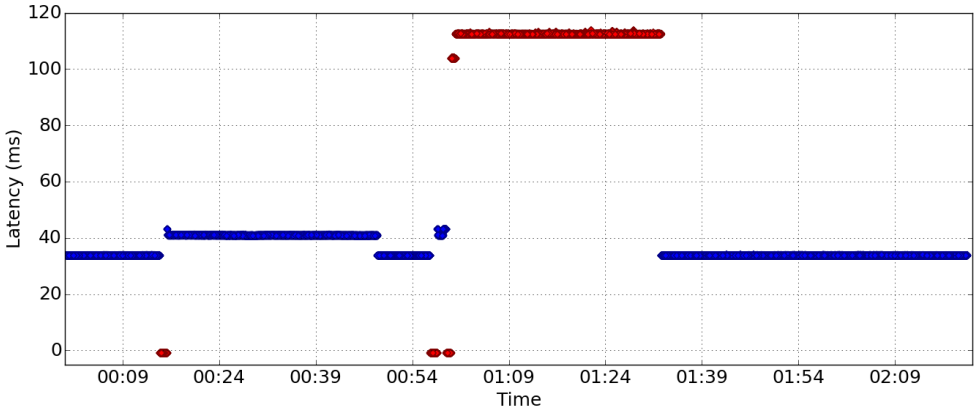


(c) Simulation Dissemination Graphs (1ms data collection, random seed = 1): 3224 packets lost or late (26 packets over 120ms not shown).

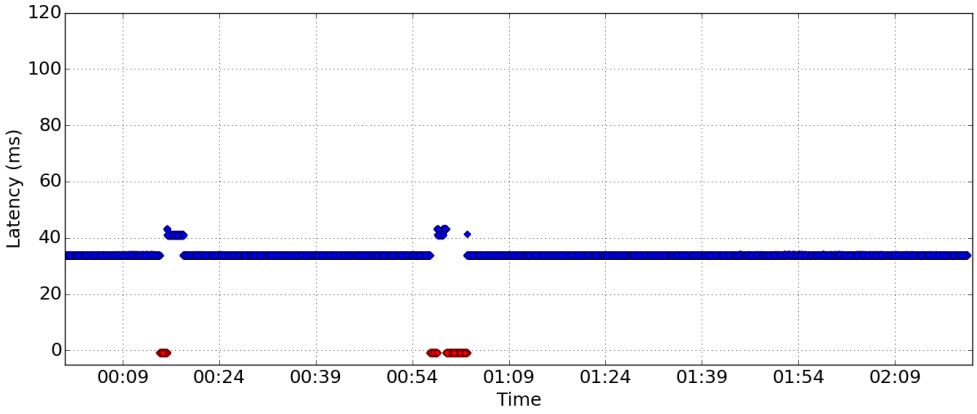
Figure 6.15: Playback Simulation results for New York to Los Angeles flow during a case-study inspired by an event on September 8, 2016, with the New York node experiencing several disconnections.



(a) Spines 1 path: 35776 packets lost or late.



(b) Spines 2 paths: 35660 packets lost or late.



(c) Spines Dissemination Graphs: 5552 packets lost or late.

Figure 6.16: Spines results for New York to Los Angeles flow during a case-study inspired by an event on September 8, 2016, with the New York node experiencing several disconnections. Real latencies and emulated loss rates in global wide-area environment.

6.5.3 Evaluation Summary

The above evaluation of our implementation of the dissemination-graph-based transport service in Spines demonstrates that our approach of using dissemination graphs with targeted redundancy improves performance compared with simpler approaches in practice under realistic network conditions. This evaluation validates the simulation results presented in Chapter 5 in two ways. First, the controlled evaluation, comparing results from Spines and the Playback simulator running in parallel on simple overlay topologies, shows that the simulation is able to accurately reflect performance differences between protocols we consider and can accurately model overall performance in terms of loss and latency (with the caveat that low levels of uniform loss are generally modeled too pessimistically when the coarser 10ms data collection interval is used, due to induced correlation reducing the effectiveness of recoveries). Second, the case studies, comparing a single path, two disjoint paths, and dissemination graphs with targeted redundancy under realistic conditions drawn from the data analyzed in Chapter 5, show that the Spines implementation is able to deliver the performance benefits claimed in scenarios like those we observed in the collected data from a real global overlay topology.

Chapter 7

Supporting Application Services

Our dissemination-graph-based transport service is designed to support applications with demanding combinations of timeliness and reliability requirements. In this chapter, we describe examples of two such applications: remote robotic manipulation and high-value video feed transmission. We assess our transport service's ability to support the communication needs of these applications, based on simulation using the four weeks of data collected on a real global overlay network described in Chapter 4. In both cases, these are high-value applications, where it is reasonable to pay the additional cost of redundant dissemination (about twice the cost of a single path), although the high overhead of an approach like time-constrained flooding would be impractical.

7.1 Remote Robotic Manipulation Support

A major use case for this service is to support remote robotic surgery or other remote manipulation tasks. These applications require high reliability and have extremely stringent timeliness constraints: in order for interaction to feel natural, the round-trip delay between performing an action and receiving a response (e.g. video, haptic feedback) must be less than about 130ms (65ms each way). Position updates for manipulating robots are commonly sent at a frequency of 1000 Hz.

This type of highly interactive application is discussed as our main motivation throughout this thesis, with the results in Sections 5.4 and 6.5 addressing support for such applications. These evaluations use a 65ms oneway latency constraint and a sending rate of one packet per millisecond to match the characteristics and requirements of remote manipulation applications. Table 5.1 from the evaluation in Section 5.4 is reproduced here as Table 7.1 and shows the level of availability and reliability our transport service can support for remote manipulation applications.

Routing Approach	Availability (%)	Unavailability (seconds per flow per week)	Reliability (%)	Reliability (packets lost or late per million)
Time-Constrained Flooding	99.995883%	24.90	99.999863%	1.37
Targeted Redundancy (via Dissemination Graphs)	99.995864%	25.02	99.999849%	1.51
Dynamic Two Disjoint Paths	99.995676%	26.15	99.999103%	8.97
Static Two Disjoint Paths	99.995266%	28.63	99.998438%	15.62
Redundant Single Path	99.995223%	28.89	99.998715%	12.85
Single Path	99.994286%	34.56	99.997710%	22.90

Table 7.1: Aggregate availability and reliability with 65ms latency constraint, over four weeks and sixteen transcontinental flows (using the recovery protocol of [1]).

Overall, the analysis shows that we can provide reliability such that less than 2 packets per million will not be successfully delivered at their destination within the time-constraint for flows between the East Coast of the US and the West Coast of the US.

Note that this reliability analysis excludes an average of 25.02 seconds per flow per week that the service is not available due to disconnections. While this is a relatively small period of time, we would like to support even higher availability, on the order of 99.999%. Unfortunately, in our analysis even the optimally reliable approach of time-constrained flooding suffers from similar unavailability, as it is largely due to cases where an entire site is disconnected from the rest of the network. To provide higher availability, the transport service could be augmented with multi-homing capabilities, allowing it to make use of multiple ISPs and avoid such disconnections; while the general structured overlay framework incorporates this, our current implementation does not.

Collaborating with robotics researchers, we have demonstrated an initial proof-of-concept for this application, remotely manipulating a robotic arm capable of performing robotic ultrasound. Using the LTN infrastructure and our Spines overlay messaging framework, we have shown that we can manipulate a robot located in a hospital at the Technical University of Munich, Germany from Johns Hopkins University in Baltimore, Maryland over the Internet with a one-way network latency of about 50ms.

7.2 High-Value Video Feed Support

Another use case for the service is to support high-value video feeds. Such feeds carry high-quality video (e.g. professional sporting events) to a few sites from which the video can ultimately be distributed to a large number of endpoints. Because any error in the original transmission can be propagated to millions of viewers, these high-value feeds require extremely high reliability (beyond normal broadcast quality).

Routing Approach	Availability (%)	Unavailability (seconds per flow per week)	Reliability (%)	Reliability (packets lost or late per million)
Time-Constrained Flooding	99.996560%	20.81	99.999995%	0.05
Targeted Redundancy (via Dissemination Graphs)	99.996544%	20.90	99.999985%	0.15
Dynamic Two Disjoint Paths	99.996525%	21.02	99.999757%	2.43
Static Two Disjoint Paths	99.996458%	21.42	99.999573%	4.27
Redundant Single Path	99.996293%	22.42	99.999359%	6.41
Single Path	99.996284%	22.48	99.998717%	12.83

Table 7.2: Aggregate availability and reliability with 200ms latency constraint, over four weeks and sixteen transcontinental flows (using the recovery protocol of [1]).

As timeliness constraints for these feeds are less strict than those for remote manipulation, our service can employ both redundant dissemination graphs with a greater diversity of viable paths and more opportunities for recovery to succeed to achieve even higher reliability.

Table 7.2 presents reliability and availability metrics for the same data as in Table 7.1, but analyzed with respect to applications with a timeliness constraint of 200ms rather than 65ms. Comparing Table 7.2 with Table 7.1, we can see that the less strict latency constraint allows us to achieve higher availability and to provide the higher level of reliability required for these high-quality (and high-value) feeds. In this case, our dissemination-graph-based transport service is able to provide over 99.9999% reliability. In fact, reliability in this case could potentially be improved even further by allowing multiple recovery attempts for lost packets [3], since the less restrictive timeliness requirement could allow such attempts to arrive within the latency constraint.

In this setting, unavailability is reduced for two reasons. First, the impact of each disconnection is slightly shortened, as the 200ms latency allowance makes it possible for packets that previously would have been lost near the end of the disconnection to be recovered once the disconnection is resolved. Second, the 200ms latency constraint makes it possible to use additional paths to overcome some types of disconnections: there are cases in which all paths that can reach the destination with 65ms are disconnected, but additional paths can provide delivery within 200ms. One example of this is the disconnection analyzed in the September 8, 2016 case study from Section 6.5.2.3: with a latency constraint of 200ms, routing through Europe to overcome the disconnection and reach New York from Los Angeles is a viable option.

These results show that a transport service using our dissemination-graph-based routing protocol with targeted redundancy can be useful for this type of application (with extremely high reliability requirements and somewhat less strict timeliness requirements) in addition to the highly interactive applications we target.

Chapter 8

Conclusion

We have presented *dissemination graphs*, providing a unified framework for specifying routing schemes based on paths, as well as more complex graphs. Based on an extensive analysis of real-world network data, we designed a dissemination-graph-based routing approach that employs targeted redundancy to invest resources in problematic areas of the network. We demonstrated through simulation using real network data that this approach can cost-effectively cover nearly 99% of the performance gap between a traditional single-path approach and an optimal but impractical scheme.

We have implemented the protocol to create a timely, reliable, and cost-effective transport Internet service within the open-source Spines structured overlay messaging framework (www.spines.org). Our evaluation of the implementation validates the simulation results through controlled experiments. We further demonstrate the benefit of dissemination graphs with targeted redundancy through several realistic case studies, inspired by the real-world network data.

Finally, we have analyzed the ability of the protocol to support two different classes of applications: remote robotic manipulation (with a 65ms latency constraint) and high-value video transport (with a 200ms latency constraint). This analysis demonstrates the effectiveness of the transport service for both applications and the improvement it provides over simpler routing protocols.

Bibliography

[1] Y. Amir, C. Danilov, S. Goose, D. Hedqvist, and A. Terzis, "An overlay architecture for high-quality VoIP streams," *IEEE Transactions on Multimedia*, vol. 8, no. 6, pp. 1250–1262, Dec 2006.

[2] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, Nov. 1984. [Online]. Available: http://doi.acm.org/10.1145/357401.357402

[3] Y. Amir, J. Stanton, J. Lane, and J. Schultz, "System and method for recovery of packets in overlay networks," U.S. Patent 8 437 267, May, 2013.

[4] LTN Global Communications, "LTN Global Communications," http://www.ltnglobal.com, retrieved September 26, 2018.

[5] P. Papadimitratos and Z. J. Haas, "Secure message transmission in mobile ad hoc networks," *Ad Hoc Networks*, vol. 1, no. 1, pp. 193 – 209, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870503000180

[6] T. Nguyen and A. Zakhor, "Path diversity with forward error correction (PDF) system for packet switched networks," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, March 2003, pp. 663–672 vol.1.

[7] K. Karenos, D. Pendarakis, V. Kalogeraki, H. Yang, and Z. Liu, "Overlay routing under geographically correlated failures in distributed event-based systems," in *On the Move to Meaningful Internet Systems*, 2010, pp. 764–784.

[8] Johns Hopkins Distributed Systems and Networks Lab, "The Spines messaging system," http://www.spines.org, retrieved September 26, 2018.

[9] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan, "Detour: informed internet routing and transport," *IEEE Micro*, vol. 19, no. 1, pp. 50–59, Jan 1999.

[10] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, 2001, pp. 131–145. [Online]. Available: http://doi.acm.org/10.1145/502034.502048

[11] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall, "Improving the reliability of internet paths with one-hop source routing." in *Proceedings of the 6th Usenix Symposium on Operating Systems Design and Implementation (OSDI)*, 2004, pp. 183–198.

[12] Y. Amir and C. Danilov, "Reliable communication in overlay networks," in *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, June 2003, pp. 511–520.

[13] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz, "OverQoS: An overlay based architecture for enhancing internet QoS," in *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*, 2004, pp. 71–84.

[14] Y. Amir, C. Danilov, S. Goose, D. Hedqvist, and A. Terzis, "1-800-OVERLAYS: Using overlay networks to improve VoIP quality," in *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2005, pp. 51–56. [Online]. Available: http://doi.acm.org/10.1145/1065983.1065997

[15] A. C. Snoeren, K. Conley, and D. K. Gifford, "Mesh-based content routing using XML," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, 2001, pp. 160–173. [Online]. Available: http://doi.acm.org/10.1145/502034.502050

[16] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan, "Best-path vs. multi-path overlay routing," in *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2003, pp. 91–100. [Online]. Available: http://doi.acm.org/10.1145/948205.948218

[17] D. Obenshain, T. Tantillo, A. Babay, J. Schultz, A. Newell, M. E. Hoque, Y. Amir, and C. Nita-Rotaru, "Practical intrusion-tolerant networks," in *Proceedings of the 36th International Conference on Distributed Computing Systems (ICDCS)*, June 2016, pp. 45–56.

[18] A. Bessani, N. F. Neves, P. Veríssimo, W. Dantas, A. Fonseca, R. Silva, P. Luz, and M. Correia, "JITeR: Just-in-time application-layer routing," *Computer Networks*, vol. 104, pp. 122–136, 2016.

[19] P. Papadimitratos, Z. J. Haas, and E. G. Sirer, "Path set selection in mobile ad hoc networks," in *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, 2002, pp. 1–11. [Online]. Available: http://doi.acm.org/10.1145/513800.513802

[20] J. G. Apostolopoulos, "Reliable video communication over lossy packet networks using multiple state encoding and path diversity," in *Proc. SPIE, Visual Communications and Image Processing*, vol. 4310, 2001, pp. 392–409. [Online]. Available: http://dx.doi.org/10.1117/12.411817

[21] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in cooperative environments," in *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003, pp. 298–313. [Online]. Available: http://doi.acm.org/10.1145/945445.945474

[22] L. Valiant, "The complexity of enumeration and reliability problems," *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, 1979.

[23] C. J. Colbourn, *The Combinatorics of Network Reliability*. New York, NY, USA: Oxford University Press, Inc., 1987.

[24] J. Provan and M. Ball, "The complexity of counting cuts and of computing the probability that a graph is connected," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 777–788, 1983.

[25] M. Jerrum, "On the complexity of evaluating multivariate polynomials," Ph.D. dissertation, University of Edinburgh, 1981.

[26] T. Farley, "Network reliability and resilience," Ph.D. dissertation, Arizona State University, 2009.

[27] H. Cancela, F. Robledo, G. Rubino, and P. Sartor, "Monte carlo estimation of diameter-constrained network reliability conditioned by pathsets and cutsets," *Computer Communications*, vol. 36, no. 6, pp. 611 – 620, 2013, reliable Network-based Services. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366412002861

[28] K. Aggarwal, Y. Chopra, and J. Bajwa, "Topological layout of links for optimizing the s-t reliability in a computer communication system," *Microelectronics Reliability*, vol. 22, no. 3, pp. 341 – 345, 1982. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0026271482900063

[29] J. Barrera, H. Cancela, and E. Moreno, "Topological optimization of reliable networks under dependent failures," *Operations Research Letters*, vol. 43, no. 2, pp. 132 – 136, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167637714001771

[30] B. Elshqeirat, S. Soh, M. Lazarescu, and S. Rai, "Dynamic programming for minimal cost topology with two terminal reliability constraint," in *2013 19th Asia-Pacific Conference on Communications (APCC)*, Aug 2013, pp. 740–745.

[31] B. Elshqeirat, "Optimizing reliable network topology design using dynamic programming," Ph.D. dissertation, Curtin University, 2015.

[32] A. Babay, E. Wagner, M. Dinitz, and Y. Amir, "Timely, reliable, and cost-effective internet transport service using dissemination graphs," in *Proceedings of the 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 1–12.

[33] J. W. Suurballe, "Disjoint paths in a network," *Networks*, vol. 4, no. 2, pp. 125–145, 1974. [Online]. Available: http://dx.doi.org/10.1002/net.3230040204

[34] A. Babay, "Timely, reliable, and cost-effective transport service using dissemination graphs," in *IEEE/IFIP International Conference on Dependable Systems and Networks (Student Forum)*, June 2015.

[35] E. Wagner, "The Playback network simulator: Overlay performance simulations with captured data," Masters Project, Johns Hopkins University, December 2016.

[36] D. Obenshain, "Practical intrusion-tolerant networking," Ph.D. dissertation, Johns Hopkins University, November 2015.

[37] M. V. Marathe, R. Ravi, R. Sundaram, S. Ravi, D. J. Rosenkrantz, and H. B. Hunt, "Bicriteria network design problems," *Journal of Algorithms*, vol. 28, no. 1, pp. 142 – 171, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196677498909300

[38] A. Babay, M. Dinitz, and Z. Zhang, "Characterizing demand graphs for (fixed-parameter) shallow-light steiner network," *CoRR*, vol. abs/1802.10566, 2018. [Online]. Available: http://arxiv.org/abs/1802.10566

[39] L. Guo, K. Liao, and H. Shen, "On the shallow-light steiner tree problem," in *15th International Conference on Parallel and Distributed Computing, Applications and Technologies*, Dec 2014, pp. 56–60.

[40] M. T. Hajiaghayi, G. Kortsarz, and M. R. Salavatipour, "Approximating buy-at-bulk and shallow-light k-steiner trees," *Algorithmica*, vol. 53, no. 1, pp. 89–103, 2009. [Online]. Available: http://dx.doi.org/10.1007/s00453-007-9013-x

[41] C. Danilov, "Performance and functionality in overlay networks," Ph.D. dissertation, Johns Hopkins University, September 2004.

Vita

Amy Babay was born in 1990 in Erie, Pennsylvania. She received her BA in Cognitive Science in 2012 and her MSE in Computer Science in 2014, both from Johns Hopkins University. Prior to starting her PhD, she gained experience with global overlay networks in the commercial world, working at LTN Global Communications. As an MSE and PhD student, Amy was a member of the Distributed Systems and Networks Lab, where her research focused on enabling new Internet services using structured overlay networks and on building intrusion-tolerant critical infrastructure systems. She received the Johns Hopkins Computer Science Department's Special Service Award in 2015 and the Professor Joel Dean Excellence in Teaching Award in 2018.