MIT Laboratory for Computer Science
545 Technology Square, NE43-512
Cambridge, MA 02139
(617) 452-2820
snoeren@lcs.mit.edu

January 1, 2002

To Whom it May Concern:

I am writing to apply for a tenure-track position as an assistant professor in your department. I am currently a PhD candidate in the Electrical Engineering and Computer Science department at the Massachusetts Institute of Technology and expect to complete my dissertation work this year. My research interests lie in the area of computer systems, especially networking, operating systems, and mobile and distributed computing. I am particularly interested in scalable services and protocols to support secure, wide-area mobile internetworking.

I have enclosed my curriculum vitae, a list of references, statements of research and teaching interest, and four representative publications. These materials are also available electronically in both PDF and Postscript formats at http://nms.lcs.mit.edu/~snoeren/appkit/.

I look forward the opportunity to discuss my application in person.

Sincerely,

Alex C. Snoeren

encl: Curriculum Vitae (including names of references)
Statement of Research Interests & Methods
Statement of Teaching Interests
"Hash-Based IP Traceback"
"An End-to-End Approach to Host Mobility"
"Mesh-Based Content Routing using XML"
"Adaptive Inverse Multiplexing for Wide-Area Wireless Networks"

[This page intentionally left blank.]

Alex C. Snoeren

MIT Laboratory for Computer Science
545 Technology Square, NE43-512
Cambridge, MA 02139
(617) 452-2820

snoeren@lcs.mit.edu
http://nms.lcs.mit.edu/~snoeren

Education

Massachusetts Institute of Technology

2002 PhD in Electrical Engineering and Computer Science (*Summer completion expected*)
Dissertation: *An End-to-End Approach to Internet Mobility*
Hari Balakrishnan and M. Frans Kaashoek, advisors

Minor in Public Policy (at the Harvard JFK School of Government)

Georgia Institute of Technology

1997 MS in Computer Science
BS in Applied Mathematics (*summa cum laude*)
1996 BS in Computer Science (*summa cum laude*)

Research interests

Many aspects of computer systems, especially operating systems, networking, and mobile and distributed systems. Particularly interested in scalable services and protocols to support secure, robust wide-area mobile internetworking.

Employment summary

1997– **Massachusetts Institute of Technology**
Research assistant in the Advanced Network Architecture, Networks & Mobile Systems, and Parallel & Distributed Operating Systems groups of the Laboratory for Computer Science

1999– **BBN Technologies**
Research scientist in the Internetworking Research department

1995–7 **Hewlett-Packard Research Labs**
Research intern in the Performance, Management & Design group of the Software Technology lab

1994 **NASA Lewis (Glenn) Research Labs**
Summer research intern in the Electro-Physics branch

Teaching experience

Massachusetts Institute of Technology

2001 **Instructor** – *6.033: Computer Systems Engineering*
A position usually held by a member of the faculty. Prepared and taught weekly recitation sessions centered around the discussion of current and seminal papers in computer systems. The core MIT undergraduate systems course, 6.033 is a broad survey of the field of computer systems, including modularity, concurrency, file systems, networking, fault tolerance, etc.

1998 **Recitation instructor** – *6.821: Programming languages*
Taught weekly recitation sections to about 25 students. Answered students' questions, graded problem sets, led quiz reviews. Developed course material, including problem sets, exams, and Scheme code.

Georgia Institute of Technology

1997 **Teaching assistant** – *CS 4345: Computerization and Society*
Lead classroom discussions, answered questions, graded reports, and helped students with writing assignments on the societal impact of computing.

1994–7 **Recitation instructor** – *CS 2360: Knowledge Representation and Process* (**Six terms**)
Taught weekly lab/recitation section of approximately 30 students. Developed course assignments and code base, answered questions, graded assignments, and assisted students with LISP.

1996 **Teaching assistant** – *CS 3361: Artificial Intelligence*
Graded programming assignments, answered questions, and helped develop class assignments.

1995 **Recitation instructor** – *CS 2430: Control and Concurrency*
Taught weekly lab/recitation section of approximately 30 students. Answered questions, graded assignments, and assisted students with C. Lectured to class of 250 students in professor's absence on IPC and synchronization. Nominated by the course instructor for the graduate teaching award as an undergrad.

1994 **Recitation instructor** – *CS 1501: Introduction to Computing* (**Two terms**)
Assisted in the development of a new introductory course in computer science. Taught weekly recitation and lab sections of 25 students, graded homework, quizzes, and exams, designed and taught review sessions, and helped students learn a special-purpose Pascal-like language used in the course.

Advising

2001– M. Eng. Thesis — *Massachusetts Institute of Technology*
Along with Hari Balakrishnan, co-supervise Jon Salz, a student whose M. Eng. thesis on *TESLA: A Transparent Extensible Session-Layer Architecture* is related to dissertation topic.

1995–6 Academic advisor — *SPAARC, Georgia Institute of Technology*
Assisted undergraduate students in selecting appropriate classes, choosing majors, and planning their course of study. Provided guidance for majors in the colleges of computing and sciences.

Awards

2001 Best Student Paper, ACM SIGCOMM

1997 Department of Defense National Science and Engineering Graduate Fellowship
National Science Foundation Graduate Research Fellowship *(Declined)*

1993–7 Georgia Institute of Technology Faculty Honors (*4.0 GPA*)

1996 Rhodes Scholar Finalist
Outstanding College of Computing Undergraduate, Georgia Tech

1993 National Merit Scholar
National Science Scholar
National Academy of Space, Science & Technology Scholar
Georgia Institute of Technology President's Scholar

Member, ANAK Georgia Tech senior honor society
Member, Omicron Delta Kappa leadership honor society (past Alpha-Eta Circle president)
Member, Golden Key academic honor society
Member, Delta Chi fraternity

Patents

2001 Two patent applications submitted, covering various aspects of the Source Path Isolation Engine (SPIE) technology developed jointly with co-inventors at BBN Technologies.

Refereed publications

Journal articles

2002 "Single-Packet IP Traceback." Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent, and W. Timothy Strayer. To appear in *IEEE/ACM Transactions on Networking* (ton), Volume 10, 2002.

2001 "FIRE: Flexible Intra-AS Routing Environment." Craig Partridge, Alex C. Snoeren, W. Timothy Strayer, Beverly Schwartz, Matthew Condell, and Isidro Castineyra. *IEEE Journal on Selected Areas in Communications* (j-sac), Volume 19, Number 3, March 2001.

Conference papers

2001 "Mesh-Based Content Routing using XML." Alex C. Snoeren, Kenneth Conley, and David K. Gifford. *Proceedings of the 18th ACM Symposium on Operating System Principles* (sosp 18), Banff, Canada, October 2001.

"Hash-Based IP Traceback." Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Stephen T. Kent, and W. Timothy Strayer. *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (sigcomm '01), San Diego, California, August 2001.

"Hardware Support for a Hash-Based IP Traceback." Luis A. Sanchez, Walter C. Milliken, Alex C. Snoeren, Fabrice Tchakountio, Christine E. Jones, Stephen T. Kent, Craig Partridge, and W. Timothy Strayer. *Proceedings of the Second DARPA Information Survivability Conference and Exposition* (discex II), Anaheim, California, June 2001.

"Fine-Grained Failover Using Connection Migration." Alex C. Snoeren, David G. Andersen, and Hari Balakrishnan. *Proceedings of the Third USENIX Symposium on Internet Technologies and Systems* (usits '01), San Francisco, California, March 2001.

2000 "FIRE: Flexible Intra-AS Routing Environment." Craig Partridge, Alex C. Snoeren, W. Timothy Strayer, Beverly Schwartz, Matthew Condell, and Isidro Castineyra. *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (sigcomm 2000), Stockholm, Sweden, August 2000.

"An End-to-End Approach to Host Mobility." Alex C. Snoeren and Hari Balakrishnan. *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking* (MobiCom 2000), Boston, Massachusetts, August 2000.

1999 "Adaptive Inverse Multiplexing for Wide-Area Wireless Networks." Alex C. Snoeren. *Proceedings of the IEEE Conference on Global Communications* (GlobeCom '99), *Global Internet Symposium*, Rio de Janiero, Brazil, December 1999.

Workshop papers

2001 "The Migrate Approach to Internet Mobility." Alex C. Snoeren, Hari Balakrishnan, and M. Frans Kaashoek. *Proceedings of the Student Oxygen Workshop* (sow '01), Gloucester, Massachusetts, July 2001.

"Reconsidering Internet Mobility." Alex C. Snoeren, Hari Balakrishnan, and M. Frans Kaashoek. *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems* (HotOS-VIII), Elmau, Germany, May 2001.

1998 "Automated Whole-System Diagnosis of Distributed Services Using Model-Based Reasoning." George Forman, Mudita Jain, Masoud Mansouri-Samani, Joseph Martinka, and Alex C. Snoeren. *Proceedings of the Ninth IFIP/IEEE Workshop on Distributed Systems: Operations and Management* (dsom '98), Newark, Delaware, October 1998.

(All publications are available electronically from http://nms.lcs.mit.edu/~snoeren/publications.html)

Talks

2001 "Hash-Based IP Path Tracing," IP Path Tracing (IPPT) BOF, 52nd Internet Engineering Task Force Meeting (IETF 52), Salt Lake City, Utah, December 13, 2001.

"Mesh-Based Content Routing using XML," 18th ACM Symposium on Operating Systems Principles (SOSP 18), Banff, Canada, October 23, 2001.

——, PEO Interchange XML Initiative (PIXIT) Meeting, The MITRE Corporation, Bedford, Massachusetts, October 16, 2001.

"Hash-Based IP Traceback," ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '01), San Diego, California, August 29, 2001.

——, Systems Seminar, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, October 18, 2001.

"Reconsidering Internet Mobility," Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII), Elmau, Germany, May 21, 2001.

"Fine-Grained Failover Using Connection Migration," Third USENIX Symposium on Internet Technologies and Systems (USITS '01), San Francisco, California, March 28, 2001.

2000 "FIRE: Flexible Intra-AS Routing Environment," ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM 2000), Stockholm, Sweden, August 31, 2000.

"An End-to-End Approach to Host Mobility," Sixth Annual International Conference on Mobile Computing and Networking (MobiCom 2000), Boston, Massachusetts, August 9, 2000.

"TCP Connection Migration," IRTF End-to-End Research Group Meeting, Cambridge, Massachusetts, June 23, 2000.

1999 "Adaptive Inverse Multiplexing for Wide-Area Wireless Networks," IEEE Conference on Global Communications (GlobeCom '99), Global Internet Symposium, Rio de Janiero, Brazil, December 6, 1999.

1996 "PROMISE - Using Flipper for Automated System Diagnosis: A Tutorial," HP Research Labs, Palo Alto, California, September 1996.

Service

1998– Referee — *GlobeCom, HotOS, MobiCom, MoMuC, SOSP, USITS, CCR, J-SAC, TOCS*

2000 Network Reading Group (NetRead) Organizer — *MIT Laboratory for Computer Science*
Organized weekly student-led seminars discussing recent and seminal papers in computer networking and related fields.

1997 Curriculum Committee Member — *Georgia Tech College of Computing*
Participated in the complete redefinition of the requirements for the Bachelors degree in computer science at Georgia Tech in preparation for the conversion from quarters to the semester system in the fall of 1999.

Graduate senator — *Georgia Tech Student Government Association*
Served as one of two senators representing the College of Computing in the Graduate Senate.

1996 Computer Ownership Committee Member — *Georgia Institute of Technology*
Helped define the first mandatory computer ownership policy in the state university system, instituted in Fall 1998. Assisted the President and Provost in presenting the policy to the Georgia Board of Regents.

1995–6 Department representative — *Georgia Tech Student Government Association*
Served as one of two students representing the College of Computing in the Undergraduate House.

Research summary

Massachusetts Institute of Technology

1999– **End-to-end mobility** – Dissertation research investigates end-to-end techniques to support mobility and disconnected operation in the Internet. As Internet hosts become increasingly mobile, many applications require additional system support. Developed a mobility architecture called *Migrate* that allows session-based applications to gracefully handle changes in network attachment point as well as unexpected periods of disconnectivity. The Migrate architecture is based solely on end-host signaling and requires no additional infrastructure support.

Migrate provides programmatic system support for managing session state during periods of intermittent connectivity, and allows for the reallocation of both application and kernel resources during periods of disconnection. By constantly monitoring network connectivity, interpreting user preferences, and reacting to changes in network conditions, Migrate provides a unified framework to ease the burden of mobile-aware application programming.

An initial version of the mobility architecture, based upon a novel TCP migration scheme, was presented in 1999. A full-featured version, implemented as a session-layer service, forms the core of the thesis. A system-wide Migrate service monitors network performance and brokers session communication, guided by user- and application- specified policies. An extended Migrate API provides mobile-aware applications with advanced session-based resource management facilities, while a conformance library allows legacy applications to leverage basic mobility and disconnectivity support.

2001– **Robust content distribution** – Some high-value streaming applications require greater reliability and lower latency than is typically available by current Internet distribution technologies. One way to simultaneously increase reliability and decrease latency is to send redundant information over disjoint paths, sometimes called *dispersity routing*. Helped design and implement an extremely reliable and timely content streaming service by distributing data through a redundant mesh of overlay routers.

Co-designed and co-implemented the Diversity Control Protocol (DCP), which allows a receiver to intelligently reassemble a packet flow redundantly forwarded over multiple paths, enabling low-latency delivery of time-critical data in the presence of lossy channels. DCP forms the basis of a mesh-based XML content delivery network. A prototype network delivers XML-encoded real-time air traffic data in a timely and reliable fashion.

1997–9 **IP over wireless links** – In an effort to achieve higher data rates using current-generation wide-area wireless technologies, developed a technique to efficiently bundle shared heterogeneous links called *link quality balancing* (LQB). Unlike previous techniques, which assume stable (typically identical) performance characteristics across bundled links, LQB provides high throughput even in the presence of dynamic channel and traffic characteristics. Implemented LQB in FreeBSD's multi-link PPP module, and used it to multiplex IP traffic over bundled Cellular Digital Packet Data (CDPD) channels. Installed a prototype router (and supporting power and communication equipment) into an AMC Hummer all-terrain vehicle.

BBN Technologies

2000– **Denial of service detection** – Due to the anonymous nature of the IP protocol, it is impossible to reliably discern the source of an IP packet by examining it upon receipt. Increased awareness of the significant damage potential of denial-of-service attacks has motivated techniques to isolate the source of IP packets. Most known *IP traceback* techniques, however, require a large number of packets from the same source in order to isolate the sender. Along with several other researchers, developed a hash-based IP traceback system, the Source Path Isolation Engine (SPIE). SPIE represents the only known scalable solution that can trace a single IP packet to its source, even if it was transformed (tunneled, NATed, fragmented, etc.) in flight.

Through the use of Bloom filters, SPIE-enabled routers can efficiently maintain a sliding history of packets they have forwarded in the recent past. By recursively inspecting packet histories at each router backwards along the packet's path to the destination, SPIE is able to determine the path traversed by any arbitrary, recently-received IP packet.

1999–2000 **Active networking** – The promise of active networking has proven difficult to achieve in routing protocols. While the ability to continually describe custom forwarding paths for specific classes of traffic is attractive, designing a secure, scalable, and robust routing protocol is no small task. Hence, operators are loathe to adjust tuning parameters in operational networks, let alone replace routing protocols.

Helped design and implement the Flexible Intra-AS Routing Environment (FIRE), an extensible routing protocol that enables class-based routing through the use of multiple forwarding tables, constructed by downloadable routing algorithms using user-defined metrics. These algorithms build forwarding tables based upon link-state metrics collected using a shared, secure distribution protocol. FIRE frees traffic engineers to focus on routing algorithms by alleviating the burden of developing a distribution protocol.

Hewlett-Packard Research Labs

1995–7 **Distributed application management** – Over the course of three summers, assisted in the design and implementation of a Model-Based Reasoning system for distributed application management, PROMISE. Modeled distributed applications using a novel inference language that allowed the expression of performance dependencies between components of a distributed system. Performance monitoring and control services enabled application management at the infrastructure (CORBA, DCE) level. In addition to service modeling, was responsible for network monitoring and configuration, as well as graphical user interfaces for model visualization.

NASA Lewis (Glenn) Research Labs

1994 **Monte-Carlo simulation** – Designed and implemented Monte-Carlo simulations of spacecraft coating degradation during low-earth orbit due to high atomic-oxygen fluence, resulting in order-of-magnitude performance improvements over previous simulations, with no loss of accuracy. The physical phenomena resulting in decay occurs with infinitesimal probability, forcing traditional simulations to run for days before producing statistically significant results. By leveraging probability distributions, was able to dramatically decrease the number of runs required for convergence.

References

Prof. Hari Balakrishnan
MIT Laboratory for Computer Science
545 Technology Square, ne43-510
Cambridge, ma 02139
(617) 253-8713
hari@lcs.mit.edu

Prof. M. Frans Kaashoek
MIT Laboratory for Computer Science
545 Technology Square, ne43-522
Cambridge, ma 02139
(617) 253-7149
kaashoek@lcs.mit.edu

Prof. David K. Gifford
MIT Laboratory for Computer Science
545 Technology Square, ne43-401
Cambridge, ma 02139
(617) 253-6039
gifford@lcs.mit.edu

Dr. Craig Partridge
BBN Technologies
10 Moulton Street
Cambridge, ma 02138
(517) 324-3425
craig@bbn.com

Additional references available upon request.

Statement of Research Interests & Methods

An advisor once described me as "being interested in any interesting problem." I look forward with great anticipation to establishing a research group with students working on diverse problems in networking and computer systems that we mutually find intriguing. Supervising an M. Eng. student at MIT recently, I learned that working closely with students can be a wonderful source of creativity and insight. I hope to foster an environment where I can collaborate with students on related, but distinct projects that personally excite each of them. I enjoy tackling several projects at once. Inspiration strikes on its own schedule, and I often find it rewarding to pursue several promising ideas in parallel.

At MIT, I've worked with a number of different faculty members on research projects in a broad range of areas, including mobile networking, wireless network performance, robust content distribution, fault-tolerant Internet services, and distributed application management. In addition, I have worked part-time at BBN Technologies for the past three years, where I have focused on active networking and denial of service detection. While the focus of my projects has been varied, there are several unifying themes in my research. Chief among them is an emphasis on networking, particularly in the mobile and wide-area environments. Networks have become the life-blood of computer systems and provide a virtually inexhaustible source of new topics for exploration.

My projects endeavor to develop new functionality. Achieving good performance is critical, but the art of squeezing out the last few cycles is best used in moderation. I strive to develop systems that extend the state-of-the-art, and qualitatively, rather than only quantitatively, improve the fields of networking and computer systems. I derive great satisfaction from doing something today that was impossible yesterday.

There is also commonality in the way I approach problems. While all researchers enjoy charting new territory, it is often fruitful to re-examine classical problems with known (limited) solutions. I enjoy considering alternative approaches. Where the classical approach remains superior, examining unexplored alternatives may yield a deeper understanding of the limits or applicability of the current approach. However, if one seeks to identify domains in which circumstances have changed, or new technologies have been developed, one may find techniques previously discarded as foolish or infeasible are relevant once again. This is precisely the case in two of my recent research projects.

My dissertation research proposes, implements, and evaluates the Migrate architecture for Internet host mobility. A fundamental problem in mobile networking is preserving connectivity when communicating hosts change network locations and efficiently resuming communication after periods of disconnection. Current approaches to host mobility, such as Mobile IP and ad-hoc routing, assume Internet hosts are either rarely mobile or constantly mobile and always on and reachable. Neither model is appropriate for the intermittently connected laptops and PDAs that populate today's Internet, and both approaches lack the application support necessary for the cell phones and personal communication devices that will comprise tomorrow's. Migrate addresses these shortcomings by handling intermittent connectivity and a variety of mobility modes.

One interesting technical component of Migrate is a novel TCP connection migration scheme. It allows a TCP connection to be resumed from a different IP address, solving a problem that Vint Cerf described in a 1983 paper as "[having] plagued network designers since the design

of the ARPANET in 1968." A TCP connection is identified by its end points; hence, a change in end point (IP address) invalidates the connection. TCP migration schemes have been proposed previously but failed in various ways to preserve the semantics, performance, or security of the connection. The Migrate option avoids such ill-effects by utilizing multiple, standard TCP connections. It negotiates a cryptographically secure identifier for each connection which can then be used to continue over an entirely new connection at another end point. The key observation is that only the connection end points need to be aware of the previous connection; all other entities in the network (routers, NATs, firewalls, etc.) should view the continuation as another, independent connection. So long as the end points ensure they can stitch together a TCP connection and its continuation, the resulting connection must perform identically to a standard TCP connection, as the network cannot tell the difference.

At BBN, I helped design the Source Path Isolation Engine (SPIE), which was developed to help combat denial of service and other IP-based attacks. SPIE identifies the true source of an IP packet and the path the packet followed while in the network, allowing victims to reliably identify their attacker(s). Paths are constructed by maintaining records of every packet forwarded at each router, and simply querying each router backwards along the path beginning at the victim. This general technique, known in the literature as packet logging, was previously regarded as intractable in the wide-area due to the massive packet histories required. The centerpiece of SPIE is a hash-based packet digesting procedure that reduces the storage requirement of an IP packet to only a few bits, bringing packet logging into the realm of practicality.

My experience with both the cryptographic aspects of Migrate and the hashing and data storage requirements of SPIE has reinforced my belief that systems research is often best conducted in a collaborative, cross-disciplinary fashion. It is critical that systems researchers spend time with experts in more formal disciplines, such as mathematicians and theoreticians. Recent systems projects in a wide range of areas, from distributed storage systems to overlay networks to event notification systems, have shown theoretical computer science can provide a powerful toolbox of techniques for those with sufficient understanding, desire, and patience to search for creative applications.

Do not mistake my predilection for clever theoretical and algorithmic approaches as an aversion to detailed implementation. I firmly believe that high-quality computer systems research requires building, deploying, and evaluating real systems. I have built every system I've proposed as part of my graduate research, and each implementation experience has presented additional complications, subtleties, and sometimes even principles that would otherwise have gone unnoticed. Not only is a full-scale implementation absolutely essential to fully understanding the implications of a proposed system, but it often uncovers fertile areas for future research as well—perhaps unrelated to the project at hand. In either case, system building keeps faculty in touch with today's trends, and produces students who are intimately familiar with the details of real systems, knowledge and skills they will find extremely valuable in both academic research and industry.

Moving forward, I plan to continue to focus on providing new functionality in computer systems. In particular, I believe the ever-increasing permeation of light-weight, mobile computing devices will challenge classical notions of computer systems and provide a fertile area for research. Just as recent interest in peer-to-peer systems has rejuvenated work in distributed file systems, I expect the significant computational power now available in a hand-held form factor will lead to a renaissance of distributed computing. I look forward to designing systems

agile enough to function on such a substrate and networks capable of securely and robustly communicating between them. In the near term, I intend to explore topics exposed by my dissertation and recent work in robust content distribution.

The Migrate architecture proposed in my dissertation addresses host mobility, but stops short of handling user or service mobility, where an application or communication end point moves not only across network location, but across computational location (e.g. from laptop to cell phone) as well. As cell phones, PDAs, and other devices approach the computational power available in laptops, the ability to migrate sessions across devices becomes increasingly practical and desirable. This form of migration is complicated by the heterogeneity of system resources and methods of user interaction, but may yield to a continuation-based approach similar that advocated by Migrate.

In another thrust, I hope to extend my recent work on content distribution. Mesh-based, multi-path overlay routing has shown promise in delivering greater levels of reliability and performance than traditional distribution networks, but little is currently known about how to effectively construct distribution topologies in the wide area, or how their configuration affects overall network performance. By making a general-purpose, standards-based content distribution utility available to fellow researchers, I hope to build a distribution network large enough to study such effects.

[This page intentionally left blank.]

Statement of Teaching Interests

More than anything else, a desire to teach at the collegiate level motivated my pursuit of a PhD. This passion ignited in only my second term on a college campus, when I was offered a position as a recitation instructor for CS1501, Georgia Tech's introductuctory computer science course. That initial experience was so addictive that I continued to serve as a recitation instructor or teaching assistant each term for the remainder of my four years at Georgia Tech, including five different classes over the course of eleven academic terms. My years at MIT have been tightly focused on research activities, yet I taught two additional courses, one as a recitation instructor and the last as a full-fledged instructor, along side several professors. In total, these seven different courses span the entire range from freshman-year introductory courses to the advanced graduate level.

Regardless of the level or subject matter, my teaching is driven by a quest to impart intuition and inspire curiosity. Facts and methods learned by rote are likely soon forgotten, but the seeds of interest, once planted, often sprout anew. During my four years of teaching at Georgia Tech, I had the opportunity to teach several students multiple times as they proceeded through the curriculum. I found a great sense of satisfaction in watching students who I taught in the first introductory course succeed at progressively higher and higher levels. Just recently, I chanced upon a student whom I recalled having considerable difficulty in my CS1501 class as he searched for direction in his studies. Now seven years later, he informed me that he had not only successfully graduated with a degree in Computer Science, but was employed as a software engineer. The joy the news brought me reaffirmed my belief that I have chosen the right vocation.

My broad academic background equips me to teach a range of undergraduate courses, from my core expertise in networking and operating systems to more distant topics such as computer architecture, algorithms, and discrete mathematics. I take pride in my ability to make complex material accessible at many levels. I strive to frame lectures around intuitive explanations backed by concrete examples. Examples, whenever possible, are reinforced with illustrative problems. I try to craft exam and homework problems that not only reinforce material covered in lecture, but are learning experiences in themselves.

At the graduate level, I look forward to teaching topics directly related to my research, such as computer networking and advanced operating systems. It is my firm belief that graduate courses in computer systems should be project-based. Carefully crafted class projects frequently produce publishable results, and, more importantly, motivated researchers. To that end, I would be particularly excited to develop a seminar course on mobile internetworking or content distribution networks as a vehicle for introducing students to my areas of research.

Classroom learning is only part of the educational process. Once a student has been motivated to move beyond the classroom and into the research laboratory, the professor's role takes on an additional dimension. A student's advisor can greatly influence the success of student's graduate career, not only through straightforward means such as exposing students to interesting and topical problems, but in far more subtle, yet likely more critical ways like thoughtful selection of office-mates and project teams. My most rewarding experiences as a researcher have come from deep, collaborative immersion in a focused problem area. I believe such experiences are critical for developing a taste for promising research areas and defining crisp problems to address. I have watched as my thesis advisors, Hari Balakrishnan and Frans Kaashoek, have constructed productive and thriving research groups. While the flavor and personality of the two groups differ, in my view, the groups share a similar cohesive spirit of collaborative exploration and critical self-examination fostered by careful mentoring. I look forward to continuing this tradition of collegial collaboration as I form my own research group.

[This page intentionally left blank.]

Hash-Based IP Traceback

Alex C. Snoeren[†], Craig Partridge, Luis A. Sanchez[‡], Christine E. Jones,
Fabrice Tchakountio, Stephen T. Kent, and W. Timothy Strayer

BBN Technologies
10 Moulton Street, Cambridge, MA 02138

{snoeren, craig, cej, ftchakou, kent, strayer}@bbn.com

ABSTRACT

The design of the IP protocol makes it difficult to reliably identify the originator of an IP packet. Even in the absence of any deliberate attempt to disguise a packet's origin, wide-spread packet forwarding techniques such as NAT and encapsulation may obscure the packet's true source. Techniques have been developed to determine the source of large packet flows, but, to date, no system has been presented to track individual packets in an efficient, scalable fashion.

We present a hash-based technique for IP traceback that generates audit trails for traffic within the network, and can trace the origin of a *single* IP packet delivered by the network in the recent past. We demonstrate that the system is effective, space-efficient (requiring approximately 0.5% of the link capacity per unit time in storage), and implementable in current or next-generation routing hardware. We present both analytic and simulation results showing the system's effectiveness.

1 INTRODUCTION

Today's Internet infrastructure is extremely vulnerable to motivated and well-equipped attackers. Tools are readily available, from covertly exchanged exploit programs to publicly released vulnerability assessment software, to degrade performance or even disable vital network services. The consequences are serious and, increasingly, financially disastrous, as can be seen by all-too-frequent headlines naming the most recent victim of an attack.

While distributed denial of service attacks, typically conducted by flooding network links with large amounts of traffic, are the most widely reported, there are other forms of network attacks. Many other classes of attacks can be conducted with significantly smaller packet flows. In fact, there are a number of widely-deployed operating systems and routers that can be disabled by a single well-targeted packet [13]. To institute accountability for these attacks, the source of individual packets must be identified.

Unfortunately, the anonymous nature of the IP protocol makes it difficult to accurately identify the true source of an IP datagram if the source wishes to conceal it. The network routing infrastructure is stateless and based largely on destination addresses; no entity in an IP network is officially responsible for ensuring the source address is correct. Many routers employ a technique called *ingress filtering* [9] to limit source addresses of IP datagrams from a stub network to addresses belonging to that network, but not all routers have the resources necessary to examine the source address of each incoming packet, and ingress filtering provides no protection on transit networks. Furthermore, spoofed source addresses are legitimately used by network address translators (NATs), Mobile IP, and various unidirectional link technologies such as hybrid satellite architectures.

Accordingly, a well-placed attacker can generate offending IP packets that appear to have originated from almost anywhere. Furthermore, while techniques such as ingress filtering increase the difficulty of mounting an attack, transit networks are dependent upon their peers to perform the appropriate filtering. This interdependence is clearly unacceptable from a liability perspective; each motivated network must be able to secure itself independently.

Systems that can reliably trace individual packets back to their sources are a first and important step in making attackers (or, at least, the systems they use) accountable. There are a number of significant challenges in the construction of such a tracing system including determining which packets to trace, maintaining privacy (a tracing system should not adversely impact the privacy of legitimate users), and minimizing cost (both in router time spent tracking rather than forwarding packets, and in storage used to keep information).

We have developed a *Source Path Isolation Engine* (SPIE) to enable IP *traceback*, the ability to identify the source of a particular IP packet given a copy of the packet to be traced, its destination, and an approximate time of receipt. Historically, tracing individual packets has required prohibitive amounts of memory; one of SPIE's key

[†]Alex C. Snoeren is also with the MIT Laboratory for Computer Science (snoeren@lcs.mit.edu).

[‡]Luis A. Sanchez was with BBN Technologies; he is now with Megisto Systems, Inc. (lsanchez@megisto.com).

This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) under contract No. N66001-00-C-8038. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied.

SIGCOMM'01, August 27-31, 2001, San Diego, California, USA.

innovations is to reduce the memory requirement (down to 0.5% of link bandwidth per unit time) through the use of Bloom filters. By storing only packet digests, and not the packets themselves, SPIE also does not increase a network's vulnerability to eavesdropping. SPIE therefore allows routers to efficiently determine if they forwarded a particular packet within a specified time interval while maintaining the privacy of unrelated traffic.

The rest of this paper examines SPIE in detail. We begin by defining the problem of IP traceback in section 2, and articulate the desired features of a traceback system. We survey previous work in section 3, relating their feature sets against our requirements. Section 4 describes the digesting process in detail. Section 5 presents an overview of the SPIE architecture, while section 6 offers a practical implementation of the concepts. Section 7 provides both analytic and simulation results evaluating SPIE's traceback success rates. We discuss the issues involved in deploying SPIE in section 8 before concluding in section 9 with a brief look at future work.

2 IP TRACEBACK

The concept of IP traceback is not yet well defined. In an attempt to clarify the context in which SPIE was developed, this section presents a detailed and rather formal definition of traceback. We hope that presenting a strawman definition of traceback will also help the community better evaluate different traceback schemes.

In order to remain consistent with the terminology in the literature, we will consider a packet of interest to be nefarious, and term it an *attack packet*; similarly, the destination of the packet is a *victim*. We note, however, that there are many reasons to trace the source of a packet; many packets of interest are sent with no ill intent whatsoever.

2.1 Assumptions

There are several important assumptions that a traceback system should make about a network and the traffic it carries:

- Packets may be addressed to more than one physical host
- Duplicate packets may exist in the network
- Routers may be subverted, but not often
- Attackers are aware they are being traced
- The routing behavior of the network may be unstable
- The packet size should not grow as a result of tracing
- End hosts may be resource constrained
- Traceback is an infrequent operation

The first two assumptions are simply characteristics of the Internet Protocol. IP packets may contain a multicast or broadcast address as their destination, causing the routing infrastructure to duplicate them internally. An attacker can also inject multiple, identical packets itself, possibly at multiple locations. A tracing system must be prepared for a situation where there are multiple sources of the same (identical) packet, or a single source of multiple (also typically identical) packets.

The next two assumptions speak to the capabilities of the attacker(s). An attacker may gain access to routers along (or adjacent to) the path from attacker to victim by a variety of means. Further, a sophisticated attacker is aware of the characteristics of the network, including the possibility that the network is capable of tracing an attack. The traceback system must not be confounded by a motivated attacker who subverts a router with the intent to subvert the tracing system.

The instability of Internet routing is well known [15] and its implications for tracing are important. Two packets sent by the same host to the same destination may traverse wildly different paths. As a result, any system that seeks to determine origins using multi-packet analysis techniques must be prepared to make sense of divergent path information.

The assumption that the packet size should not grow is probably the most controversial. There are a number of protocols today that cause the packet size to grow, for example technologies that rely on IP tunnels, such as IPsec and mobile IP. However, increasing the packet size causes MTU problems and increases overhead sharply (each byte of additional overhead reduces system bandwidth by about 1%, given the average packet size of about 128 bytes). It follows that an efficient traceback system should not cause packet size to grow.

We assume that an end host, and in particular the victim of an attack, may be resource-poor and unable to maintain substantial additional administrative state regarding the routing state or the packets it has previously received. This assumption comes from the observed rise in special purpose devices such as microscopes, cameras, and printers that are attached to the Internet but have few internal resources other than those devoted to performing their primary task.

The final assumption that traceback queries are infrequent has important design implications. It implies queries can be handled by a router's control path, and need not be dealt with on the forwarding path at line speed. While there may be auditing tasks associated with packet forwarding to support traceback that must be executed while forwarding, the processing of the audit trails is infrequent with respect to their generation.

2.2 The goal

Ideally, a traceback system should be able to identify the source of any piece of data sent across the network. In an IP framework, the packet is the smallest atomic unit of data. Any smaller division of data (a byte, for instance) is contained within a unique packet. Hence an optimal IP traceback system would precisely identify the source of an arbitrary IP packet. Any larger data unit or stream can be isolated by searching for any particular packet containing data within the stream.[1]

As with any auditing system, a traceback system can only be effective in networks in which it has been deployed. Hence we consider the source of a packet to be one of:

- The ingress point to the traceback-enabled network
- The actual host or network of origin
- One or more compromised routers within the enabled network

If one assumes that any router along the path may be co-opted to assist in concealing a packet's source, it becomes obvious that one

[1] Indeed, we would argue that it is desirable to trace the individual packets within a stream because the individual packets may have originated at different sites (meeting only at the victim) and are likely to have followed different paths through the network.

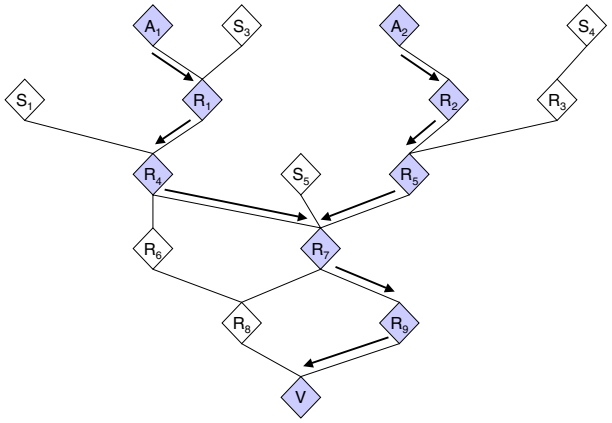

Figure 1: An attack graph containing attack paths for two identical packets injected by A_1 and A_2 and received by the victim, V. The arrows indicate links traversed by the packet; nodes on an attack path are shaded: $\{A_1, R_1, R_4, R_7, R_9, V\}$ and $\{A_2, R_2, R_5, R_7, R_9, V\}$.

must attempt to discern not only the packet's source, but its entire path through the network. If a path can be traced through any number of non-subverted routers, then it must terminate at either the source of the flow or pass through a subverted router which can be considered to be a co-conspirator and treated appropriately. Hence, we are interested in constructing an *attack path*, where the path consists of each router traversed by the packet on its journey from source to the victim. Because conspiring routers can fabricate trace information, the path can only be guaranteed to be accurate on the portion from the victim to the first source—multiple sources may be identified if routers are subverted. Further, since multiple, indistinguishable packets may be injected into the network from different sources in the general case, a traceback system should construct an *attack graph* composed of the attack paths for every instance of the attack packet that arrived at the victim. Figure 1 depicts the network as viewed by the victim and a particular attack graph for that victim.

An attack graph may contain false positives in the presence of subverted routers; that is, the attack graph may identify sources that did not actually emit the packet. We argue this is an unavoidable consequence of admitting the possibility of subverted routers. An ideal traceback system, however, produces no false *negatives* while attempting to minimize false positives; it must never exonerate an attacker by not including the attacker in the attack graph.

Further, when a traceback system is deployed, it must not reduce the privacy of IP communications. In particular, entities not involved in the generation, forwarding, or receipt of the original packet should not be able to gain access to packet contents by either utilizing or as part of participating in the IP traceback system. An ideal IP traceback system must not expand the eavesdropping capabilities of a malicious party.

2.3 Transformations

It is important to note that packets may be modified during the forwarding process. In addition to the standard decrementing of the time to live (TTL) field and checksum recomputation, IP packets may be further transformed by intermediate routers. Packet *transformation* may be the result of valid processing, router error, or malicious intent. A traceback system need not concern itself with packet transformations resulting from error or malicious behavior. Packets resulting from such transformations only need be traced to the point of transformation, as the transforming node either needs to be fixed or can be considered a co-conspirator. An optimum traceback system should trace packets through valid transformations, however, back to the source of the original packet.

Valid packet transformations are defined as a change of packet state that allows for or enhances network data delivery. Transformations occur due to such reasons as hardware needs, network management, protocol requirements, and source request. Based on the transform produced, packet transformations are categorized as follows:

1. *Packet Encapsulation*: A new packet is generated in which the original packet is encapsulated as the payload (e.g., IPsec). The new packet is forwarded to an intermediate destination for de-encapsulation.
2. *Packet Generation*: One or more packets are generated as a direct result of an action by the router on the original packet (e.g. an ICMP Echo Reply sent in response to an ICMP Echo Request). The new packets are forwarded and processed independent of the original packet.

Common packet transformations include those performed by RFC 1812-compliant routers [1] such as packet fragmentation, IP option processing, ICMP processing, and packet duplication. Network address translation (NAT) and both IP-in-IP and IPsec tunneling are also notable forms of packet transformation. Many of these transformations result in an irrecoverable loss of the original packet state due to the stateless nature of IP networks.

A study of wide-area traffic patterns conducted by the Cooperative Association for Internet Data Analysis (CAIDA) found less than 3% of IP traffic undergoes common transformation and IP tunneling [12]. While this study did not encompass all forms of transformation (NAT processing being a notable omission), it seems safe to assume that packet transformations account for a relatively small fraction of the overall IP traffic traversing the Internet today. However, attackers may transmit packets engineered to experience transformation. The ability to trace packets that undergo transformation is, therefore, an essential feature of any viable traceback system.

3 RELATED WORK

There are two approaches to the problem of determining the route of a packet flow: one can audit the flow as it traverses the network, or one can attempt to infer the route based upon its impact on the state of the network. Both approaches become increasingly difficult as the size of the flow decreases, but the latter becomes infeasible when flow sizes approach a single packet because small flows generally have no measurable impact on the network state.

Route inference was pioneered by Burch and Cheswick [5] who considered the restricted problem of large packet flows and proposed a novel technique that systematically floods candidate network links. By watching for variations in the received packet flow due to the restricted link bandwidth, they are able to infer the flow's

route. This requires considerable knowledge of network topology and the ability to generate large packet floods on arbitrary network links.

One can categorize auditing techniques into two classes according to the way in which they balance resource requirements across the network components. Some techniques require resources at both the end host and the routing infrastructure, others require resources only within the network itself. Of those that require only infrastructure support, some add packet processing to the forwarding engine of the routers while others offload the computation to the control path of the routers.

3.1 End-host schemes

Some auditing approaches attempt to distribute the burden by storing state at the end hosts rather than in the network. Routers notify the packet destination of their presence on the route. Because IP packets cannot grow arbitrarily large, schemes have been developed to reduce the amount of space required to send such information. Recently proposed techniques by Savage *et al.* [21] and Bellovin [2] explore in-band and out-of-band signaling, respectively.

Because of the high overhead involved, neither Savage nor Bellovin attempt to provide audit information for every packet. Instead, each employs probabilistic methods that allow sufficiently large packet flows to be traced. By providing partial information on a subset of packets in a flow, auditing routers enable an end host to reconstruct the entire path traversed by the packet flow after receiving a sufficient number of packets belonging to the flow.

The two schemes diverge in the methods used to communicate the information to the end host. Savage *et al.* employ a packet marking scheme that encodes the information in rarely-used fields within the IP header itself. This approach has been improved upon by Song and Perrig to improve the reconstruction of paths and authenticate the encodings [23]. In order to avoid the backwards compatibility issues and increased computation required by the sophisticated encoding schemes employed in the packet marking schemes, Bellovin's scheme (and later extensions by Wu *et al.* [25]) simply sends the audit information in an ICMP message.

3.2 Infrastructure approaches

End-host schemes require the end hosts to log meta data in case an incoming packet proves to be offensive. Alternatively, the network itself can be charged with maintaining the audit trails.

The obvious approach to auditing packet flow is simply to log packets at various points throughout the network and then use appropriate extraction techniques to discover the packet's path through the network. Logging requires no computation on the router's fast path and, thus, can be implemented efficiently in today's router architecture. Sager suggests such a monitoring approach [19]. However, the effectiveness of the logs is limited by the amount of space available to store them. Given today's link speeds, packet logs quickly grow to intractable sizes, even over relatively short time frames. An OC-192 link is capable of transferring 1.25GB per second. If one allows 60 seconds to conduct a query, a router with 16 links would require 1.2TB of high-speed storage.

These requirements can be reduced by sampling techniques similar to those of the end-host schemes, but down-sampling reduces the probability of detecting small flows and does not alleviate the security issues raised by storing complete packets in the router. The ability of an attacker to break into a router and capture terrabytes of actual traffic has severe privacy implications.

Alternatively, routers can be tasked to perform more sophisticated auditing in real time, extracting a smaller amount of information as packets are forwarded. Many currently available routers support *input debugging*, a feature that identifies on which incoming port a particular outgoing packet (or set of packets) of interest arrived. Since no history is stored, however, this process must be activated before an attack packet passes by. Furthermore, due to the high overhead of this operation on many popular router architectures, activating it may have adverse effects on the traffic currently being serviced by the router.

3.3 Specialized routing

One of the main problems with the link testing or logging methods above is the large amount of repetition required. A trace is conducted in a hop-by-hop fashion requiring a query at each router along the way. Once the incoming link or links have been identified, the process must be repeated at the upstream router.

Several techniques have been developed to streamline and automate this process. Some ISPs have developed their own ad hoc mechanisms for automatically conducting input debugging across their networks. Schnackenberg *et al.* [22] propose a special Intruder Detection and Isolation Protocol (IDIP) to facilitate interaction between routers involved in a traceback effort. IDIP does not specify how participating entities should track packet traffic; it simply requires that they be able to determine whether or not they have seen a component of an attack matching a certain description. Even with automated tools, however, each router in the ISP must support input debugging or logging which are not common in today's high-speed routers for reasons discussed above.

In order to avoid this requirement, Stone [24] suggests constructing an overlay network connecting all the edge routers of an ISP. By using a deliberately simple topology of specialized routers, suspicious flows can be dynamically rerouted across the special tracking network for analysis. This approach has two major shortcomings. First, the attack must be sufficiently long-lived to allow the ISP to effect the rerouting before the relevant flow terminates. Second, the routing change is perceptible by the attacker, and an especially motivated attacker may be able to escape detection by taking appropriate action. While techniques exist to hide precisely what changed about the route, changes in layer-three topology are hard to mask.

4 PACKET DIGESTING

SPIE, the Source Path Isolation Engine, uses auditing techniques to support the traceback of individual packets while reducing the storage requirements by several orders of magnitude over current log-based techniques [19]. Traffic auditing is accomplished by computing and storing 32-bit packet digests rather than storing the packets themselves. In addition to reducing storage requirements, storing packet digests instead of the actual packet contents preserves traffic confidentiality by preventing SPIE from being used as a tool for eavesdropping.

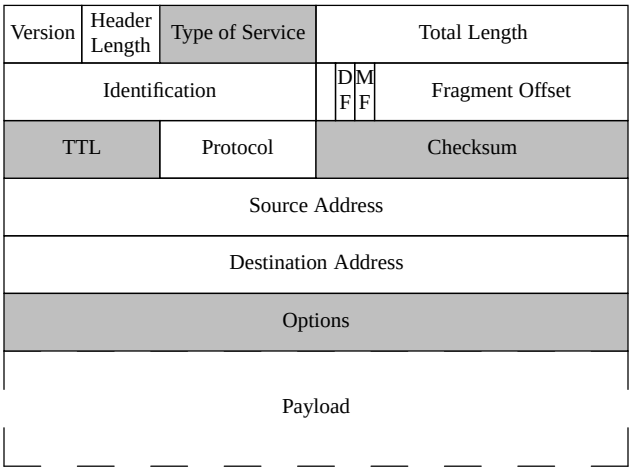

Figure 2: The fields of an IP packet. Fields in gray are masked out before digesting, including the Type of Service, Time to Live (TTL), IP checksum, and IP options fields.

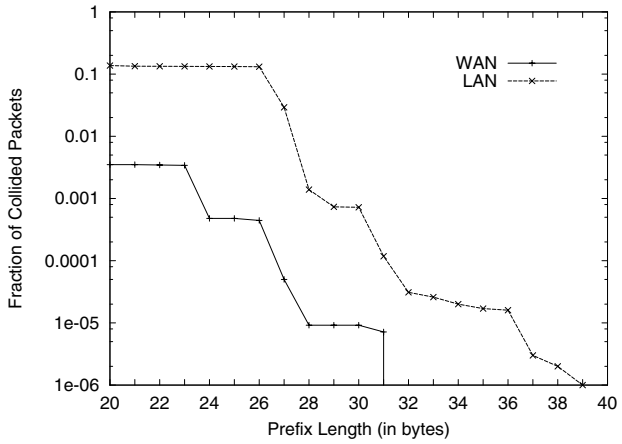

Figure 3: The fraction of packets that collide as a function of prefix length. The WAN trace represents 985,150 packets (with 5,801 duplicates removed) collected on July 20, 2000 at the University of Florida OC-3 gateway [14]. The LAN trace consists of one million packets (317 duplicates removed) observed on an Ethernet segment at the MIT Lab for Computer Science.

4.1 Hash input

The packet content used as input to the hash function must uniquely represent an IP packet and enable the identification of the packet across hops in the forwarding path. At the same time, it is desirable to limit the size of the hash input both for performance and for reasons discussed below (c.f. section 5.3). Duffield and Grossglauser encountered similar requirements while sampling a subset of forwarded packets in an attempt to measure traffic flows [7]. We use a similar approach, masking variant packet content and selecting an appropriate-length prefix of the packet to use as input to the digesting function. Our choice of invariant fields and prefix length is slightly different, however.

Figure 2 shows an IP packet and the fields included by the SPIE digesting function. SPIE computes digests over the invariant portion of the IP header and the first 8 bytes of the payload. Frequently modified header fields are masked prior to digesting. Note that beyond the obvious fields (TTL, TOS, and checksum), certain IP options cause routers to rewrite the option field at various intervals. To ensure a packet appears identical at all steps along its route, SPIE masks or compensates for these fields when computing the packet digests. It is important to note that the invariant IP fields used for SPIE digesting may occasionally be modified by a packet transform (c.f. section 5.3).

Our research indicates that the first 28 invariant bytes of a packet (masked IP header plus the first 8 bytes of payload) are sufficient to differentiate almost all non-identical packets. Figure 3 presents the rate of packet collisions for an increasing prefix length for two representative traces: a WAN trace from an OC-3 gateway router, and a LAN trace from an active 100Mb Ethernet segment. (Results were similar for traces across a number of sites.) Two unique packets which are identical up to the specified prefix length are termed a collision. A 28-byte prefix results in a collision rate of approximately 0.00092% in the wide area and 0.139% on the LAN.

Unlike similar results reported by Duffield and Grossglauser [7, fig. 4], our results include only unique packets; exact duplicates were removed from the packet trace. Close inspection of packets in the wide area with identical prefixes indicates that packets with matching prefix lengths of 22 and 23 bytes are ICMP Time Exceeded error packets with the IP identification field set to zero. Similarly, packets with matching prefixes between 24 and 31 bytes in length are TCP packets with IP identifications also set to zero which are first differentiated by the TCP sequence number or acknowledgment fields.[2]

The markedly higher collision rate in the local area is due to the lack of address and traffic diversity. This expected result does not significantly impact SPIE's performance, however. LANs are likely to exist at only two points in an attack graph: immediately surrounding the victim and the attacker(s). False positives on the victim's local network can be easily eliminated from the attack graph—they likely share the same gateway router in any event. False positives at the source are unlikely if the attacker is using spoofed source addresses, as this provides the missing diversity in attack traffic, and remain in the immediate vicinity of the true attacker by definition. Hence, for the purposes of SPIE, IP packets are effectively distinguished by the first 28 invariant bytes of the packet.

4.2 Bloom filters

Storing the set of digests for the traffic forwarded by the router would require massive amounts of storage. Instead, SPIE uses a space-efficient data structure known as a Bloom filter to record packet digests [4]. A Bloom filter computes k distinct packet digests for each packet using independent uniform hash functions, and uses the n-bit results to index into a 2^n-sized bit array. The array is initialized to all zeros, and bits are set to one as packets are received. Figure 4 depicts a Bloom filter with k hash functions.

Membership tests can be conducted simply by computing the k digests on the packet in question and checking the indicated bit posi-

[2]Further investigation indicates a number of current operating systems, including recent versions of Linux, frequently set the IP ID to zero.

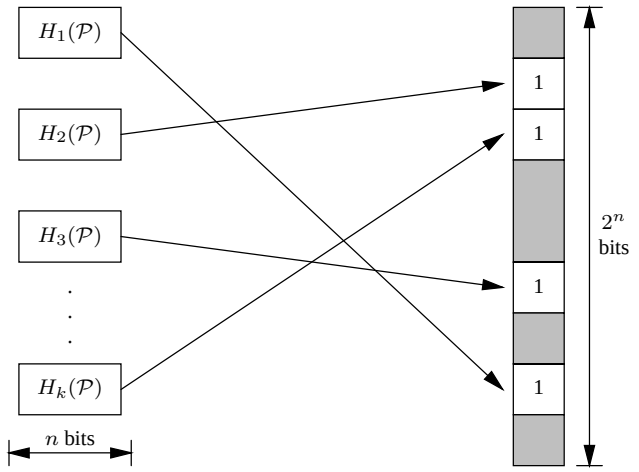

Figure 4: For each packet received, SPIE computes k independent n-bit digests, and sets the corresponding bits in the 2^n-bit digest table.

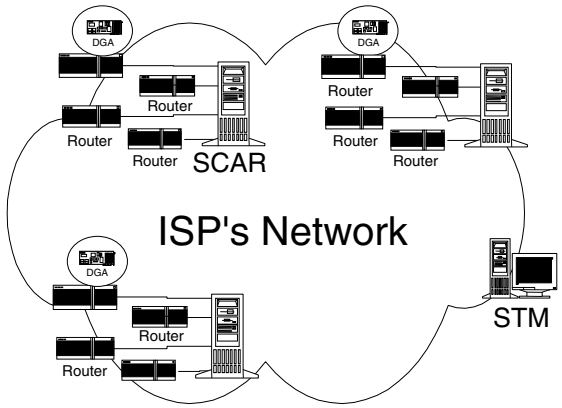

Figure 5: The SPIE network infrastructure, consisting of Data Generation Agents (DGAs), SPIE Collection and Reduction Agents (SCARs), and a SPIE Traceback Manager (STM).

tions. If any one of them is zero, the packet was not stored in the table. If, however, all the bits are one, it is highly likely the packet was stored. It is possible that some set of other insertions caused all the bits to be set, creating a *false positive*, but the rate of such false positives can be controlled [8].

4.3 Hash functions

SPIE places three major restrictions on the family of hash functions, $\mathcal{F}$, used in its Bloom filters. First, each member function must distribute a highly correlated set of input values (IP packet prefixes), $\mathcal{P}$, as uniformly as possible over the hash's result value space. That is, for a hash function $H : \mathcal{P} \rightarrow 2^m$ in $\mathcal{F}$, and distinct packets $x \neq y \in \mathcal{P}$, $\Pr[H(x) = H(y)] = 1/(2^m)$. This is a standard property of good hash functions.

SPIE further requires that the event that two packets collide in one hash function ($H(x) = H(y)$ for some H) be independent of collision events in any other functions ($H'(x) = H'(y), H' \neq H$). Intuitively, this implies false positives at one router are independent of false positives at neighboring routers. Formally, for any function $H \in \mathcal{F}$ chosen at random independently of the input packets x and y, $\Pr[H(x) = H(y)] = 2^{-m}$ with high probability. Such hash families, called universal hash families, were first defined by Carter and Wegman [6] and can be implemented in a variety of fashions [3, 10, 11].

Finally, member functions must be straightforward to compute at high link speeds. This requirement is not impractical because SPIE hash functions do not require any cryptographic "hardness" properties. That is, it does not have to be difficult to generate a valid input packet given a particular hash value. Being able to create a packet with a particular hash value enables three classes of attacks, all of which are fairly benign. One attack would ensure that all attack packets have the same fingerprint in the Bloom filter at some router (which is very difficult since there are multiple, independent hashes at each router), but this merely elicits a packet trace that reveals a larger set of systems from which the attacker can attack. Another attack is to ensure all attack packets have different fingerprints, but that is the common case already. The third, and most difficult attack, is to create an attack packet with the same fingerprint as another, non-attack packet. In general, this attack simply yields one more false-positive path, usually only for one hop (as the hash functions change at each hop).

5 SOURCE PATH ISOLATION ENGINE

SPIE-enhanced routers maintain a cache of packet digests for recently forwarded traffic. If a packet is determined to be offensive by some intrusion detection system (or judged interesting by some other metric), a query is dispatched to SPIE which in turn queries routers for packet digests of the relevant time periods. The results of this query are used in a simulated reverse-path flooding (RPF) algorithm to build an attack graph that indicates the packet's source(s).

5.1 Architecture

The tasks of packet auditing, query processing, and attack graph generation are dispersed among separate components in the SPIE system. Figure 5 shows the three major architectural components of the SPIE system. Each SPIE-enhanced router has a Data Generation Agent (DGA) associated with it. Depending upon the type of router in question, the DGA can be implemented and deployed as a software agent, an interface card plug to the switching background bus, or a separate auxiliary box connected to the router through some auxiliary interface.

The DGA produces packet digests of each packet as it departs the router, and stores the digests in bit-mapped *digest tables*. The tables are paged every so often, and represent the set of traffic forwarded by the router for a particular interval of time. Each table is annotated with the time interval and the set of hash functions used to compute the packet digests over that interval. The digest tables are stored locally at the DGA for some period of time, depending on the resource constraints of the router. If interest is expressed in the traffic data for a particular time interval, the tables are transferred to a SPIE Collection and Reduction (SCAR) agent for longer-term storage and analysis.

SCARs are responsible for a particular region of the network, serving as data concentration points for several routers. SCARs monitor and record the topology of their area and facilitate traceback of any packets that traverse the region. Due to the complex topologies of today's ISPs, there will typically be several SCARs distributed over an entire network. Upon request, each SCAR produces an attack graph for its particular region. The attack graphs from each SCAR are grafted together to form a complete attack graph by the SPIE Traceback Manager (STM).

The STM controls the whole SPIE system. The STM is the interface to the intrusion detection system or other entity requesting a packet trace. When a request is presented to the STM, it verifies the authenticity of the request, dispatches the request to the appropriate SCARs, gathers the resulting attack graphs, and assembles them into a complete attack graph. Upon completion of the traceback process, STMs reply to intrusion detection systems with the final attack graph.

5.2 Traceback processing

Before the traceback process can begin, an attack packet must be identified. Most likely, an intrusion detection system (IDS) will determine that an exceptional event has occurred and provide the STM with a packet, $\mathcal{P}$, victim, V, and time of attack, T. SPIE places two constraints on the IDS: the victim must be expressed in terms of the last-hop router, not the end host itself, and the attack packet must be identified in a timely fashion. The first requirement provides the query process with a starting point; the latter stems from the fact that traceback must be initiated before the appropriate digest tables are overwritten by the DGAs. This time constraint is directly related to the amount of resources dedicated to the storage of traffic digests. (We discuss timing and resource tradeoffs in section 7).

Upon receipt of traceback request, the STM cryptographically verifies its authenticity and integrity. Any entity wishing to employ SPIE to perform a traceback operation must be properly authorized in order to prevent denial of service attacks. Upon successful verification, the STM immediately asks all SCARs in its domain to poll their respective DGAs for the relevant traffic digests. Time is critical because this poll must happen while the appropriate digest tables are still resident at the DGAs. Once the digest tables are safely transferred to SCARs, the traceback process is no longer under real-time constraints.

Beginning at the SCAR responsible for the victim's region of the network, the STM sends a query message consisting of the packet, egress point, and time of receipt. The SCAR responds with a partial attack graph and the packet as it entered the region (it may have been transformed, possibly multiple times, within the region). The attack graph either terminates within the region managed by the SCAR, in which case a source has been identified, or it contains nodes at the edge of the SCAR's network region, in which case the STM sends a query (with the possibly-transformed packet) to the SCAR abutting that edge node.

This process continues until all branches of the attack graph terminate, either at a source within the network, or at the edge of the SPIE system. The STM then constructs a composite attack graph which it returns to the intrusion detection system.

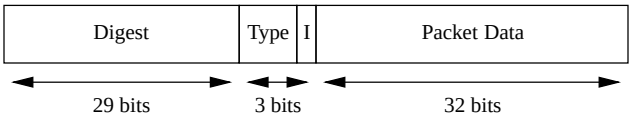

Figure 6: A Transform Lookup Table (TLT) stores sufficient information to invert packet transformations at SPIE routers. The table is indexed by packet digest, specifies the type of transformation, and stores any irrecoverable packet data.

5.3 Transformation processing

IP packets may undergo valid transformation while traversing the network, and SPIE must be capable of tracing through such transformations. In particular, SPIE must be able to reconstruct the original packet from the transformed packet. Unfortunately, many transformations are not invertible without additional information due to the stateless nature of IP networks. Consequently, sufficient packet data must be recorded by SPIE at the time of transformation such that the original packet is able to be reconstructed.

The packet data chosen as input to the digesting function determines the set of packet transformations SPIE must handle—SPIE need only consider transformations that modify fields used as input to the digest function. SPIE computes digests over the IP header and the first eight bytes of the packet payload but masks out (or omits in the case of IP options) several frequently updated fields before digesting, as shown in figure 2 of section 4. This hides most hop-by-hop transformations from the digesting function, but forces SPIE to explicitly handle each of the following transformations: fragmentation, network address translation (NAT), ICMP messages, IP-in-IP tunneling, and IP security (IPsec).

Recording the information necessary to reconstruct the original packet from a transformed packet requires additional resources. Fortunately for SPIE, the circumstances that cause a packet to undergo a transformation will generally take that packet off of the fast path of the router and put it onto the control path, relaxing the timing requirements. The router's memory constraints remain unchanged, however; hence, transformation information must be stored in a scalable and space-efficient manner.

5.3.1 Transform lookup table

Along with each packet digest table collected at a DGA, SPIE maintains a corresponding transform table for the same interval of time called a *transform lookup table*, or TLT. Each entry in the TLT contains three fields. The first field stores a digest of the transformed packet. The second field specifies the type of transformation—three bits are sufficient to uniquely identify the transformation type among those supported by SPIE. The last field contains a variable amount of packet data the length of which depends upon the type of transformation being recorded.

For space efficiency, the data field is limited to 32 bits. Some transformations, such as network address translation, may require more space. These transformations utilize a level of indirection—one bit of the transformation type field is reserved as an *indirect* flag. If the indirect, or I, flag is set, the third field of the TLT is treated as a pointer to an external data structure which contains the information necessary to reconstruct the packet.

The indirect flag can also be used for flow caching. In many cases, packets undergoing a particular transformation are related. In such cases, it is possible to reduce the storage requirements by suppressing duplicate packet data, instead referencing a single copy of the required data that can be used to reconstruct any packet in the flow. Such a scheme requires, however, that the SPIE-enabled router itself be capable of flow caching, or at least identification, so that the packets within the flow can be correlated and stored appropriately.

In order to preserve alignment, it is likely efficient implementations would store only 29 bits of the packet digest resulting in 64-bit wide TLT entries. This width implies eight distinct packet digests will map to the same TLT entry. The relative rarity of packet transformations [12], the sparsity of the digest table, and the uniformity of the digesting function combine to make collisions extremely rare in practice. Assuming a digest table capacity of roughly 3.2Mpkts (16Mb SRAM, see section 7.2) and a transformation rate of 3%, the expected collision rate is approximately 1:5333 packets. Even if a collision occurs, it simply results in an additional possible transformation of the queried packet. Each transformation is computed (including the null transformation) and traceback continues. Incorrectly transformed packets likely will not exist at neighboring routers and, thus, will not contribute any false nodes to the attack graph.

5.3.2 Special-purpose gateways

Some classes of packet transformations, notably NAT and tunneling, are often performed on a large fraction of packets passing through a particular gateway. The transform lookup table would quickly grow to an unmanageable size in such instances; hence, SPIE considers the security gateway or NAT functionality of routers as a separate entity. Standard routing transformations are handled as above, but special purpose gateway transformations require a different approach to transformation handling. Transformations in these types of gateways are generally computed in a stateful way (usually based on a static rule set); hence, they can be inverted in a similar fashion. While the details are implementation-specific, inverting such transformations is straightforward; we do not consider it here.

5.3.3 Sample transformations

A good example of transformation is packet fragmentation. To avoid needing to store any of the packet payload, SPIE supports traceback of only the first packet fragment. Non-first fragments may be traced to the point of fragmentation which, for fragment-based attacks [13], is the attacker. (If only a subset of the fragments is received by the victim the packet cannot be reassembled; hence, the only viable attack is a denial of service attack on the reassembly engine. But, if the fragmentation occurs within the network itself, an attacker cannot control which fragments are received by the victim so the victim will eventually receive a first fragment to use in traceback.) Packet data to be recorded includes the total length, fragment offset, and more fragments (MF) field. Since properly-behaving IP routers cannot create fragments with less than 8 bytes of payload information [17], SPIE is always able to invert fragmentation and construct the header and at least 64 bits of payload of the pre-fragmented packet which is sufficient to continue traceback.

Observe that SPIE never needs to record any packet payload information. ICMP transformations can be inverted because ICMP error messages always include at least the first 64 bits of the offending packet [16]. Careful readers may be concerned that encapsulation cannot be inverted if the encapsulated packet is subsequently fragmented and the fragments containing the encapsulated IP header and first 64 bits of payload are not available. While this is strictly true, such transformations need to be inverted only in extreme cases as it takes a very sophisticated attacker to cause a packet to be first encapsulated, then fragmented, and then ensure fragment loss. If all the fragments are received, the original header can be extracted from the reassembled payload. It seems extremely difficult for an attacker to insure that packet fragments are lost. It can cause packet loss by flooding the link, but to do so requires sending such a large number of packets that it is extremely likely that all the fragments for at least one packet will be successfully received by the decapsulator for use in traceback.

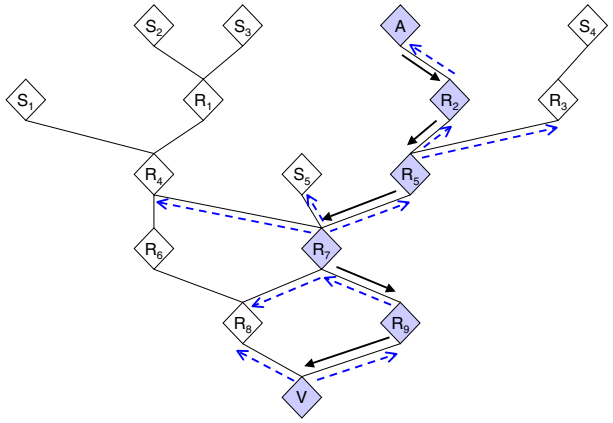

Figure 7: Reverse path flooding, starting at the victim's router, V, and proceeding backwards toward the attacker, A. Solid arrows represent the attack path; dashed arrows are SPIE queries. Queries are dropped by routers that did not forward the packet in question.

5.4 Graph construction

Each SCAR constructs a subgraph using topology information about its particular region of the network. After collecting the digest tables from all of the routers in its region, a SCAR simulates reverse-path flooding (RPF) by examining the digest tables in the order they would be queried if an actual reverse path flood was conducted on the topology that existed at the time the packet was forwarded. Figure 7 shows how reverse-path flooding would discover the attack path from V to A, querying routers $R_8, R_9, R_7, R_4, S_5, R_5$, and R_2 along the way. It is important to note that the routers are *not* actually queried—the SCAR has already cached all the relevant hash digests locally.

In order to query each router, a SCAR computes the appropriate set of digests as indicated by the table, and then consults the table for membership. If an entry exists for the packet in question, the router is considered to have forwarded the packet. The SCAR adds the current node to the attack graph and moves on to each of its neighbors (except, of course, the neighbor already queried). If, however, the digest is not found in the table, it may be necessary to search the digest table for the previous time period. Depending on the link latency between routers, SCARs may need to request multiple di-

gest tables from each router in order to assure they have the digest for the appropriate time frame. Once a digest is located, the packet arrival time is always considered to be the latest possible time in the interval. This insures the packet must have been seen at an earlier time at adjacent routers.

If the packet is not found in any of the digest tables for the relevant time period, that particular branch of the search tree is terminated and searching continues at the remaining routers. A list of previously visited nodes is kept at all times, and cycles are pruned to assure termination.

The result of this procedure is a connected graph containing the set of nodes believed to have forwarded the packet toward the victim. Assuming correct operation of the routers, this graph is guaranteed to be a superset of the actual attack graph. But due to digest collisions, there may be nodes in the attack graph that are not in the actual attack graph. We call these nodes *false positives* and base the success of SPIE on its ability to limit the number of false positives contained in a returned attack graph.

6 PRACTICAL IMPLEMENTATION

For our FreeBSD SPIE prototype, we simulate a universal hash family using MD5 [18]. A random member is defined by selecting a random input vector to prepend to each packet. The properties of MD5 ensure that the digests of identical packets with different input vectors are independent. The 128-bit output of MD5 is then considered as four independent 32-bit digests which can support Bloom filters of dimension up to four. Router implementations requiring higher performance are likely to prefer other universal hash families specifically tailored to hardware implementation [11]. A simple family amenable to fast hardware implementation could be constructed by computing a CRC modulo a random member of the set of indivisible polynomials over Z_{2^k}.

In order to ensure hash independence, each router periodically generates a set of k independent input vectors and uses them to select k digest functions needed for the Bloom filter from the family of universal hashes. These input vectors are computed using a pseudo-random number generator which is independently seeded at each router. For increased robustness against adversarial traffic, the input vectors are changed each time the digest table is paged, resulting in independence not only across routers but also across time periods.

The size of the digest bit vector, or *digest table*, varies with the total traffic capacity of the router; faster routers need larger vectors for the same time period. Similarly, the optimum number of hash functions varies with the size of the bit vector. Routers with tight memory constraints can compute additional digest functions and provide the same false-positive rates as those who compute fewer digests but provide a larger bit vector.

Figure 8 depicts a possible implementation of a SPIE Data Generation Agent in hardware for use on high-speed routers. A full discussion of the details of the architecture and an analysis of its performance were presented previously [20]. Briefly, each interface card in the router is outfitted with an Interface Tap which computes multiple independent digests of each packet as it is forwarded. These digests are passed to a separate SPIE processor (implemented either in a line card form factor or as an external unit) which stores them as described above in digest tables for specific time periods.

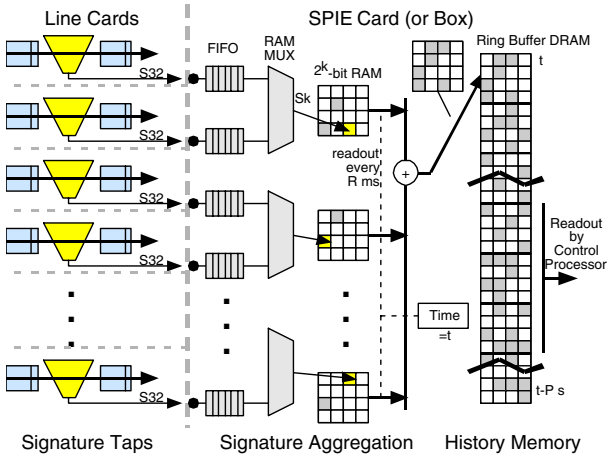

Figure 8: A sample SPIE DGA hardware implementation for high-speed routers.

As time passes, the forwarded traffic will begin to fill the digest tables and they must be paged out before they become over-saturated, resulting in unacceptable false-positive rates. The tables are stored in a history buffer implemented as a large ring buffer. Digest tables can then be transferred by a separate control processor to SCARs while they are stored in the ring buffer.

7 ANALYSIS

There are several tradeoffs involved when determining the optimum amount of resources to dedicate to SPIE on an individual router or the network as a whole. SPIE's resource requirements can be expressed in terms of two quantities: the number of packet digest functions used by the Bloom filter, and the amount of memory used to store packet digests. Similarly, SPIE's performance can be characterized in two orthogonal dimensions. The first is the length of time for which packet digests are kept. Queries can only be issued while the digests are cached; unless requested by a SCAR within a reasonable amount of time, the DGAs will discard the digest tables in order to make room for more recent ones. The second is the accuracy of the candidate attack graphs which can be measured in the number of false positives in the graph returned by SPIE.

Both of these metrics can be controlled by adjusting operational parameters. In particular, the more memory available for storing packet digests, the longer the time queries can be issued. Similarly, digest tables with lower false-positive rates yield more accurate attack graphs. Hence, we wish to characterize the performance of SPIE in terms of the amount of available memory and digest table performance.

7.1 False positives

We first relate the rate of false positives in an attack graph to the rate of false positives in an individual digest table. This relationship depends on the actual network topology and traffic being forwarded at the time. We can, however, make some simplifying assumptions in order to derive an upper bound on the number of false positives as a function of digest table performance.

7.1.1 Theoretical bounds

Suppose, for example, each router whose neighbors have degree at most d ensures its digest tables have a false-positive rate of at most $P = p/d$, where $0 \leq p/d \leq 1$ (p is just an arbitrary tuning factor). A simplistic analysis shows that for any true attack graph G with n nodes, the attack graph returned by SPIE will have at most $np/(1-p)$ extra nodes in expectation.

The false-positive rate of a digest table varies over time, depending on the traffic load at the router and the amount of time since it was paged. Similarly, if the tables are paged on a strict schedule based on maximum link capacity, and the actual traffic load is less, digest tables will never reach their rated capacity. Hence, the analytic result is a worst case bound since the digest table performs strictly better while it is only partially full. Furthermore, our analysis assumes the set of neighbors at each node is disjoint which is not true in real networks. It seems reasonable to expect, therefore, that the false-positive rate over real topologies with actual utilization rates would be substantially lower.

For the purposes of this discussion, we arbitrarily select a false-positive rate of $n/7$, resulting in no more than 5 additional nodes in expectation for a path length of over 32 nodes (approaching the diameter of the Internet) according to our theoretical model. Using the bound above, $p = 1/8$ seems a reasonable starting point and we turn to considering its effectiveness in practice.

7.1.2 Simulation results

In order to relate false-positive rate to digest table performance in real topologies, we have run extensive simulations using the actual network topology of a national tier-one ISP made up of roughly 70 backbone routers with links ranging from T-1 to OC-3. We obtained a topology snapshot and average link utilization data for the ISP's network backbone for a week-long period toward the end of 2000, sampled using periodic SNMP queries, and averaged over the week.

We simulated an attack by randomly selecting a source and victim, and sending 1000 attack packets at a constant rate between them. Each packet is recorded by every intermediate router along the path from source to destination. A traceback is then simulated starting at the victim router and (hopefully) proceeding toward the source. Uniformly distributed background traffic is simulated by selecting a fixed maximum false-positive rate, P, for the digest table at each off-path router. (Real background traffic is not uniform, which would result in slight dependencies in the false-positive rates between routers, but we believe that this represents a reasonable starting point.) In order to accurately model performance with real traffic loads, the effective false-positive rate is scaled by the observed traffic load at each router.

For clarity, we consider a non-transformed packet with only one source and one destination. Preliminary experiments with multiple sources (as might be expected in a distributed denial of service (DDoS) attack) indicate false positives scale linearly with respect to the size of the attack graph, which is the union of the attack paths for each copy of the packet. We do not, however, consider this case in the experiments presented here. (A DDoS attack sending identical packets from multiple sources only aids SPIE in its task. A wise attacker would instead send *distinct* packets from each source, forcing the victim to trace each packet individually.)

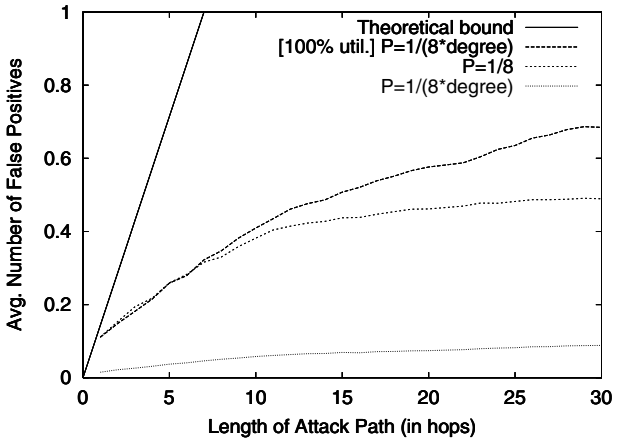

Figure 9: The number of false positives in a SPIE-generated attack graph as a function of the length of the attack path, for $p = 1/8$. The theoretical bound is plotted against three simulation results, two with false-positive rates conditioned on router degree, one without. For the two degree-dependent runs, one considered actual link utilization, while the other assumed full utilization. Each simulation represents the average of 5000 runs using actual topology and utilization data from a national tier-one ISP.

In order to validate our theoretical bound, we have plotted the expected number of false positives as a function of attack path length and digest table performance, $np/(1-p)$ as computed above, and show that in comparison to the results of three simulations on our ISP backbone topology. In the first, we set the maximum digest table false-positive probability to $P = p/d$, as prescribed above. Figure 9 shows a false-positive rate significantly lower than the analytic bound. A significant portion of the disparity results from the relatively low link utilizations maintained by operational backbones (77% of the links in our data set had utilization rates of less than 25%), as can be seen by comparing the results of a second simulation assuming full link utilization. There remains, however, a considerable gap between the analytic bound and simulated performance in network backbones.

The non-linearity of the simulation results indicates there is a strong damping factor due to the topological structure of the network. Intuitively, routers with many neighbors are found at the core of the network (or at peering points), and routers with fewer neighbors are found toward the edge of the network. This suggests false positives induced by core routers may quickly die out as the attack graph proceeds toward less well-connected routers at the edge.

To examine the dependence upon vertex degree, we conducted another simulation. This time, we removed the false-positive rate's dependence upon the degree of the router's neighbors, setting the digest table performance to simply $P = p$ (and returning to actual utilization data). While there is a marked increase in the number of false positives, it remains well below the analytic bound. This somewhat surprising result indicates that despite the analytic bound's dependence on router degree, the hierarchical structure of ISP backbones may permit a relaxation of the coupling, allowing the false positive rate of the digest tables, P, to be set independently of the degree, d, resulting in significant space savings.

7.2 Time and memory utilization

The amount of time during which queries can be supported is directly dependent on the amount of memory dedicated to SPIE. The appropriate amount of time varies depending upon the responsiveness of the method used to identify attack packets. For the purposes of discussion, however, we will assume one minute is a reasonable amount of time in which to identify an attack packet and initiate a traceback. As discussed in section 5.1, once the appropriate digest tables have been moved to SCARs the actual query process can arbitrarily be delayed.

Given a particular length of time, the amount of memory required varies linearly with the total link capacity at the router and can be dramatically affected by the dimension of the Bloom filter in use. Bloom filters are typically described in terms of the number of digesting functions used and the ratio of data items to be stored to memory capacity. The effective false-positive rate for a Bloom filter that uses m bits of memory to store n packets with k digest functions can be expressed as

$$P = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k.$$

Tables providing the effective false-positive rates for various capacities and digesting functions are readily available [8]. For the purposes of discussion, we will consider using a Bloom filter with three digesting functions ($k = 3$) and a capacity factor (m/n) of five, meaning to store n packets, we will use a Bloom filter of size $m = 5n$. Such a filter provides an effective false-positive rate of $P = 0.092$ when full.

While this is well below the value of $1/8$ or 0.125 used in our degree-independent simulations, it is high if digest tables are calibrated with respect to router degree. Luckily, by increasing the number of digesting functions, Bloom filters are able to achieve significantly lower false-positive rates with slight increases in capacity. For instance, a false-positive rate of 0.00314, which corresponds to our degree-dependent simulation, $P = p/d$, with $p = 1/8$ for routers with as many as 40 neighbors, can be achieved using 8 digesting functions and memory factor of only 12—slightly greater than twice what we suggest.

SPIE's memory needs are determined by the number of packets processed. Hence, we consider an average-sized packet of approximately 1000 bits, and describe link speeds in terms of packets per second. We combine this with the Bloom filter factor of 5 from above to compute a rule of thumb: SPIE requires roughly 0.5% of the total link capacity in digest table storage. For a typical low-end router with four OC-3 links, this results in roughly 47MB of storage. On the very high end, a core router with 32 OC-192 links[3] has a maximum capacity of about 640Mpkts/sec which would require roughly 3.125Gb/sec of digest table memory or 23.4GB for one minute's worth of storage. In practice, however, the size of a digest table will be limited by the type of memory required.

Capacity is not the only memory consideration, however—access times turn out to be equally important. Packets must be recorded in the digest table at a rate commensurate with their arrival. Even given an optimistic DRAM cycle time of 50ns per read-modify-write cycle, routers processing more than 20Mpkts/sec (roughly 1 OC-192 link, or 4 OC-48s) require an SRAM digest table. Current SRAM technology limits digest tables to 16Mb which must be paged to SDRAM in order to store a minute's worth of digests as described in section 6. Hence, an entire minute's worth of traffic can only be stored in one (unpaged) digest table at low link speeds.

[3] Current production routers support at most one OC-192 link.

7.3 Timing uncertainties

In the OC-192 scenario described above, 16Mb would hold roughly 5ms of traffic data; hence, the history buffer would store 12,000 individual digest tables. This observation gives rise to another important issue: imperfect timing may cause SPIE to need to examine multiple packet digests at a particular router. The more digests that must be considered, the greater the chance of false positives, so it is advantageous to make the digest tables as large as possible. For reasonable link speeds, the memory access time becomes slow enough that SDRAM can be used which, using current technology, would allow 256Mb digest tables, with a capacity of roughly 50Mpkts.

It may be the case that the approximate packet service time cannot be confined to an interval covered by one digest table. In that case, we expect the false-positive rate to increase linearly with the number of digest tables examined. For high-speed routers, it is especially important to maintain precise timing synchronization between adjacent routers. We have not yet examined the impact of typical NTP clock skew on SPIE's performance, but believe synchronization can be maintained to within a small number of digesting intervals, not significantly impacting our false-positive rate.

8 DISCUSSION

There are several issues that must be dealt with for a SPIE query to succeed. First, traceback operations will often be requested when the network is unstable (likely due to the attack that triggered the traceback); SPIE communications must succeed in a timely fashion even in the face of network congestion and instability. The best solution is to provide SPIE with an out-of-band channel, possibly through either physically or logically separate (ATM VCs) links. But even without private channels, it is still possible to ensure successful transmission by granting sufficient priority to SPIE traffic.

SPIE's usefulness increases greatly with widespread deployment because SPIE can only construct an attack graph for that portion of the packet's path within the SPIE domain. However, it is likely that independent ISPs may lack sufficient levels of technical or political cooperation to unite their SPIE infrastructure. Instead, many ISPs will prefer to have their own STM responsible for all queries within their network. In such a case, one ISP's STM must be granted the authority to issue queries to adjacent ISPs' STMs in order to complete the traceback.

In very rare cases, one may not wish to expose the content of a packet yet still wish to employ SPIE. In such a case, it might be possible to support call-backs from SCARs which would provide the querying IDS with the applicable digesting function and transformation information and ask it to do actual digesting. This is an expensive operation, but the existence of such a case implies the querying IDS has grave cause for concern in the first place and is likely willing to dedicate a great deal of resources to the traceback.

Finally, transformations raise several additional issues, some related to performance, others to policy. In particular, assuming that

packet transformations represent a small percentage of the overall IP traffic traversing a router, an efficient SPIE implementation can easily handle the resource requirements of logging transformation information. Attackers, though, may view packet transformations as a method of denial of service attack on SPIE. The number of transformations that are recorded during a given time interval is bounded by the rate at which the router is able to process the packet transformations. Therefore, SPIE aims to handle packet transformations at a rate equal or greater than the router. As a result, the router rather than SPIE is the bottleneck in processing packet transformations. This task is made easier when one realizes that the vast majority of transformations occur only at low-to-medium speed routers. Sophisticated transformations such as tunneling, NATing, and the like are typically done at customer premises equipment. Further, many ISPs turn off standard transformation handing, often even ICMP processing, at their core routers.

9 CONCLUSION & FUTURE WORK

Developing a traceback system that can trace a single packet has long been viewed as impractical due to the tremendous storage requirements of saving packet data and the increased eavesdropping risks the packet logs posed. We believe that SPIE's key contribution is to demonstrate that single packet tracing is feasible. SPIE has low storage requirements and does not aid in eavesdropping. Furthermore, SPIE is a complete, practical system. It deals with the complex problem of transformations and can be implemented in high-speed routers (often a problem for proposed tracing schemes).

The most pressing challenges for SPIE are increasing the window of time in which a packet may be successfully traced and reducing the amount of information that must be stored for transformation handling. One possible way to extend the length of time queries can be conducted without linearly increasing the memory requirements is by relaxing the set of packets that can be traced. In particular, SPIE can support traceback of large packet flows for longer periods of time in a fashion similar to probabilistic marking schemes—rather than discard packet digests as they expire, discard them probabilistically as they age. For large packet flows, odds are quite high some constituent packet will remain traceable for longer periods of time.

ACKNOWLEDGEMENTS

We are indebted to Walter Milliken for his assistance in ensuring our architecture was implementable in today's high-speed routers, and for designing the initial DGA hardware prototype shown in figure 8. David Karger first pointed out we could generalize our hashing technique through Bloom filters to decrease memory requirements, and Michael Mitzenmacher and Ron Rivest provided advice on appropriate digesting functions. We thank Christine Alvarado, David Andersen, John Jannotti, Allen Miu, and the anonymous reviewers for their feedback on earlier drafts.

References

[1] BAKER, F. Requirements for IP version 4 routers. RFC 1812, IETF, June 1995.

[2] BELLOVIN, S. M. ICMP traceback messages. Internet Draft, IETF, Mar. 2000. `draft-bellovin-itrace-05.txt` (work in progress).

[3] BLACK, J., HALEVI, S., KRAWCZYK, J., KROVETZ, T., AND ROGAWAY, P. UMAC: fast and secure message authentication. In *Proc. Advances in Cryptology — CRYPTO '99* (Aug. 1999), pp. 216–233.

[4] BLOOM, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM 13*, 7 (July 1970), 422–426.

[5] BURCH, H., AND CHESWICK, B. Tracing anonymous packets to their approximate source. In *Proc. USENIX LISA '00* (Dec. 2000).

[6] CARTER, L., AND WEGMAN, M. Universal classes of hash functions. *Journal of Computer and System Sciences* (1979), 143–154.

[7] DUFFIELD, N. G., AND GROSSGLAUSER, M. Trajectory sampling for direct traffic observation. In *Proc. ACM SIGCOMM '00* (Aug. 2000), pp. 271–282.

[8] FAN, L., CAO, P., ALMEIDA, J., AND BRODER, A. Z. Summary cache: a scalable wide-area web cache sharing protocol. *ACM Trans. on Networking 8*, 3 (2000), 281–293.

[9] FERGUSON, P., AND SENIE, D. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2267, IETF, Jan. 1998.

[10] HALEVI, S., AND KRAWCZYK, H. MMH: Software message authentication in the Gbit/second rates. In *Proc. 4th Workshop on Fast Software Encryption* (1997), pp. 172–189.

[11] KRAWCZYK, H. LFSR-Based hashing and authentication. In *Proc. Advances in Cryptology — CRYPTO '94* (Aug. 1994), pp. 129–139.

[12] MCCREARY, S., AND CLAFFY, K. Trends in wide area IP traffic patterns: A view from Ames Internet exchange. In *ITC Specialist Seminar on IP Traffic Modeling, Measurement and Management* (2000).

[13] MICROSOFT CORPORATION. Stop 0A in tcpip.sys when receiving out of band (OOB) data. `http://support.microsoft.com/support/kb/articles/Q143/4/78.asp`.

[14] NATIONAL LABORATORY FOR APPLIED NETWORK RESEARCH (NLANR). Network traffic packet header traces. `http://moat.nlanr.net/Traces/Traces/20000720/FLA-964095596.crl.enc`.

[15] PAXSON, V. End-to-end internet path dynamics. *ACM Trans. on Networking 7*, 3 (1999), 277–292.

[16] POSTEL, J. Internet Control Message Protocol. RFC 792, IETF, Sept. 1981.

[17] POSTEL, J. Internet Protocol. RFC 791, IETF, Sept. 1981.

[18] RIVEST, R. The MD5 message-digest algorithm. RFC 1321, IETF, Apr. 1992.

[19] SAGER, G. Security fun with OCxmon and cflowd. Internet 2 Working Group Meeting, Nov. 1998. `http://www.caida.org/projects/NGI/content/security/1198`.

[20] SANCHEZ, L. A., MILLIKEN, W. C., SNOEREN, A. C., TCHAKOUNTIO, F., JONES, C. E., KENT, S. T., PARTRIDGE, C., AND STRAYER, W. T. Hardware support for a hash-based IP traceback. In *Proc. Second DARPA Information Survivability Conference and Exposition* (June 2001).

[21] SAVAGE, S., WETHERALL, D., KARLIN, A., AND ANDERSON, T. Practical network support for IP traceback. In *Proc. ACM SIGCOMM '00* (Aug. 2000), pp. 295–306.

[22] SCHNACKENBERG, D., DJAHANDARI, K., AND STERNE, D. Infrastructure for intrusion detection and response. In *Proc. First DARPA Information Survivability Conference and Exposition* (Jan. 2000).

[23] SONG, D. X., AND PERRIG, A. Advanced and authenticated marking schemes for IP traceback. In *Proc. IEEE Infocom '01* (Apr. 2001).

[24] STONE, R. CenterTrack: An IP overlay network for tracking DoS floods. In *Proc. USENIX Security Symposium '00* (Aug. 2000).

[25] WU, S. F., ZHANG, L., MASSEY, D., AND MANKIN, A. Intention-driven ICMP trace-back. Internet Draft, IETF, Feb. 2001. `draft-wu-itrace-intention-00.txt` (work in progress).

An End-to-End Approach to Host Mobility

Alex C. Snoeren and Hari Balakrishnan
MIT Laboratory for Computer Science
Cambridge, MA 02139
{snoeren, hari}@lcs.mit.edu

Abstract

We present the design and implementation of an end-to-end architecture for Internet host mobility using dynamic updates to the Domain Name System (DNS) to track host location. Existing TCP connections are retained using secure and efficient connection migration, enabling established connections to seamlessly negotiate a change in endpoint IP addresses without the need for a third party. Our architecture is secure—name updates are effected via the secure DNS update protocol, while TCP connection migration uses a novel set of *Migrate* options—and provides a pure end-system alternative to routing-based approaches such as Mobile IP.

Mobile IP was designed under the principle that fixed Internet hosts and applications were to remain unmodified and only the underlying IP substrate should change. Our architecture requires no changes to the unicast IP substrate, instead modifying transport protocols and applications at the end hosts. We argue that this is not a hindrance to deployment; rather, in a significant number of cases, it allows for an easier deployment path than Mobile IP, while simultaneously giving better performance. We compare and contrast the strengths of end-to-end and network-layer mobility schemes, and argue that end-to-end schemes are better suited to many common mobile applications. Our performance experiments show that handoff times are governed by TCP migrate latencies, and are on the order of a round-trip time of the communicating peers.

1 Introduction

The proliferation of mobile computing devices and wireless networking products over the past decade has made host and service mobility on the Internet an important problem. Delivering data to a mobile host across a network address change without disrupting existing connections can be tackled by introducing a level of indirection in the routing system. This is the approach taken by Mobile IP [27, 29], which deploys a home agent that intercepts packets destined for a host currently away from its home network, and delivers it to the mobile host via a foreign agent in the foreign network. This approach does not require any changes to the fixed (correspondent) hosts in the Internet, but does require changing the underlying IP substrate to effect this *triangle* routing, and an authentication protocol to ensure that connections are not hijacked by a malicious party. Mobile IP is a "pure" routing solution, a network-layer scheme that requires no changes to any higher layer of the Internet protocol stack.

There are many classes of mobile applications [16]: those where other hosts originate connections to a mobile host and can benefit from both host location and handoff support (e.g., a mobile Web server, mobile telephony); those where the mobile host originates all connections, which do not require host location services but can benefit from handoff support (e.g., mail readers, Web browsers); and those where an application-level retry suffices if the network address changes unexpectedly during a short transaction, which need neither to work well (e.g., DNS resolution). We believe that a good end-to-end architecture for host mobility will support all these modes, and empower applications to make the choice best suited to their needs. Our architecture is motivated by, and meets, this goal. It is an end-to-end approach; no changes to the IP substrate are required.

In our mobility architecture, the decision of whether to support transparent connectivity across network address changes (especially useful for mobile servers) or not (not needed for short client-server transactions) is left to the application. While Mobile IP-style, fully-transparent mobility support is general and sufficient for mobile applications, this generality comes at significant cost, complexity, and performance degradation.

To locate mobile hosts as they change their network attachment point, we take advantage of the widely-deployed Domain Name System (DNS) [20] and its ability to support secure dynamic updates [8, 35]. Because most Internet applications resolve hostnames to an IP address at the beginning of a transaction or connection, this approach is viable for initiating new sessions with mobile hosts. When a host changes its network attachment point (IP address), it sends a secure DNS update to one of the name servers in its home domain updating its current location. The name-to-address mappings for these hosts are uncacheable by other domains, so stale bindings are eliminated.

The ability to support continuous communication during periods of mobility without modifying the IP substrate is a more challenging problem. Because TCP is a connection-oriented reliable protocol,

This research was supported in part by DARPA (Grant No. MDA972-99-1-0014), NTT Corporation, and IBM. Alex C. Snoeren is supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

MobiCom 2000 08/2000 Boston, MA

many TCP applications reasonably expect this service model in the face of losses and transient link failures, route changes, or mobility. The two communicating peers must securely negotiate a change in the underlying network-layer IP address and then seamlessly continue communication. Furthermore, an approach that *requires* either communicating peer to learn about the new network-layer address before a move occurs is untenable because network-layer moves may be quite sudden and unpredictable.

We design a new end-to-end TCP option to support the secure migration of an established TCP connection across an IP address change. Using this option, a TCP peer can suspend an open connection and reactivate it from another IP address, transparent to an application that expects uninterrupted reliable communication with the peer. In this protocol, security is achieved through the use of a connection identifier, or *token*, which may be secured by a shared secret key negotiated through an Elliptic Curve Diffie-Hellman (ECDH) key exchange [36] during initial connection establishment. It requires no third party to authenticate migration requests, thereby allowing the end points to use whatever authentication mechanism they choose to establish a trust relationship. Although we only describe details for TCP migration, we find that this idea is general and can be implemented in a like manner for specific UDP-based protocols such as the Real-time Transport Protocol (RTP) to achieve seamless mobility for those protocols as well.

One way of thinking of our work is in the context of the end-to-end argument [32], which observes that functionality is often best implemented in a higher layer at an end system, where it can be done according to the application's specific requirements. We show that it is possible to implement mobility as an end-to-end service without network-layer support, while providing multiple mobility modes. In this sense, this is akin to applications deciding between UDP and TCP as a transport protocol; many opt for UDP's simplicity and timeliness over TCP's reliability. In the same fashion, applications should be able to select the mobility mode of their choice.

The other significant advantage of handling mobility on an end-to-end basis is that it enables higher layers like TCP and HTTP to learn about mobility and adapt to it. For example, it is a good idea after a network route change to restart TCP transmissions from slow start or a window-halving [13] since the bottleneck might have changed, or adapt the transmitted content to reflect new network conditions. These optimizations can be made naturally if mobility is handled end-to-end, since no extra signalling is needed. Indeed, the large body of work in mobile-aware applications [15, 22, 25] can benefit from our architecture.

Experience with previous end-to-end enhancements such as various TCP options (e.g., SACK [19]), path MTU discovery, HTTP/1.1, etc., has shown that such techniques often meet with less resistance to widespread deployment than changes to the IP substrate. This supports our belief that, in addition to the flexibility it offers, an end-to-end approach may be successfully deployed.

We have implemented this mobility architecture in Linux 2.2 and have conducted several experiments with it. We are encouraged by the ease with which seamless mobility can be achieved, the flexibility it provides, and the lack of performance degradation. Since our scheme does not impose any triangle routing anomalies, end-to-end latency for active connections is better than standard Mobile IP, and similar to Mobile IP with route optimization.

The rest of this paper describes the technical details of our approach. In Section 2, we survey related work in the area of mobility support. We describe our architecture in Section 3, and detail our new Migrate TCP option in Section 4. We discuss the security ramifications of our approach in Section 5 and our implementation and performance results in Section 6. We address some deployment issues in Section 7 and conclude in Section 8.

2 Related work

The problem of Internet host mobility has been approached from many angles in the literature, but they can be classified into two categories. Some techniques attempt to handle host relocation in a completely transparent fashion, hiding any changes in network structure from the end hosts. We term these techniques *network-layer* mobility. By contrast, many other approaches attempt to handle relocation at a higher level in the end host.

2.1 Network-layer mobility

Mobile IP (RFC 2002) [29] is the current IETF standard for supporting mobility on the Internet. It provides transparent support for host mobility by inserting a level of indirection into the routing architecture. By elevating the mobile host's *home address* from its function as an interface identifier to an *end-point identifier* (EID), Mobile IP ensures the delivery of packets destined to a mobile host's home address, independent of the host's physical point of attachment to the Internet, as reflected in its *care-of address*. Mobile IP does this by creating a routing tunnel between a mobile host's home network and its care-of address.

Such routing tunnels need to be implemented with care because advertising explicit host routes into the wide-area routing tables destroys routing scalability. Mobile IP uses a *home agent* physically attached to the mobile host's home network to intercept and tunnel packets to the mobile host. Hence, packets undergo *triangle* routing, which is often longer than the optimal unicast path.

Further compounding the problem is the widespread deployment of ingress filters [9], ratified in February 2000 by the IETF as a "Best Current Practice" to combat denial-of-service attacks. With this mechanism, a router does not forward packets with a source address foreign to the local network, which implies that a packet sent by a mobile host in a foreign network with its source address set to its home address will not be forwarded. The solution to this is to use *reverse tunneling*, which tunnels packets originating at the mobile host first to the host's home agent (using the host's care-of address as a source address), and then from there on to the destination using the home address as the source address. Thus, routing anomalies occur in both directions.

Perkins and Johnson present a route optimization option for Mobile IP to avoid triangle routing [28]. Here, correspondent hosts cache the care-of address of mobile hosts, allowing communication to proceed directly. It requires an authenticated message exchange from the home agent to the correspondent host [26]. The resulting Mobile IP scheme achieves performance almost equivalent to ours, but requires modifications to the end hosts' IP layer[1] as well as the

[1] In fact, the draft allows on-path routers to cache the care-of addresses instead of the end host, but this requires modifying yet another level of infrastructure.

infrastructure. In contrast, our approach achieves secure, seamless connection migration without a third-party home agent. It also provides a mobile host the ability to pick a mobility mode based on the needs of its applications.

IPv6 provides native support for multiple simultaneous host addresses, and Mobile IPv6 provides mobility support for IPv6 in much the same fashion as Mobile IP for IPv4. IPv6 extensions allow for the specification of a care-of address, which explicitly separates the role of the EID (the host's canonical IP address) and routing location (the care-of address). Gupta and Reddy propose a similar redirection mechanism for IPv4 through the use of ICMP-like control messages which establish care-of bindings at the end hosts [10].

Mysore and Bharghavan propose an interesting approach to network-layer mobility [23], where each mobile host is issued a permanent Class D IP multicast address that can serve as a unique EID. If multicast were widely deployed, this is a promising approach; because a Class D EID has the benefit of being directly routable by the routing infrastructure, it removes the need for an explicit care-of address. However, this scheme requires a robust, scalable, and efficient multicast infrastructure for a large number of sparse groups.

2.2 Higher-layer methods

The home-agent-based approach has also been applied at the transport layer, as in MSOCKS [18], where connection redirection was achieved using a split-connection proxy.

The general idea of using names as a level-of-indirection to handle object and node mobility is part of computer systems folklore. For some years now, people have talked about using the DNS to effect the level-of-indirection needed to support host mobility, but to our knowledge ours is the first specific and complete architecture that uses the DNS to support Internet host mobility. Recently, Adjie-Winoto *et al.* proposed the integration of name resolution and message routing in an Intentional Naming System to implement a "late binding" option that tracks highly mobile services and nodes [1], and it seems possible to improve the performance of that scheme using our connection migration approach.

Our approach differs fundamentally from EID/locator techniques since it requires no additional level of global addressing or indirection, but only a (normally pre-existing) DNS entry and a shared connection key between the two end hosts. Furthermore, unlike previous connection-ID draft proposals such as Huitema's ETCP [11] for TCP connection re-addressing, it requires no modification to the TCP header, packet format, or semantics.[2] Instead, it uses an additional TCP option and the inserts an additional field into the Transmission Control Block (TCB).

There is a large body of work relating to improving TCP performance in wireless and mobile environments [5, 6]. While not the focus of our work, our adherence to standard TCP semantics allows these schemes to continue to work well in our architecture. Furthermore, since end hosts are explicitly notified of mobility, significant performance enhancements can be achieved at the application level [25].

[2]Special RST handling is required on some networks that may rapidly reassign IP addresses; Section 4.5 discusses this issue.

3 An end-to-end architecture

In this section, we describe our end-system mobility architecture. There are three important components in this system: addressing, mobile host location, and connection migration. By giving the mobile host explicit control over its mobility mode, we remove the need for an additional (third-party) home-agent to broker packet routing. The DNS already provides a host location service, and any further control is managed by the communicating peers themselves, triggered by the mobile host when it changes network location.

We assume, like most mobility schemes, that mobile hosts do not change IP addresses more than a few times a minute. We believe this is a reasonable assumption for most common cases of mobility. We emphasize that this does not preclude physical mobility at rapid velocities across a homogeneous link technology, since that can be handled at the physical and link layers, e.g., via link-layer bridging [12].

The rest of this section discusses addressing in a foreign network and host location using the DNS. Section 4 is devoted to a detailed description of TCP connection migration.

3.1 Addressing

The key to the scalability of the Internet architecture is that the IP address serves as a routing locator, reflecting the addressee's point of attachment in the network topology. This enables aggregation based on address prefixes and allows routing to scale well. Our mobility architecture explicitly preserves this crucial property of Internet addressing.

Like Mobile IP, we separate the issues of obtaining an IP address in a foreign domain from locating and seamlessly communicating with mobile hosts. Any suitable mechanism for address allocation may be employed, such as manual assignment, the Dynamic Host Configuration Protocol (DHCP) [7], or an autoconfiguration protocol [34].

While IP addresses fundamentally denote a point of attachment in the Internet topology and say nothing about the identity of the host that may be connected to that attachment point, they have implicitly become associated with other properties as well. For example, they are often used to specify security and access policies as in the case of ingress filtering to alleviate denial-of-service attacks. Our architecture works without violating this trust model and does not require any form of forward or reverse tunneling to maintain seamless connectivity. In a foreign network, a mobile host uses a locally obtained interface address valid in the foreign domain as its source address while communicating with other Internet hosts.

3.2 Locating a mobile host

Once a mobile host obtains an IP address, there are two ways in which it can communicate with correspondent hosts. First, as a client, when it actively opens connections to the correspondent host. In this case, there is no special host location task to be performed in our architecture; using the DNS as before works. However, if the mobile host were to move to another network attachment point during a connection, a new address would be obtained as described in the previous section, and the current connection would continue seamlessly via a secure negotiation with the communicating peer as

described in Section 4. If a mobile host were always a client (not an uncommon case today), then no updates need to be made to any third party such as a home agent or the DNS.

To support mobile servers and other applications where Internet hosts actively originate communication with a mobile host, we use the DNS to provide a level of indirection between a host's current location and an invariant end-point identifier. In Mobile IP, a host's home address is the invariant, and all routing (in the absence of route optimization) occurs via the home agent that intercepts packets destined to this invariant. Ours is not a network-layer solution and we can therefore avoid the indirection for every packet transmission. We take advantage of the fact that a hostname lookup is ubiquitously done by most applications that originate communication with a network host, and use the DNS name as the invariant. We believe that this is a good architectural model: a DNS name identifies a host and does not assume anything about the network attachment point to which it may currently be attached, and the indirection occurs only when the initial lookup is done via a control message (a DNS lookup).

This implies that when the mobile host changes its attachment point, it must detect this and change the hostname-to-address ("A-record") mapping in the DNS. Fortunately, both tasks are easy to accomplish, the former by using a user-level daemon as in Mobile IP, and the latter by using the well-understood and widely available secure DNS update protocol [8, 35]. We note that some DHCP servers today issue a DNS update at client boot time when handing out a new address to a known client based on a static MAC-to-DNS table. This augurs well for the incremental deployability of our architecture, since DNS update support is widely available.

The DNS provides a mechanism by which name resolvers can cache name mappings for some period of time, specified in the time-to-live (TTL) field of the A-record. To avoid a stale mapping from being used from the name cache, we set the time-to-live (TTL) field for the A-record of the name of the mobile host to *zero*, which prevents this from being cached.[3] Contrary to what some might expect, this does not cause a significant scaling problem; name lookups for an uncached A-record do not have to start from a root name server, because in general the "NS-record" (name server record) of the mobile host's DNS name is cacheable for a long period of time (many hours by default). This causes the name lookup to start at the name server of the mobile host's domain, which scales well because of administrative delegation of the namespace and DNS server replication in any domain. We note that some content distribution networks for Web server replication of popular sites use the same approach of small-to-zero TTL values to redirect client requests to appropriate servers (e.g., Akamai [2]). There is no central hot spot because the name server records for a domain are themselves cacheable for relatively long periods of time.

Even with uncacheable DNS entries there still exists a possible race condition where a mobile host moves between when a correspondent host receives the result of its DNS query and when it initiates a TCP connection. Assuming a mobile host updates its DNS entry immediately upon reconnection, the chances of such an occurrence are quite small, but non-zero, especially for a mobile host that makes frequent moves. In this case, the correspondent host will attempt to open a TCP connection to the mobile host's old address, and has no automatic fail-over mechanism.

In this case, the application must perform another DNS lookup to find the new location of the mobile host. We note that the trend towards dynamic DNS records has caused such application-level retries to find their way into applications already—for instance, current FreeBSD `telnet` and `rsh` applications try alternate addresses if an initial connection fails to a host that has multiple DNS A-records. It seems to be only a minor addition to refresh DNS bindings if connection establishment fails.

4 TCP connection migration

A TCP connection [31] is uniquely identified by a 4-tuple: ⟨*source address*, *source port*, *dest address*, *dest port*⟩. Packets addressed to a different address, even if successfully delivered to the TCP stack on the mobile host, must not be demultiplexed to a connection established from a different address. Similarly, packets from a new address are also not associated with connections established from a previous address. This is crucial to the proper operation of servers on well-known ports.

We propose a new *Migrate* TCP option, included in SYN segments, that identifies a SYN packet as part of a previously established connection, rather than a request for a new connection. This Migrate option contains a *token* that identifies a previously established connection on the same destination ⟨*address*, *port*⟩ pair. The token is negotiated during initial connection establishment through the use of a *Migrate-Permitted* option. After a successful token negotiation, TCP connections may be uniquely identified by either their traditional ⟨*source address*, *source port*, *dest address*, *dest port*⟩ 4-tuple, or a new ⟨*source address*, *source port*, *token*⟩ triple on each host.

A mobile host may restart a previously-established TCP connection from a new address by sending a special Migrate SYN packet that contains the token identifying the previous connection. The fixed host will than re-synchronize the connection with the mobile host at the new end point. A migrated connection maintains the same control block and state (with a different end point, of course), including the sequence number space, so any necessary retransmissions can be requested in the standard fashion. This also ensures that SACK and any similar options continue to operate properly. Furthermore, any options negotiated on the initial SYN exchange remain in effect after connection migration, and need not be resent in a Migrate SYN.[4]

Since SYN segments consume a byte in the TCP sequence number space, Migrate SYNs are issued with the same sequence number as the last transmitted byte of data. This results in two bytes of data in a migrated TCP connection with the same sequence number (the new SYN and the previously-transmitted actual data), but this is not a problem since the Migrate SYN segment need never be explicitly acknowledged. Any packet received from the fixed host by a migrating host at the mobile host's new address that has a sequence number in the appropriate window for the current connection implicitly acknowledges the Migrate SYN. Similarly, any further seg-

[3] Modern versions of BIND honor this correctly.

[4] They can be, if needed. For example, it might be useful to renegotiate a new maximum segment size (MSS) reflecting the properties of the new path. We have not yet explored this in detail.

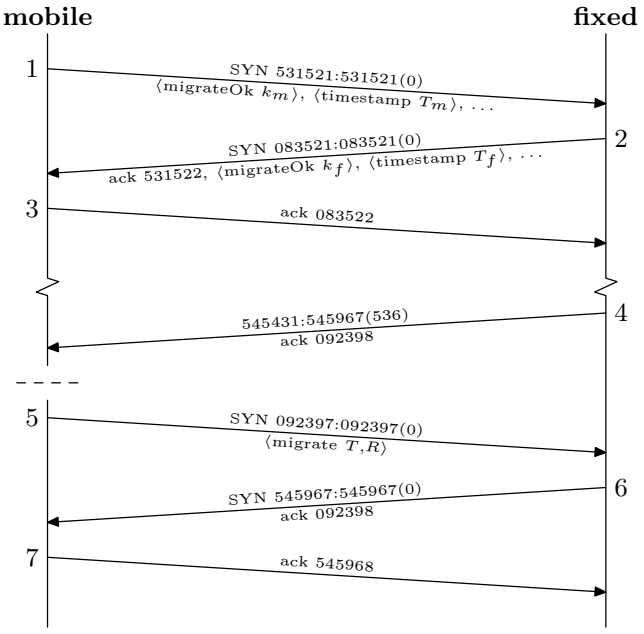

Figure 1: TCP Connection Migration

ments from the mobile host provide the fixed host an implicit acknowledgement of its SYN/ACK. Thus, there is exactly one byte in the sequence space that needs explicit acknowledgement even when the Migrate SYN is used.

4.1 An example

Figure 1 shows a sample connection where a mobile client connects to a fixed host and later moves to a new address. The mobile client initiates the TCP connection in standard fashion in message 1, including a Migrate-Permitted option in the SYN packet. The values k_m and T_m are parameters used in the token negotiation, described in Section 4.3. The fixed server, with a migrate-compliant TCP stack, indicates its acceptance of the Migrate-Permitted option by including the Migrate-Permitted option in its response (message 2). The client completes the three-way handshake with message 3, an ACK. The connection then proceeds as any other TCP connection would, until message 4, the last packet from the fixed host to the mobile host at its current address.

At some time later the mobile host moves to a new address, and notifies the fixed server by sending a SYN packet from its new address in message 5. This SYN includes the Migrate option, which contains the previously computed connection token as part of a migration request. Note that the sequence number of this Migrate SYN segment is the same as the last byte of transmitted data. The server responds in kind in message 6, also using the sequence number of its last transmitted byte of data. The ACK, however, is from the same sequence space as the previous connection. While in this example it acknowledges the same sequence number as the SYN that generated it, it could be the case that segments were lost during a period of disconnect while the mobile host moves, and that the ACK will be a duplicate ACK for the last successfully received in-sequence byte. Since it is addressed to the mobile host's new location, however, it serves as an implicit ACK of the SYN as well. Upon receipt of this SYN/ACK, the mobile host similarly ACKs in the previous sequence space, and the connection resumes as before. All of the options negotiated on the initial SYN except the Migrate-Permitted option are still in effect, and need not be replicated in this or any subsequent migrations.

4.2 Securing the migration

It is possible to partially hijack TCP connections if an attacker can guess the sequence space being used by the connection [21]. With the Migrate options, an attacker who can guess both the sequence space and the connection token can hijack the connection completely. Furthermore, the ability to generate a Migrate SYN from anywhere greatly increases the connection's exposure. While ingress filtering can be used to prevent connection hijacking by attackers not on the path between the end hosts, such methods are ineffective in our case. We must therefore take care to secure the connection token.

The problem is relatively easy to solve if IP security (IPsec) [4] were deployed. While the spectrum of approaches that could be used is outside the scope of this paper, we note that IPsec provides sufficient mechanisms to secure migrateable connections. Currently, however, IPsec has not found wide-spread deployment. Hence, we provide a mechanism to self-secure the Migrate options. End hosts may elect to secretly negotiate an unguessable connection token, which then reduces the security of a migrateable TCP connection to that of a standard TCP connection, since no additional attacks are possible against a migrateable connection without guessing the token, and any attack against a standard TCP connection clearly remains feasible against a migrateable TCP connection.

An unguessable connection token is secured with a secret *connection key*. Since any host that obtains the connection key could fabricate the token and issue a Migrate request, we select the key with an Elliptic Curve Diffie-Hellman key exchange [36], as described below. Hosts using IPsec, or unconcerned with connection security, may choose to disable key negotiation to avoid excess computation.

4.3 Migrate-Permitted option

Hosts wishing to initiate a migrateable TCP connection send a Migrate-Permitted option in the initial SYN segment. Similar to the SACK-Permitted option [19], it should only be sent on SYN segments, and not during an established connection. Additionally, hosts wishing to cryptographically secure the connection token may conduct an Elliptic Curve Diffie-Hellman (ECDH) key exchange through the option negotiation. (Elliptic Curve Diffie-Hellman is preferred to other methods of key establishment due to its high security-to-bit-length ratio. Readers unfamiliar with Elliptic Curve cryptography can find the necessary background material in [3].)

As seen in figure 2, the Migrate-Permitted option comes in two variants—the insecure version, of length 3, and the secure version, with length 20. The secure version is used to negotiate a secret connection key, and contains an 8-bit *Curve Name* and a 136-bit *ECDH Public Key* fragment. The curve name field selects a particular set of domain parameters (the curve, underlying finite field, F, and its representation, the generating point, P, and the order of P, n), as specified in [3]. Use of the insecure version, which contains only a Curve Name field (which must be set to zero) allows the end host

Kind: 15	Length = 3/20	Curve Name	ECDH PK
ECDH Public Key (cont.)			
ECDH Public Key (cont.)			
ECDH Public Key (cont.)			
ECDH Public Key (cont.)			

Figure 2: TCP Migrate-Permitted option

	Kind: 16	Length = 19	ReqNo
Token			
Token (cont.)			
Request			
Request (cont.)			

Figure 3: TCP Migrate option

to skip the key negotiation process. In that case, the connection key is set to all zeros.

The secure variant of the Migrate-Permitted option also requires the use of the Timestamp [14] option in order to store up to 200 bits of ECDH keying material. The EDCH Public Key is encoded using the compressed conversion routine described in [3, Section 4.3.6]. The 136 least-significant bits are stored in the EDCH Public Key field of the Migrate-Permitted option, while the remaining 64 bits of the key are encoded in the Timestamp option. The timestamp option, while often included, is not used on SYN segments. The Protection Against Wrapped Sequence Numbers (PAWS) [14] check is only performed on synchronized connections, which by definition [31] includes only segments after the three-way handshake. Similarly, the Round-Trip Time Measurement (RTTM) [14] procedure only functions when a timestamp has been echoed—clearly this is never the case on an initial SYN segment. Hence the value of the Timestamp option on SYN segments is entirely irrelevant to current TCP stacks. Legacy TCP stacks will never receive a Migrate-Permitted option on a SYN/ACK, hence the Timestamp option will be processed normally. Special handling is only required for the SYN/ACK and following ACK segment on connections that have negotiated the Migrate-Permitted option, as Timestamp fields on these segments will not contain timestamps. Hence the RTTM algorithm must not be invoked for SYN/ACK or initial ACK segments of connections that have negotiated the Migrate-Permitted option.

The Timestamp *TSVal* field contains the 32 most-significant bits of the public key, while the *TSecr* field contains the next 32 most-significant bits. These two components, combined with the 136-bit EDCH Public Key field of the Migrate-Permitted option, constitute the host's public key, k. If the public key is less than 200 bits, it is left-padded with zeros. For any host, i, k_i is generated by selecting a random number, $X_i \in [1, n-1]$, where n is the order of P, and computing

$$k_i = X_i * P$$

The $*$ operation is the scalar multiplication operation over the field F. The security of the connection hinges on the secrecy of the negotiated key, hence X_i should be randomly generated and stored in the control block for each new connection. Any necessary retransmissions of the SYN or SYN/ACK must include identical values for the Migrate-Permitted and Timestamp option.

Upon receipt of an initial SYN with a Migrate-Permitted option, a host, j, with a compliant TCP stack must include a Migrate-Permitted option (and a Timestamp option if the secure variant is used) in its SYN/ACK segment. It similarly selects a random $X_j \in [1, n-1]$ which it uses to construct k_j, its public key, which it sends in the same fashion.

After the initiating host's reception of the SYN/ACK with the Migrate-Permitted and Timestamp options, both hosts can then compute a shared secret key, K, as specified in [36]:

$$K = k_i * X_j = k_j * X_i$$

This secret key is then used to compute a connection validation token. This token, T, is computed by hashing together the key and the initial sequence numbers N_i and N_j using the Secure Hash Algorithm (SHA-1) [24] in the following fashion (recall that host i initiated the connection with an active open, and host j is performing a passive open):

$$T = SHA1(N_i, N_j, K)$$

While SHA-1 produces a 160-bit hash, all but the 64 most-significant bits are discarded, resulting in a cryptographically-secure 64-bit token that is unique to the particular connection. Since SHA-1 is collision-resistant, the chance that another connection on the same $\langle address, port \rangle$ pair has an identical token is extremely unlikely. If a collision is detected, however, the connection must be aborted by sending a RST segment. (The host performing a passive open can check for collisions before issuing a SYN/ACK, and select a new random X_j until a unique token is obtained. Hence the only chance of collision occurs on the host performing the active open.)

4.4 Migrate option

The Migrate option is used to request the migration of a currently open TCP connection to a new address. It is sent in a SYN segment to a host with which a previously-established connection already exists (in the ESTABLISHED or FIN_WAIT states), over which the Migrate-Permitted option has been negotiated.

There are two 64-bit fields in a Migrate option: a *token*, and a *request*. In addition, there is an 8-bit sequence number field, *reqNo*, which must be monotonically increasing with each new migrate request issued by an end host for a connection. (The sequence number allows correspondent hosts to ensure Migrate SYNs were not reordered by the network. Sequence space wrap-around is dealt with in the standard fashion.) The token is simply the 64 most-significant bits of the connection's SHA-1 hash as computed in the Migrate-Permitted option exchange. The request, R, is similarly

the 64 most-significant bits of a SHA-1 hash calculated from the sequence number of the connection initial sequence numbers N, Migrate SYN segment, S, the connection key, K, and the request sequence number, I.

$$R = SHA1(N_i, N_j, K, S, I)$$

SYN segments may now correctly arrive on a bound port not in the LISTEN state. They should be processed only if they contain the Migrate option as specified above. Otherwise, they should be treated as specified in [31]. Upon receipt of a SYN packet with the Migrate option, a TCP stack that supports migration attempts to locate the connection on the receiving port with the corresponding token. The token values for each connection were precomputed at connection establishment, reducing the search to a hash lookup.

If the token is valid, meaning an established connection on this ⟨*address*, *port*⟩ pair has the same token, and the *reqNo* is greater than any previously received migrate request, the fixed host then computes $R = SHA1(N_i, N_j, K, S, I)$ as described above, and compares it with the value of the request in the Migrate SYN. If the comparison fails, or the token was invalid, a RST is sent to the address and port issuing the Migrate SYN, and the SYN ignored. If, on the other hand, the token and request are valid, but the reqNo is smaller than a previously received request, the SYN is assumed to be out-of-order and silently discarded. If the reqNo is identical to the most recently received migrate request this SYN is assumed to be a duplicate of the most recently received SYN, and processed accordingly.

Otherwise, the destination address and port[5] associated with the matching connection should be updated to reflect the source of the Migrate SYN, and a SYN/ACK packet generated, with the ACK field set to the last received contiguous byte of data, and the connection placed in the SYN_RCVD state. Upon receipt of an ACK, the connection continues as before.

4.5 MIGRATE_WAIT state

This section assumes that the reader is familiar with the TCP state machine and transitions [33, Chapter 18].

Special processing of TCP RST messages is required with migrateable connections, as a mobile host's old IP address may be reassigned before it has issued a migrate request to the fixed host. Figure 4 shows the modified TCP state transition diagram for connections that have successfully negotiated the Migrate-Permitted option. The receipt of a RST that passes the standard sequence number checks in the ESTABLISHED state does not immediately terminate the connection, as specified in [31]. Instead, the connection is placed into a new *MIGRATE_WAIT* state. (A similar, but far less likely situation can occur if the fixed host is in the FIN_WAIT1 state—the application on the fixed host has closed the connection, but there remains data in the connection buffer to be transmitted. For simplicity, these additional state transitions are not shown in figure 4.)

Connections in the MIGRATE_WAIT state function as if they were in the ESTABLISHED state, except that they do not emit any segments (data or ACKs), and are moved to CLOSED if they remain

[5]Migrated connections will generally originate from the same port as before. However, if the mobile host is behind a NAT, it is possible the connection has been mapped to a different port.

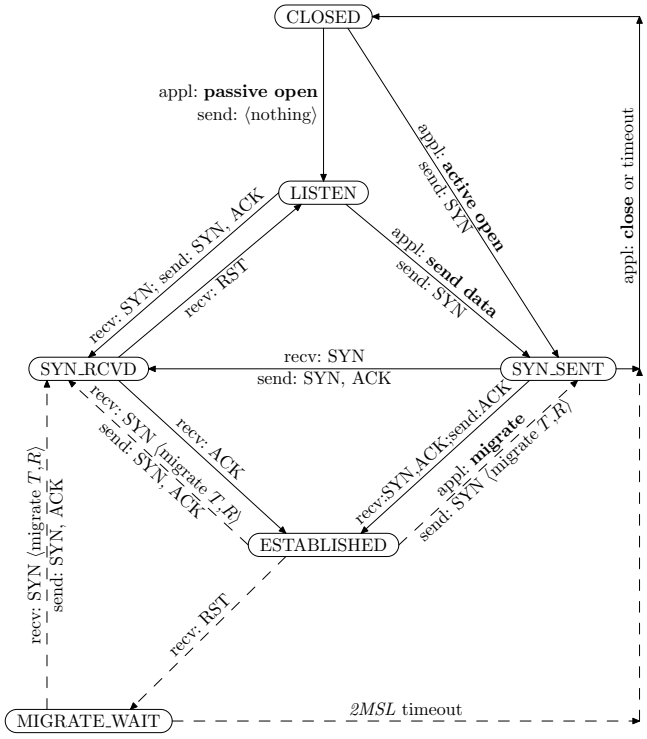

Figure 4: Partial TCP state transition diagram with Migrate transitions (adapted from [33, figure 18.12])

in MIGRATE_WAIT for over a specified period of time. We recommend using the *2MSL* ([31] specifies a Maximum Segment Lifetime (MSL) as 2 minutes, but common implementations also use values of 1 minute or 30 seconds for MSL [33]) period of time specified for the TIME_WAIT state.

Any segments received while in the MIGRATE_WAIT state should be processed as in the ESTABLISHED state, except that no ACKs should be generated. The only way a connection is removed from the MIGRATE_WAIT state is on the receipt of a Migrate SYN with the corresponding connection key. The connection then responds in the same fashion as if it were in the ESTABLISHED state when it received the SYN.

The MIGRATE_WAIT state prevents connections from being inadvertently dropped if the address allocation policy on the mobile host's previous network reassigns the mobile host's old IP address before the mobile host has reconnected at a new location and had a chance to migrate the connection. It also prevents the continued retransmission of data to an unreachable host.

This passive approach to disconnection discovery is preferred over an active, mobile-initiated squelch message because any such message could be lost.[6] Furthermore, a mobile host may not have sufficient (if any) notice of address reassignment to issue such messages. As an added performance enhancement, however, mobile hosts aware of an impending migration may themselves emit a special RST to the peer, which will force the connection into MIGRATE_WAIT, preventing additional packet transmission until the

[6]And any guaranteed-reliable transmission mechanism could take unbounded time.

mobile host has successfully relocated, although such action invokes the strict 2MSL time bound on the allowable delay for host relocation and connection migration.

5 Security issues

An end-to-end approach to mobility simplifies the trust relationships required to securely support end-host mobility compared to network-layer approaches such as Mobile IP. In addition to the relationship between a mobile host and any proxies or home agents, several Mobile IP-based proposals require that a correspondent host in communication with a mobile host assume the responsibility of authenticating communication with an arbitrary set of foreign agents. In their route optimization draft [28], Perkins and Johnson state:

> One of the most difficult aspects of Route Optimization for Mobile IP in the Internet today is that of providing authentication for all messages that affect the routing of datagrams to a mobile node.

Since no third parties are required or even authorized to speak on the mobile host's behalf in an end-to-end architecture, the only trust relationship required for secure relocation is between the mobile and correspondent host. Clearly they already must have a level of trust commensurate with the nature of their communications since they chose to communicate in the first place.

Regardless of the simplicity of trust relationships, there remains the possibility that untrusted parties could launch attacks against the end hosts or connections between them utilizing either dynamic DNS updates or the Migrate and Migrate-Permitted options. The security of dynamic DNS updates is addressed in RFC 2137 [8], resting on the strength of the digital signature scheme used to authenticate mobile hosts.

Possible attacks against the Migrate TCP options include both denial-of-service attacks and methods of migrating connections away from their appropriate end hosts. We discuss these attacks below, and either show why the Migrate options are not vulnerable, or explain why the attack presents no additional threat in relation to standard TCP.

5.1 Denial of service

SYN flooding is a common form of Denial-of-Service (DoS) attack, and most modern TCP implementations have taken great care to avoid consuming unnecessary resources unless a three-way handshake is complete. To validate a Migrate request, the correspondent host performs a significant computation (the SHA-1 hash), which implies we need to be especially vigilant against DoS attacks that attempt to deplete the CPU resources of a target host. The validation is not performed unless an attacker succeeds in guessing a valid, pre-computable token (with a 1 in 2^{64} probability); since a RST message is generated if either the token or the request is invalid, an attacker has no way to identify when it has found a valid token. Because a would-be attacker would therefore have to issue roughly 2^{63} Migrate SYNs to force a request validation, we argue that the TCP Migrate option does not introduce any additional DoS concerns above standard TCP.

5.2 Connection hijacking

Since a Migrate request contains a hash of both the SYN segment's sequence number and migrate request sequence number, a replayed Migrate option can only be used until either a new byte of data or another migrate connection is sent on the connection. Since self-migration is not allowed, duplicate Migrate SYNs (received outside of the three-way handshake) are ignored by the peer TCP. If, however, the mobile host moves rapidly to a another new location, a replayed Migrate SYN could be used to migrate the connection back to the mobile host's previous IP, which may have been subsequently assumed by the attacker. In order to prevent this attack, the Migrate Request option processing ignores the source address and port in duplicate packets, as a valid request from a relocated mobile host would include a higher request number.

More worrisome, however, is the fact that once a Migrate SYN has been transmitted, the token is known by any hosts on the new path, and denial-of-service attacks could be launched by sending bogus Migrate SYNs with valid tokens. If a mobile host includes a new Migrate-Permitted option in its Migrate SYN, however, the window of opportunity when the previous connection token can be used (if it was snooped) is quite small—only until the new three-way handshake is successfully completed.

5.3 Key security

The connection key used by the Migrate option is negotiated via Elliptic Curve Diffie-Hellman to make it extremely difficult even for hosts that can eavesdrop on the connection in both directions to guess the key. Without sufficient information to verify possible keys off-line, an attacker would have to continually generate Migrate SYNs and transmit them to one of the end hosts, hoping to receive a SYN/ACK in response to a correct guess. Clearly such an attack is of little concern in practice, as the expected 2^{63} SYN packets required to successfully guess the key would generate sufficient load as to be a DoS problem in and of themselves.

Hosts that lie on the path between end hosts, however, have sufficient information (namely the two Elliptic Curve Diffie-Hellman components) to launch an attack against the Elliptic Curve system itself. The best known attack is a distributed version of Pollard's rho-algorithm [30], which [17] uses to show that a 193-bit EC system would require $8.52 \cdot 10^{14}$ MIPS years, or about $1.89 \cdot 10^{12}$ years on a 450Mhz Pentium II, to defeat.

While this seems more than secure against ordinary attackers, an extremely well-financed attacker might be able to launch such an attack on a long-running connection in the not-too-distant future. The obvious response is to increase the key space. Unfortunately, we are restricted by the 40-byte limitation on TCP options. Given the prevalence of the MSS (4 bytes), Window Scale (3 bytes), SACK Permitted (2 bytes), and Timestamp (10 bytes) options (of which we are already using 8 bytes) in today's SYN segments, the 20-byte Migrate-Permitted option is already as large as is feasible. We argue that further securing the connection key against brute-force attacks from hosts on the path between the two end hosts is largely irrelevant, given the ability of such hosts to launch man-in-the-middle attacks against TCP with much less difficulty!

The security of TCP connections, migrateable or not, continues to remain with the authentication of end hosts, and the establishment

of strong session keys to authenticate ongoing communication. Although we have taken care to ensure the Migrate option does not further decrease the security of TCP connections, the latter are inherently insecure, as IP address spoofing and sequence number guessing are not very difficult. Hence we strongly caution users concerned with connection security to use additional application-layer cryptographic techniques to authenticate end points and the payload traffic.

5.4 IPsec

When used in conjunction with IPsec [4], there are additional issues raised by the use of the Migrate options. IPsec Security Associations (SAs) are established on an IP-address basis. When a connection with an associated SA is migrated, a new SA must be established with the new destination address before communication is resumed. If the establishment of a this new SA conflicts with existing policy, the connection is dropped. This seemingly unfortunate result is actually appropriate. Since IPsec's Security Policy Database (SPD) is keyed on IP network address, the policies specified within speak to a belief about the trustworthiness of a particular portion of the network.

If a mobile host attaches to a foreign network, any security assumptions based on its normal point of attachment are invalid. If the end host itself continues to have sufficient credentials independent of its point of attachment, an end-to-end authentication method should be used, and a secure tunnel established for communication over the untrusted network. A discussion of such techniques is outside of the scope of this document.

6 Implementation

We have implemented this architecture in the Linux 2.2.15 kernel, using Bind 8.2.2-P3 as the name server for mobile hosts. The IPv4 TCP stack has been modified to support the Migrate options. Connection migration can be affected through two methods. Applications with open connections may explicitly request a migration by issuing an `ioctl()` on the connection's file descriptor specifying the address to migrate to. Most current applications, however, lack a notification method so the system can inform them the host has moved. Hence we also provide a mechanism for processes to migrate open connections, regardless of whether they have the file descriptor open or not.

This is done through the Linux `/proc` file system. A directory `/proc/net/migrate` contains files of the form *source address*`:`*source port*`->`*dest address*`:`*dest port* for each open connection that has successfully negotiated the Migrate-Permitted option. These files are owned by the user associated with the process that opened the connection. Any process with appropriate permissions can then write a new IP address to these files, causing the corresponding connection to be migrated to the specified address. This method has the added benefit of being readily accessed by a user directly through the command line.

It is expected that mobile hosts will run a mobility daemon that tracks current points of network attachment, and migrates open connections based on some policy about the user's preference for certain methods of attachment. For instance, when an 802.11 interface comes up on a laptop that previously established connections on a CDPD link, it seems likely that the user would opt to migrate most open connections to the address associated with the 802.11 link. Similarly the daemon could watch for address changes on attached interfaces (possibly as a result of DHCP lease expirations and renewals) and migrate connections appropriately. We plan to implement such a daemon in the near future.

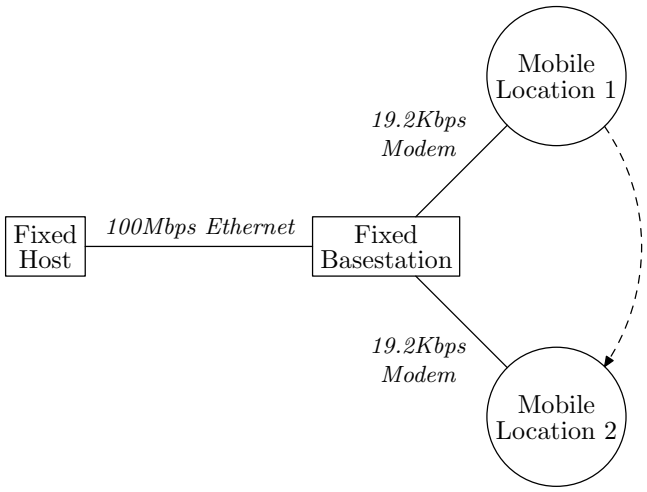

Figure 5: Network topology used for migration experiments

6.1 Experiments

Figure 5 shows the network topology used to gather the TCP traces shown in figures 6 and 7. The traces were collected at the fixed basestation, which is on the path between the fixed host and both mobile host locations. We conducted TCP bulk transfers from a server on the fixed host to a client on the mobile host. The client initiates the connection from one location, and migrates to another location at some later point. Both mobile host locations use identical connections, a 19.2Kbps serial link with ≈100ms round-trip latency. The basestation and fixed host are on a 100Mbps Ethernet segment, hence the link to the mobile host is the connection bottleneck. This topology is intentionally simple in order to isolate the various subtleties of migrating TCP connections, as discussed below.

Figure 6 shows the TCP sequence trace of a migrated TCP connection. At time $t \approx 4.9s$ the mobile host moved to a new address and issued a Migrate SYN, as depicted by the dotted line. Since the host is no longer attached at its previous address, all of the enqueued segments at the bottleneck are lost. (The amount of lost data is bounded by the advertised receive window of the mobile host. A host that moves frequently across low-bandwidth connections may wish to advertise a smaller receive window to reduce the number of wasted segments.) Finally, at $t \approx 6.8s$ the fixed host's SYN/ACK passes through the bottleneck, and is ACKed by the fixed host a RTT later.

The fixed host does not immediately restart data transmissions because the TCP Migrate options do not change the congestion-avoidance or retransmission behavior of TCP. The sender is still waiting for ACKs for the lost segments; as far as it is concerned, it has only received two (identical) ACKs—the original ACK, and one duplicate as part of the Migrate SYN three-way handshake.

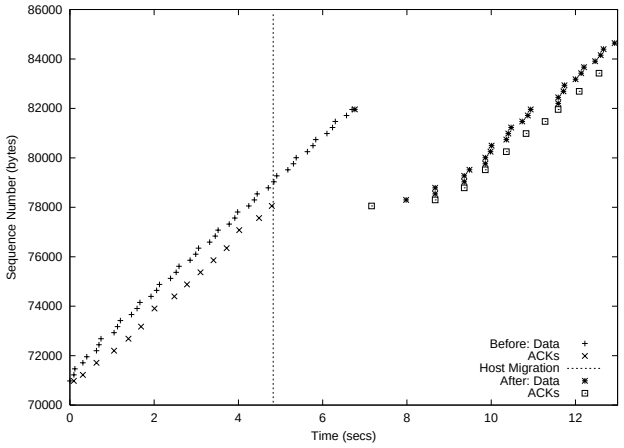

Figure 6: A TCP connection sequence trace showing the migration of an open connection

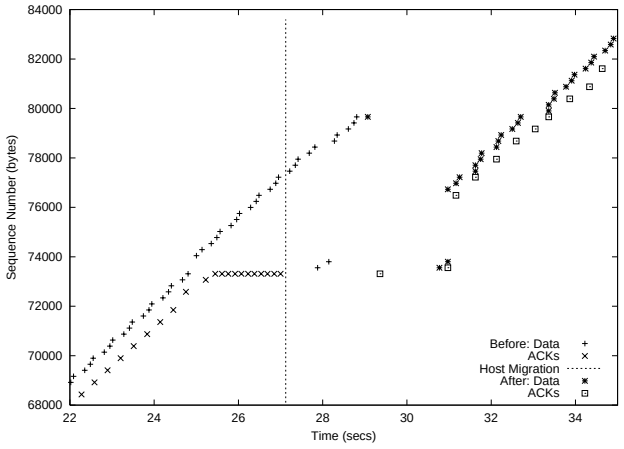

Figure 7: A TCP Migrate connection (with SACK) sequence trace with losses just before migration

Finally, at $t \approx 7.8s$ the retransmission timer expires (the interval is from the first ACK, sent earlier at $t \approx 4.9s$) and the fixed host retransmits the first of the lost segments. It is immediately acknowledged by the mobile host, and TCP resumes transmission in slow-start after the timeout.

Figure 7 shows the TCP sequence trace of a similar migrate TCP connection. As before, the dashed line indicates the mobile host issued a migrate request at time $t \approx 27.1s$. This time, however, there were additional losses on the connection that occurred just before the migration, as can be seen at $t \approx 24.9s$. These segments are fast-retransmitted, and pass through the bottleneck at $t \approx 28s$ due to the DUP-ACKs generated by the remaining SYNs. Unfortunately, this is after the mobile host has migrated, so they, along with all the segments addressed to the mobile host's initial address after $t \approx 27.1s$, are lost.

At $t \approx 29s$, the Migrate SYN/ACK makes it out of the queue at the bottleneck, and the mobile host immediately generates an ACK. As in the previous example, however, the fixed host is still awaiting ACKs for previously transmitted segments. It is only at $t \approx 31s$ that the timer expires and the missing segments are retransmitted. Notice that because SACK prevents the retransmission of the previously-received segments, only those segments lost due to the mobile host's address change are retransmitted, and the connection continues as before. The success of this trace demonstrates that the Migrate options work well with SACK due to the consistency of the sequence space across migrations.

6.2 Performance enhancements

Several enhancements can be made by implementations to improve overall connection throughput during connection migration. The most obvious of these is issuing three DUP-ACKs immediately after a migrate request, thereby triggering the fast-retransmit algorithm and avoiding the timeout seen in the previous example [6]. By preempting the timeout, the connection further avoids dropping into slow-start and congestion avoidance.

Such techniques should be used with care, however, as they assume the available bandwidth of the new path between mobile and fixed host is on the same order-of-magnitude as the previous path. For migrations across homogeneous technologies this may be a reasonable assumption. When moving from local to wide-area technologies, however, there may be order-of-magnitude discrepancies in the available bandwidth. Hence we do not include such speed-ups in the TCP Migrate specification, and leave it to particular implementations to responsibly evaluate the circumstances and provide behavior compatible with standard TCP.

7 Deployment Issues

As with any scheme for mobility support, there are some deployment issues to be addressed. By pushing the implementation of mobility mechanisms—connection migration in particular—to the end points, our system requires changes to each transport protocol. Fortunately, our TCP connection migration protocol can be generalized to other UDP-based protocols with little difficulty. Significant examples include streaming protocols such as RTP and proprietary protocols like Real, Quicktime and Netshow. We note that most of these already have a control channel used for congestion and quality control, and such applications would likely wish to be informed of changes due to mobility as well. Furthermore, we argue that not all applications *require* network-layer mobility, especially those characterized by short transactions where an application-level retry of the transaction is easy to perform; we therefore make the case using the end-to-end argument that mobility might be best implemented as a higher-level, end-to-end function just like reliability.

Perhaps the biggest limitation of our approach is that both peers cannot move *simultaneously*.[7] Because our scheme has no anchor point like Mobile IP's home agent, any IP address change must be completed before the other can proceed. We do not view this as a serious limitation to the widespread applicability of the protocol, since we are primarily targeting infrastructure-based rather than ad-hoc network topologies in this work.

In addition to these two limitations, there are several issues that crop up when one considers presently-deployed applications. While it is currently possible for Internet hosts to be re-addressed while

[7] "Simultaneously" is defined as whenever the intervals between address change and the (would-be) reception of the Migrate SYN by the corresponding host for both end hosts overlap.

operating (due to a DHCP lease expiration or similar event), it is quite rare. Hence some applications have made assumptions about the stability of network addresses, which are no longer valid in our architecture. We discuss some of these issues below.

7.1 Address caching

There is a class of applications that store IP addresses within the application, and communicate these addresses to a remote host. Such applications would not function properly under our architecture. They are readily identifiable, however, as another currently widely-deployed technology also breaks such applications: Network Address Translators (NATs). While the wisdom of Network Address Translation is a hotly debated topic, there is little chance it will disappear any time soon. Hence most applications designed today take care not to transmit addresses as part of the application-layer communication, and therefore will also work in our architecture. In fact, one can make the case that such applications are broken, since IP addresses are only identifiers of attachment points, not hosts.

Another, larger class of applications cache the results of *gethostbyname()*, and may not perform further hostname resolution.[8] Furthermore, DNS resolvers themselves cache hostname bindings as discussed in Section 3. Unfortunately many older name servers enforce a local TTL minimum, often set to five minutes. Since newer versions of popular name servers adhere to the TTL specified in the returned resource record, this problem should disappear as upgrades are made.

7.2 Proxies and NATs

Proxies actually help the deployment of our scheme, as we only need to modify the proxy itself, and all communications through the proxy will support mobility. Similarly, NATs can also provide transparent support without remote system modification. In fact, a NAT doesn't even need a modified TCP stack. It need only snoop on TCP SYNs (which it does anyway), note the presence of a Migrate-Permitted option, and snoop for the SYN/ACK (which it does anyway). If the SYN/ACK does not contain a Migrate-Permitted option, the NAT can support connection migration internal to its network by inserting a corresponding Migrate-Permitted option, and continuing to snoop the flow looking for any Migrate SYNs. It need only fabricate a corresponding SYN/ACK and update its address-to-port mappings, without passing anything to the end host. Further, by avoiding any explicit addressing in migrate requests, the Migrate options function properly though legacy NATs, and even allow a mobile host to move between NATs, as connections may change not only address but port as well.

7.3 Non-transactional UDP applications

Many UDP applications are transactional in nature. UDP is, by definition, a datagram protocol, and an inopportune change of IP address is only one of many reasons for an unsuccessful UDP transaction. The transaction will need to be retried, although a new hostname binding should be obtained first.

There exists at least one glaring exception to this rule. The Network File System protocol (NFS) represents one of the most prevalent UDP applications in use today and uses IP addresses in its mount points.[9] We believe, given the characteristics of network links likely to be encountered by mobile hosts, it is likely that NFS-over-TCP is a better choice than UDP. Otherwise, a mobile host would need to dismount and re-mount NFS filesystems upon reconnection.

8 Conclusion

This paper presents an end-to-end architecture for Internet host mobility that makes no changes to the underlying IP communication substrate. It uses secure updates to the DNS upon an address change to allow Internet hosts to locate a mobile host, and a set of connection migration options to securely and efficiently negotiate a change in the IP address of a peer without breaking the end-to-end connection. We have implemented this architecture in the Linux operating system and are encouraged by the ease with which mobility can be achieved without any router support, the flexibility to mobile hosts provided by it, and performance comparable to Mobile IP with route optimization.

Our architecture allows end systems to choose a mobility mode best suited to their needs. Routing paths are efficient with no triangle routing, and any connection involving the mobile host shares fate only with the communicating peer and not with any other entity like a home agent. When a mobile host is in a foreign network and communicating with another host, the disruption in connectivity caused by a sudden IP address change is proportional to the round-trip time of the connection. When a mobile host accepts no passive connections, the protocol does not require even the DNS update notification, and seamless connectivity across host mobility is achieved using completely end-to-end machinery.

The security of our approach is based on a combination of the well-documented secure DNS update protocol in conjunction with a new secure connection migration mechanism. Our architecture and implementation function across a variety of other components of the Internet architecture, including firewalls, NATs, proxies, IPsec, and IPv6. We believe that our architecture scales well even when most Internet hosts become mobile because lookups and updates are distributed across administratively-delegated, replicated DNS servers.

We note that our connection migration scheme, the MIGRATE_WAIT state in particular, avoids address assignment race conditions, but does *not* support host disconnectivity. Hence, as with Mobile IP and other mobility schemes, TCP connections may be lost if the mobile host's relocation is accompanied by a prolonged period of disconnectivity. We are hopeful our end-to-end approach may be extended to support general host disconnectivity as well.

Acknowledgements

We thank John Ankcorn, Frans Kaashoek, Eddie Kohler, Robert Morris, Srinivasan Seshan, Tim Sheppard, and Karen Sollins for helpful comments on earlier drafts of this paper. We are indebted to David Andersen, who helped improve the security of our initial Migrate scheme, and David Mazieres, who suggested we use Elliptic Curve Diffie-Hellman key exchange for additional key strength.

[8] Some popular Web browsers display this behavior.

[9] We note that most other advanced file systems, such as Coda [22] and newer versions of NFS use TCP, which gives good congestion control and reliability behavior.

References

[1] ADJIE-WINOTO, W., SCHWARTZ, E., BALAKRISHNAN, H., AND LILLEY, J. The design and implementation of an intentional naming system. In *Proc. ACM SOSP '99* (Dec. 1999), pp. 186–201.

[2] AKAMAI TECHNOLOGIES, INC. `http://www.akamai.com`.

[3] AMERICAN NATIONAL STANDARDS INSTITUTE. Public key cryptography for the financial service industry: The elliptic curve digital signature algorithm. ANSI X9.62 - 1998, Jan. 1999.

[4] ATKINSON, R. Security architecture for the internet protocol. RFC 1825, IETF, Aug. 1995.

[5] BALAKRISHNAN, H., SESHAN, S., AND KATZ, R. H. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks 1*, 4 (Dec. 1995), 469–481.

[6] CACERES, R., AND IFTODE, L. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE JSAC 13*, 5 (June 1995).

[7] DROMS, R. Dynamic Host Configuration Protocol. RFC 2131, IETF, Mar. 1997.

[8] EASTLAKE, 3RD, D. E. Secure domain name system dynamic update. RFC 2137, IETF, Apr. 1997.

[9] FERGUSON, P., AND SENIE, D. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2267, IETF, Jan. 1998.

[10] GUPTA, S., AND REDDY, A. L. N. A client oriented, IP level redirection mechanism. In *Proc. IEEE Infocom '99* (Mar. 1999).

[11] HUITEMA, C. Multi-homed TCP. Internet Draft, IETF, May 1995. (expired).

[12] IEEE. Wireless medium access control (MAC) and physical layer (PHY) specifications. Standard 802.11, 1999.

[13] JACOBSON, V. Congestion avoidance and control. In *Proc. ACM SIGCOMM '88* (Aug. 1988), pp. 314–329.

[14] JACOBSON, V., BRADEN, R., AND BORMAN, D. TCP extensions for high performance. RFC 1323, IETF, May 1992.

[15] JOSEPH, A. D., TAUBER, J. A., AND KAASHOEK, M. F. Mobile computing with the rover toolkit. *IEEE Trans. on Computers 46*, 3 (Mar. 1997), 337–352.

[16] KARN, P. Qualcomm white paper on mobility and IP addressing. `http://people.qualcomm.com/karn/papers/mobility.html`, Feb. 1997.

[17] LENSTRA, A. K., AND VERHEUL, E. R. Selecting cryptographic key sizes. `http://www.cryptosavvy.com`, Nov. 1999.

[18] MALTZ, D., AND BHAGWAT, P. MSOCKS: An architecture for transport layer mobility. In *Proc. IEEE Infocom '98* (Mar. 1998).

[19] MATHIS, M., MAHDAVI, J., FLOYD, S., AND ROMANOW, A. TCP selective acknowledgment options. RFC 2018, IETF, Oct. 1996.

[20] MOCKAPETRIS, P. V., AND DUNLAP, K. Development of the domain name system. In *Proc. ACM SIGCOMM '88* (Aug. 1988), pp. 123–133.

[21] MORRIS, R. T. A weakness in the 4.2BSD UNIX TCP/IP software. Computing science technical report 117, AT&T Bell Laboratories, Murray Hill, New Jersey, Feb. 1985.

[22] MUMMERT, L. B., EBLING, M. R., AND SATYANARAYANAN, M. Exploiting weak connectivity for mobile file access. In *Proc. ACM SOSP '95* (Dec. 1995), pp. 143–155.

[23] MYSORE, J., AND BHARGHAVAN, V. A new multicasting-based architecture for internet host mobility. In *Proc. ACM/IEEE Mobicom '97* (Sept. 1997), pp. 161–172.

[24] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. The Secure Hash Algorithm (SHA-1). NIST FIPS PUB 180-1, U.S. Department of Commerce, Apr. 1995.

[25] NOBLE, B. D., SATYANARAYANAN, M., NARAYANAN, D., TILTON, J. E., FLINN, J., AND WALKER, K. R. Agile application-aware adaptation for mobility. In *Proc. ACM SOSP '97* (Oct. 1997), pp. 276–287.

[26] PERKINS, C. E., AND CALHOUN, P. R. Mobile IP challenge/response extensions. Internet Draft, IETF, Feb. 2000. `draft-ietf-mobileip-challenge-09.txt` (work in progress).

[27] PERKINS, C. E., AND JOHNSON, D. B. Mobility support in IPv6. In *Proc. ACM/IEEE Mobicom '96* (Nov. 1996), pp. 27–37.

[28] PERKINS, C. E., AND JOHNSON, D. B. Route optimization in mobile IP. Internet Draft, IETF, Feb. 2000. `draft-ietf-mobileip-optim-09.txt` (work in progress).

[29] PERKINS, ED., C. E. IP mobility support. RFC 2002, IETF, Oct. 1996.

[30] POLLARD, J. Monte carlo methods for index computation mod p. *Mathematics of Computation 32* (1978), 918–924.

[31] POSTEL, ED., J. Transmission Control Protocol. RFC 793, IETF, Sept. 1981.

[32] SALTZER, J. H., REED, D. P., AND CLARK, D. D. End-to-end arguments in system design. *ACM TOCS 2*, 4 (Nov. 1984), 277–288.

[33] STEVENS, W. R. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, Reading, Massachusetts, 1994.

[34] THOMSON, S., AND NARTEN, T. IPv6 stateless address autoconfiguration. RFC 2462, IETF, Dec. 1998.

[35] VIXIE, P., THOMSON, S., REKHTER, Y., AND BOUND, J. Dynamic updates in the domain name system (DNS UPDATE). RFC 2136, IETF, Apr. 1997.

[36] ZUCCHERATO, R., AND ADAMS, C. Using elliptic curve Diffie-Hellman in the SPKM GSS-API. Internet Draft, IETF, Aug. 1999. `draft-ietf-cat-ecdh-spkm-00.txt` (work in progress).

Mesh-Based Content Routing using XML

Alex C. Snoeren, Kenneth Conley, and David K. Gifford
MIT Laboratory for Computer Science
Cambridge, MA 02139

{snoeren, conley, gifford}@lcs.mit.edu

Abstract

We have developed a new approach for reliably multicasting time-critical data to heterogeneous clients over mesh-based overlay networks. To facilitate intelligent content pruning, data streams are comprised of a sequence of XML packets and forwarded by application-level XML routers. XML routers perform content-based routing of individual XML packets to other routers or clients based upon queries that describe the information needs of downstream nodes. Our PC-based XML router prototype can route an 18 Mbit per second XML stream.

Our routers use a novel Diversity Control Protocol (DCP) for router-to-router and router-to-client communication. DCP reassembles a received stream of packets from one or more senders using the first copy of a packet to arrive from any sender. When each node is connected to n parents, the resulting network is resilient to $(n-1)$ router or independent link failures without repair. Associated mesh algorithms permit the system to recover to $(n-1)$ resilience after node and/or link failure. We have deployed a distributed network of XML routers that streams real-time air traffic control data. Experimental results show multiple senders improve reliability and latency when compared to tree-based networks.

1 Introduction

Our research is motivated by an interest in highly reliable data distribution technologies that can deliver information to end clients with low latency in the presence of both node and link failures. Low latency can be crucial for certain data that are extremely time critical. For example, real-time trading systems rely upon the timely arrival of current security prices, air-traffic control systems require up-to-the-second data on aircraft position and status, and gaps or delay in live network video and audio feeds can be distracting. In such environments, even a sub-second pause in a data feed while a delivery network retransmits or reconfigures may be unacceptable. Recent studies have shown the Internet recovers from failures on a much slower scale, often on the order of minutes [2, 20].

We observe that the achievable latency of a reliable data stream is bounded by the packet loss-recovery mechanism. Packet losses can be handled by retransmission or redundant coding. Retransmission methods limit recovery time to the round-trip delay between communicating nodes. In order to avoid retransmission in the face of loss redundant data must be sent.

This work is based upon the assumption that, in certain cases, the value of reliable and timely data delivery may justify increased transport costs if the cost increase allows us to meet a desired reliability goal. Systems often try to avoid the delay penalty by using loss-resistant coding schemes which encode redundant information into the data stream. We extend this redundancy to network delivery paths and senders. Recent work in overlay networks has shown that multiple, distinct paths often exist between hosts on the Internet [2]. We attempt to leverage these redundant network links. While some may consider this additional bandwidth wasteful, we believe the system described herein presents an interesting and elegant method of utilizing additional network resources to achieve levels of reliability and latency previously difficult to obtain.

Our basic approach is to construct a content distribution *mesh*, where every node is connected to n parents, receiving duplicate packet streams from each of its parents. The value of n is a configuration parameter that is used to select the desired trade-off between latency, reliability, and transport costs. By maintaining an acyclic mesh, this approach guarantees that the minimum cut of the mesh is n nodes or independent links. Thus, a mesh is resilient to $(n-1)$ node or $(n-1)$ independent link failures (we say $(n-1)$ resilient) without repair. If a mesh failure occurs, recovery algorithms restore the mesh to $(n-1)$ resilience in a few seconds.

Our architecture is based upon an overlay network that transports XML streams. An *XML packet* is a single independent XML document [7]. An *XML stream* is a sequence of XML packets, and each XML packet in a stream can have a different document type definition (DTD). When clients join an overlay network they specify an XML query that describes the XML packets they would like to receive. It is the job of the overlay network to configure itself to deliver the desired XML stream to a client at reasonable cost given reliability goals. Queries are expressed in a general language such as XQuery [11].

Our overlay network is implemented by XML routers. An *XML router* is a node that receives XML packets on one or more input

This research was supported in part by DARPA (Grant No. F30602-97-1-0283). Alex C. Snoeren was supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

links and forwards a subset of the XML packets it receives to each output link. Each output link has a query that describes the portion of the router's XML stream that should be sent to the host on that link. XML routers are components in a distributed publish-subscribe network and implement the selective forwarding of XML packets according to subscriptions described by queries.

XML has a number of advantages over a byte stream for multicast delivery. First, XML permits the network to interpret client data needs in terms of well-defined XML queries. Second, XML packets suggest what logical units of data will be processed together by a client and thus can aid network scheduling. Third, many tools and standards exist for XML making it easy for both the data originator and receiver to build robust applications. Finally, our approach allows applications and databases to push part of their processing into the network fabric. We expect that query languages such as XQuery will become standardized, allowing a single language to be used to describe data requirements. This standardization will permit applications to program our network fabric to deliver the data they need in a simple, consistent fashion.

The primary disadvantage of XML is often thought to be the increased number of bytes required to represent the same information in XML when compared to an application specific encoding. However, our experimental results suggest that conventional data compression eliminates this disadvantage. While an XML stream must be decompressed and recompressed at any router that wishes to do query matching, a router that passes all packets to every client can bypass the XML switch component entirely, and no decompression or compression need be performed. Thus, routers can include a fast-path for clients that subscribe to the unfiltered XML stream.

This paper makes three distinct, novel contributions:

- *XML Routing.* To the best of our knowledge, we describe the first packet-based network XML router to support arbitrary content routing. We believe that systems for XML routing will be useful in a wide variety of contexts and will be efficient because XML wrapper overhead can be removed by appropriate use of data compression technology.
- *Mesh-based overlay networks.* We describe the first overlay network to use multiple, redundant paths for simultaneous transport of multicast streams. Our mesh approach offers better latency performance than tree-based approaches.
- *Diversity Control Protocol.* We describe a novel protocol that uses source-independent sequence numbers to reliably reconstruct a sequenced packet stream from multiple sources. DCP reduces latency and improves reliability when compared with conventional single-sender approaches.

The remainder of this paper describes our current XML routing infrastructure in the following sections:

- Previous work (Section 2)
- Architecture of our XML routing system (Section 3)
- Mesh algorithms and distribution protocol (Section 4)
- Experimental results and our air traffic control application (Section 5)
- Issues involved in routing XML over a mesh (Section 6)
- Conclusions (Section 7)

2 Previous work

Our work on XML routers and DCP builds on a large body of past work in reliable multicast and overlay networks. We consider related work in four areas: reliable multicast, overlay networks, redundant coding and transmission schemes, and publish-subscribe networks.

2.1 Reliable multicast

Reliable multicast systems send a stream of packets to a set of receivers. Reliable multicast systems are often built on IP Multicast [3]. IP Multicast packets are duplicated by the network layer as late as possible to minimize the network resources consumed to deliver a single packet to multiple receivers. Acknowledgments are required to make IP Multicast reliable. If a packet is damaged in transmit or is lost, either a receiver will send a negative acknowledgment to the sender [14, 22, 27, 41, 43], or the lack of a positive acknowledgment from a receiver will cause the sender to retransmit [17, 22, 43]. Express [15] is a single-source multicast system that simplifies IP Multicast in the face of multiple data sources but is still integrated with the network fabric.

Of particular note is RMX [12], which shares similar goals with our work. RMX provides real-time reliable multicast to heterogeneous clients through the use of application-specific transcoding gateways. For example, it supports re-encoding images using lossy compression to service under-provisioned clients. By using self-describing XML tags, our architecture allows similar functionality to be provided in a general fashion by having clients with different resource constraints subscribe to different (likely non-disjoint) portions of the data stream.

2.2 Overlay networks

An *overlay network* is a virtual network fabric that is implemented by application level routers that communicate with each other and end clients using normal IP network facilities. Overlay networks typically use reliable point-to-point byte streams, such as TCP, to implement reliable multicast. The goal of an overlay network is typically to provide increased robustness [2, 35] or additional, sophisticated network services, such as wide-area stream broadcast [16, 30, 37], without underlying network assistance. In fact, network operators may be unaware that such services are running on their network.

One advantage of building our network as an overlay is that it is easy to modify and deploy without the cooperation of network providers. We have adopted the use of overlay networks as an effective way to build a robust mesh that can effectively route XML packets. End-system-multicast [13] is an overlay-based multicast system that constructs meshes during spanning tree discovery but does not use redundant mesh links for information delivery.

2.3 Redundant encoding and transmission

Loss-tolerant encoding schemes (often termed erasure, tornado, or forward error correcting (FEC) codes) use redundant information to support the reconstruction of a data stream in the face of a certain amount of packet loss [25]. For example, in Digital Fountain's Meta-Content protocol [9] packets are encoded to allow a receiver

to recover a data stream even if a certain fraction of Meta-Content packets are never received.

Our approach to redundancy is based on sender and channel diversity while loss-tolerant encoding schemes typically use only packet diversity [9, 33]. We use channel diversity because experimental data suggests that Internet packet errors are highly path dependent [2, 29, 35]. We use sender diversity because in single-sender systems a sender failure is likely to cause a stream gap during recovery [30]. Based on these assumptions, we believe that, with appropriately configured levels of mesh redundancy, sender and channel diversity can provide lower loss rates and latency than packet diversity, albeit at a higher cost.

Several previous systems have leveraged channel diversity, sender diversity, or both in an end-to-end fashion. Dispersity routing [24] and IDA [31] split the transfer of information over multiple network paths to provide enhanced reliability and performance. Simulation results and analytic studies have shown the benefits of this approach [5, 6]. In addition, tornado codes have been suggested to combine parallel downloads to improve reliability and performance [8]. Application-level dispersity routing, IDA, and parallel downloads use multiple network paths but do not provide for any loss recovery along a single path within the network fabric. Our use of application-level routers allows us to perform loss recovery inside of the network fabric and, thus, improve loss resilience. Further, the block encoding scheme used by Digital Fountain may add additional latency during decoding. We discuss our loss resilience results in Section 5.

2.4 Publish-subscribe systems

Publish-subscribe networks, such as Tibco's TIB^{TM}Rendezvous [28], Elvin4 [36], Siena [10], Gryphon [4], and XMLBlaster [1] permit receivers to specify the portion of a data stream that they would like to receive. Receivers typically subscribe to messages using a query that summarizes their interests. Streams may be encoded such that the same content, but in varying levels of fidelity, may be requested by each client [26, 42]. Siena and Gryphon both provide distributed implementations of singly connected graphs for information distribution, but neither provides XML-based routing.

XMLBlaster [1] is a publish-subscribe system based on XML packet streams, but it only permits queries over a specific header field. Our semantics permit queries over any field in an XML packet. We believe that the overhead of making each XML packet a fully formatted document is a small price to pay for the resulting flexibility and rational query semantics. This is especially the case when data compression causes the markup overhead in each XML packet to become negligible. To our knowledge, no existing stream-based publish-subscribe network uses redundant meshes for reliability or performance enhancement.

3 Resilient mesh networks

As shown in Figure 1, a typical overlay network for routing an XML stream contains one or more root routers (R1-R2), a variable number of internal routers (I1-I3), and a variable number of edge clients (C1-C3). Root routers are the origin of data and are assumed to have independent means of generating their XML stream. Internal routers receive the XML stream from their parent routers and forward elements of the stream to their children as required. Clients connect to routers and provide a query that describes the portions of the XML stream they would like to receive.

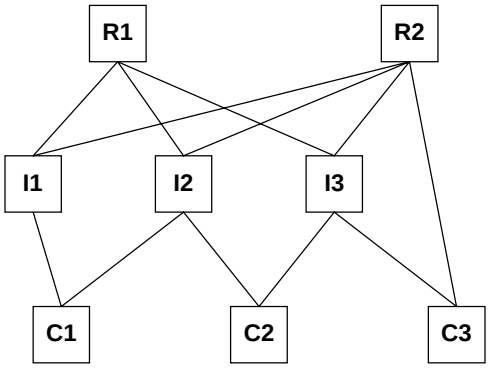

Figure 1: A mesh network comprising root routers (R1-R2), inner XML routers (I1-I3) and clients (C1-C3).

The content carried by routers in a mesh can be statically or dynamically configured. Typically, with static configuration the internal routers carry all of the XML packets available from the root routers. Thus, with a static approach to content configuration clients have a wide choice of routers that can service their request without reconfiguration delay. Unfortunately, such a mesh requires a fixed bandwidth capacity throughout. We can leverage the expressive power of XML to better control bandwidth usage.

Dynamic content configuration allows a router to carry only the packet stream necessary to service its children. In this case, a router disjoins all of the queries it receives from its children and forwards the resulting query to its parent routers. Note that since each router combines the queries of each of its children when subscribing to its parent routers, each router need only store queries for its immediate children. This results in significant bandwidth savings when clients are uninterested in the full contents of the data stream. The disadvantage of this scheme is that the mesh may not have a sufficient number of routers that currently carry the traffic needed by a node searching for an additional parent during mesh construction or repair. If a client requests information that is not available in that portion of the mesh, there will be a delay while the mesh readjusts to supply the required information although this additional startup delay is tolerable in most situations. During reconstruction, the data should be available from the current parent set. During initialization, it simply adds a slight additional startup latency.

Clients wishing to join an $(n-1)$-resilient mesh perform four distinct operations: (1) composing an XML query that describes the data desired, (2) contacting n existing routers that can service the query, (3) sending these n routers the XML query it has composed, and (4) receiving the XML stream described by the query. One particular algorithm for discovering routers is described in Section 4.

Each router includes a query table that describes the portion of the XML stream each of its children wishes to receive. Thus, each router functions as a splitter that takes a single XML stream and refines it for each child. Often a child is only interested in a subset of a stream (such as all air traffic landing in Seattle). Expressing this desire to routers saves last-mile bandwidth and end-host processing.

Our architecture also admits XML *combining routers*. A combining router merges XML feeds from different sources into a single XML feed. This can be accomplished by simply forwarding unmodified packets from both sources, or it can involve application-specific processing. For example, in our air traffic control application we are investigating merging our XML stream of air traffic data with an XML stream of runway conditions.

We will call a node k*-resilient* when any combination of k other nodes and independent links in the mesh can fail and the node will still receive its XML stream. We say a mesh is k-resilient when all of its nodes are k-resilient. The level of resilience in a network can vary according to the needs of end clients. Although we heretofore have described a uniform mesh architecture with a fixed router fan-in of n, it is entirely possible to build meshes with non-uniform fan-in. The only constraint is that in order to assure a desired level of resilience all the way to the root, the resilience of a child's parents must be equal to, or greater than the child's desired resilience. For example, one could build a core network that is 2-resilient, and certain clients could choose to be 1-resilient. The failure of a core router will most likely reduce the resilience level of many peripheral routers and clients until the mesh can reconfigure, but the mesh will continue to provide service to all clients except those clients directly connected only to the failed node. Thus, in certain circumstances, it may make sense to improve the resilience only of key portions of a network that provide service to many clients. We are investigating issues surrounding optimal mesh configuration.

4 Algorithms and protocols

An XML router implements three key algorithms and protocols:

- *XML router core*. The XML core is the engine that receives and forwards packets according to queries. Its job is to efficiently evaluate each received XML packet against all output link queries.
- *Diversity Control Protocol (DCP)*. DCP implements resilient mesh communication by allowing a receiver to reassemble a packet stream from diverse sources.
- *Mesh initialization and maintenance*. A set of algorithms automatically organizes routers and clients into a mesh and repairs the mesh when faults occur.

4.1 XML router core

Figure 2 shows the internal structure of an XML router. An XML router consists of three major components:

- An *input component* that acquires XML streams for presentation to the XML switch. The input component is responsible for maintaining DCP connections to the parents of the router and implementing the mesh initialization and reconfiguration algorithms. In addition, the input component implements data decompression. Our input component can also connect to TCP XML streams for compatibility.

  Although, in many instances, the input component will acquire a single XML stream for routing, an input component could connect to distinct meshes and merge multiple XML streams for routing. The input component is also responsible for forwarding the disjunction of its link queries to its parents.

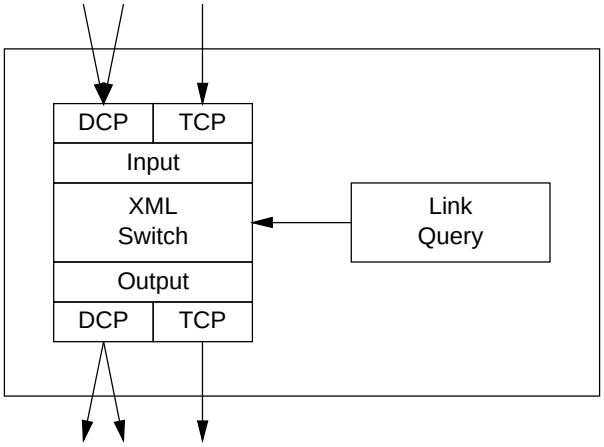

Figure 2: The internal architecture of an XML router comprises the input component, XML switch, and output component. Output link queries control XML packet forwarding.

- An *XML switch* that compares received packets against link queries, and forwards matching packets to the requesting links. An efficient XML switch attempts to combine distinct link queries into a single state machine that matches all of the link queries in a single pass over an incoming packet.
- An *output component* that forwards packets on output links using DCP. In addition, the output component is responsible for handling join requests from prospective children and implements link-based data compression. Our output component additionally can produce TCP XML streams for potential compatibility with non-DCP children.

4.2 Diversity control protocol

The *Diversity control protocol* (DCP) is so named because of the inherent sender diversity that it implements. The essential idea behind DCP is that a receiver can reassemble a packet stream from diverse senders. In DCP, the same stream of packets is sent to a receiver by multiple sources where the desired level of redundancy may vary between nodes in a mesh. As shown in Figure 3, a DCP receiver reassembles the packet stream using the first error-free packet received from any source.

4.2.1 Sequencing

Proper in-order packet stream reassembly requires that all DCP packets be assigned identifiers that admit a total ordering and that the total ordering must be known to the participants. DCP further requires identifiers obey the following invariants:

- For a given content stream, packet identifiers must be associated only with packet content and not be sender specific. This allows receivers to properly reassemble a stream based upon identifiers alone.
- Since packets may travel through a variable number of intermediate router hops, the identifiers with a particular stream must be selected at root routers and remain identifiable

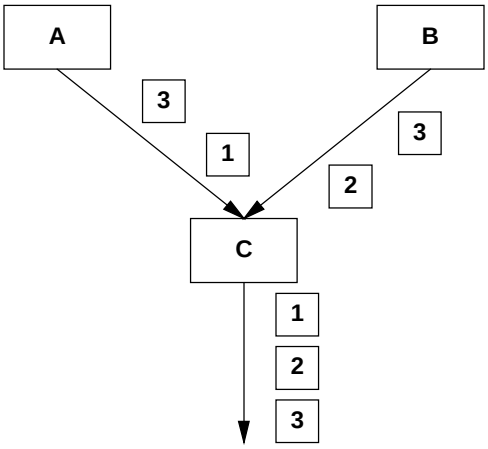

Figure 3: The Diversity Control Protocol (DCP) reassembles a packet stream from diverse senders.

throughout the mesh. Thus, the set of root routers for a particular stream must originate the same packet stream and assign the same identifiers to the same packets. This must be true even if the root routers do not generate the stream at precisely the same time or rate.

- Receiver identifier processing must admit gaps. Since intermediate routers may not forward packets containing content that was not requested by a particular receiver, the identifiers of these packets will not be received.

Our approach to satisfying these three invariants is to assign a monotonically increasing 32-bit application serial number (AN) to every DCP packet when the packet is created at a root router. Every router that forwards DCP packets maintains the last packet AN sent on each output link. The last AN sent on a link is included in the next packet along with the next packet's current AN number. Including a client-specific previous AN in each packet permits a receiver to reassemble the stream of packets from a sender in the presence of missing ANs. In our application, missing ANs are caused by filtered XML packets.

While routers may remove packets from the datagram stream, DCP itself is a reliable transport protocol. Hence, any missing datagram (as indicated by a hole in the AN sequence chain) will be retransmitted. In order to maintain redundancy invariants throughout the mesh, retransmissions are requested at each hop rather than end-to-end. Similarly, packets are buffered and transmitted in-order at each hop. This ensures that every node can consider each parent an independent source of ordered datagrams. We return to consider the implications of out-of-order forwarding in section 6.4.

DCP currently uses UDP as a transport mechanism to facilitate deployment at the application layer. Distinct DCP streams are currently transmitted on separate UDP ports. In our application, one DCP packet is used to transport one XML packet. This is possible because our XML packets are relatively small. If XML packets do not fit into a single IP packet envelope, an AN could describe both the XML packet number being transmitted and the IP packet within the XML packet. The important invariant is that an AN be based upon the content of a packet and not on when or by whom it was generated.

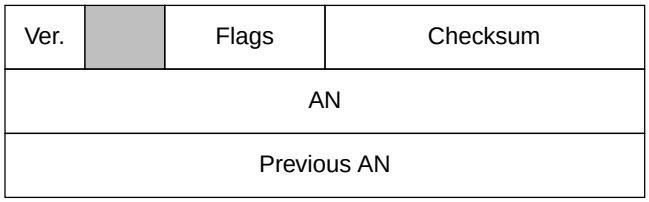

Figure 4: DCP Packet Header

Figure 4 shows the layout of a DCP packet. In addition to the ANs we have already mentioned, a DCP packet includes a 4-bit version number to allow DCP to evolve and a set of 8 bit flags. The flags permit a sender to request an acknowledgment, a receiver to send an acknowledgment or request a retransmission, and for the exchange of keep-alive and connection-establishment and tear-down information. The entire packet is covered by a 16-bit checksum which may be optionally disabled if encapsulated in UDP or when carrying streams insensitive to corruption.

While our use of DCP is as a datagram protocol, DCP is equally well-suited for the transmission of byte streams. A bit in the flags field is used to indicate that DCP is operating in stream mode. When used as a stream protocol, the AN simply refers to the sequence number of the first byte of the datagram, as in TCP. Similarly, the previous AN refers to the last byte of the previous packet in stream mode. Note that this construction allows for fragmentation or reframing of DCP packets if desired, albeit at the expense of additional complexity and buffering at the receivers. Additionally, if multiple root servers are in use each server must take care to sequence the data identically. Datagram and streaming mode cannot be used during the same DCP connection.

4.2.2 Retransmission

When a receiver joins multiple DCP senders, it waits for the first packet to arrive from any one of the hosts and uses the AN of this packet as its current AN. Packets that are subsequently received with a lower AN than the current AN are discarded and packets that are received with an AN in the future are buffered. A packet with the current AN in the previous AN field is considered the next packet in the reassembled stream and the current AN is updated. If a receiver does not receive an appropriate packet after a fixed interval, it sends a negative acknowledgment (NACK) to all senders with its current AN. This retransmission is sent only to the receiver requesting it. In a fashion similar to TCP's fast retransmit, a NACK is generated after a much shorter timeout if a packet with a subsequent AN is received, indicating either a lost or reordered packet. This NACK serves as a request for all senders to retransmit all packets after the receiver's current AN.

Assuming a regular mesh construction (equal numbers of parents and children), the negative acknowledgment process does not suffer from ACK implosion even with high degree. An individual receiver only generates a NACK if an AN is not received from any of its parents. Due to the (assumed) pairwise independence of packet loss between distinct senders and receivers, this probability drops exponentially with degree as discussed in section 5.1.2. Hence, the probability that a sender receives any NACKs at all decreases with increasing degree, avoiding the NACK implosion problem.

Senders transmit packets in order to a receiver and request an acknowledgment from a receiver from time to time. Our current implementation requests positive acknowledgment after a fixed number of packets has been sent. A receiver responds to a request for acknowledgment with an acknowledgment that contains the last AN (or last byte in streaming mode) it has processed. This serves to limit the amount of buffering required at each node and allows for rapid resynchronization of senders and receivers. If the current sender has not yet sent that AN (byte), it squelches its transmissions until after that AN (byte). A receiver can also send an unsolicited acknowledgment to squelch a sender that is behind. In contrast, if a receiver continually fails to respond to acknowledgment requests, or persistently lags behind the sequence space (indicating insufficient bandwidth between sender and receiver), the connection is terminated. The receiver must then reconnect to a new point in the mesh (presumably with a higher-bandwidth link).

4.3 Mesh formation and maintenance

A mesh begins life as a set of root routers that are all capable of supplying an XML stream of interest. We assume that failures of root routers are independent and, thus, each has an independent means of deriving the XML stream. As noted above, however, roots must be uniform in their DCP packetization and sequence number selection. Additional roots may be added to a mesh at any time provided they have a mechanism to synchronize their content stream with the existing root nodes.

Mesh discovery is outside the scope of this document, but one method of distributing the set of root nodes for a particular content stream is through DNS. All of the IP addresses for the root routers for a service could be stored in a DNS address record. For example, `stream.asdi.faa.gov` might be a DNS name that maps to a set of root routers that supply an XML stream of air traffic control data for North America.

4.3.1 Adding routers and clients

When a new internal router is added to a mesh, it can either be statically configured with a set of parents or the new router can select its own parents based upon performance experiments. A wide variety of automatic configuration algorithms can be used to form the mesh depending on the particular desires of the node. These may vary widely depending upon whether the mesh is controlled by a single administrative entity concerned with overall characteristics of the mesh such as its resilience or depth, or the new node has a more specific purpose. Clients join the mesh in the same fashion.

Rather than specify a particular algorithm or policy, we admit a host of possibilities by providing a set of mesh primitives that new nodes can use to discover the topology of the mesh and locate themselves within it. Each router supports the following primitives:

- **Join (Q):** A new node is added as a child of the router with query **Q** provided the current node is willing to admit a child with such a query.
- **Children (Q):** The router responds with its children that subscribe to a subset of **Q**. A full child list may be elicited by specifying a query that matches the entire stream.
- **Parents:** The router responds with its parent set.

1. Initialize the set S to be the root routers.
2. For each node in S, send a join request and remove the node from S.
3. If a node accepts the join, add it to the parent set P. If n nodes are in P, quit.
4. If a node declines the join, ask it for a list of its children, and add them to S.
5. If S is not empty, go to Step **2**.

Figure 5: Parent selection algorithm. Each node runs this algorithm to construct an $(n-1)$-resilient mesh.

Using these three operations, it is possible for a new node to completely walk a mesh to determine its optimal location. We note that the optimum location may vary depending on the particular desires of the joining node. We have currently implemented a very simple algorithm for automatic parent selection for a client seeking $(n-1)$ resilience shown in Figure 5.

This simple algorithm seeks to find a set of routers that are closest to the root routers and uses the timing of responses to select among candidates. The algorithm also assumes that potential parents are configured to reject join requests when they are at maximum desired capacity or they do not wish to service a requested query. We contemplate additional research on improved algorithms that are based upon both depth in the mesh and observed packet latency to select optimal parents for a new child.

Routers may refuse to serve as parents for policy reasons, if they are not receiving the portion of the XML feed necessary to service a new child's query, or if they are over-subscribed. If a prospective parent is not receiving part of the feed necessary for a new child, the prospective parent may be configured to push an expanded query up to its parent, thus propagating the information request up the mesh.

4.3.2 Mesh repair

Our mesh repair algorithm recovers from parent failures. If one of the parents of a node fails, the node actively attempts to join a new parent. The method used to obtain a new parent is currently identical to that used to obtain initial parents with one caveat. To guarantee that a mesh is acyclic, each router maintains a level number that is one greater than the maximum level of all of its parents. A router's level number is established when a router first joins the network. During mesh recovery, a router will only join parents that have a level number that is less than its own level number. If this is not possible during recovery, then a router must disconnect from all of its children and do a cold re-initialization to return to its desired level of resilience.

Our repair algorithm recovers $(n-1)$ resilience of the mesh if a non-root router fails. As discussed earlier, $(n-1)$ resilience is a fundamental property of any acyclic mesh where each child has n parents. This can be seen by forming an acyclic graph that is a dual of a mesh. In this dual graph each child is represented as a vertex that has directed edges to all of the child's parents. The min-cut of this graph is n vertices or edges if each vertex has out degree n. Thus $(n-1)$ nodes or $(n-1)$ distinct paths can fail and a node will still be connected to a root.

Due to occasional internal node failures, a mesh repaired using our algorithm will have a tendency to flatten out over time as nodes are forced to select parents with lower level numbers during each repair process. If the mesh structure is to serve extremely long-running streams, it may be necessary for nodes to occasionally remove themselves from the mesh and select an entirely new location in order to preserve the depth of the mesh and prevent overloading of root nodes. We have not yet explored efficient algorithms for determining when to start this process.

5 Evaluation

We have developed two separate implementations of our XML router. Our full-featured, multi-threaded Java implementation uses DCP for router-to-router and router-to-client communication. We have also implemented a prototype high-performance router based on Click [19]. The goal of our Java implementation was to adequately support our air traffic control application and it does not attempt to maximize absolute performance. In contrast, the Click router attempts to achieve production-grade performance using freely available XML parsing technology. Below, we report our experiences with both routers. We are mainly interested in understanding how routers will behave in a mesh under varying configurations. Thus, our evaluation focuses on the effects of mesh redundancy on DCP reliability and performance. We also provide performance results from our Click-based XML router.

5.1 DCP performance

The Diversity Control Protocol has several attractive features independent of the format of the data stream. In particular, DCP-based meshes can achieve substantially lower effective loss rates and latency than tree-based distribution networks. Further, the redundancy can be utilized to absorb unexpected decreases in the link capacity between nodes. In this section, we quantify these effects using our Java XML router. All results presented in this section represent the average of several experiments each consisting of 1000 to 10000 XML-encoded ASDI packets.

5.1.1 Experimental design

The experimental setup consisted of four 600MHz PIIIs, two running Linux 2.2.14, and two running FreeBSD 4.0. Each machine used 100Mbit Intel EtherExpress Pro100 Ethernet controllers and 128 Mbytes of memory. The roots and all intermediate XML router nodes were run on one Linux machine using Sun's JDK version 1.3. The XML client node was run on the other Linux machine, also with Sun's JDK version 1.3.

For each experiment, the root node received an XML feed containing a 1Kbyte per second substream of the live ASDI flight data described in the following section. While each node in our experimental topology requests the entire test XML stream, it does so by specifying an XML query predicate, hence each packet is parsed by the intermediate nodes as part of the forwarding process. The intermediate nodes connect to the root over the loopback interface, so there was no packet loss. The desired link loss rates between intermediate and client nodes were obtained by routing each DCP connection through one of the FreeBSD machines which passed the packets through an appropriately configured Dummynet [32] tunnel.

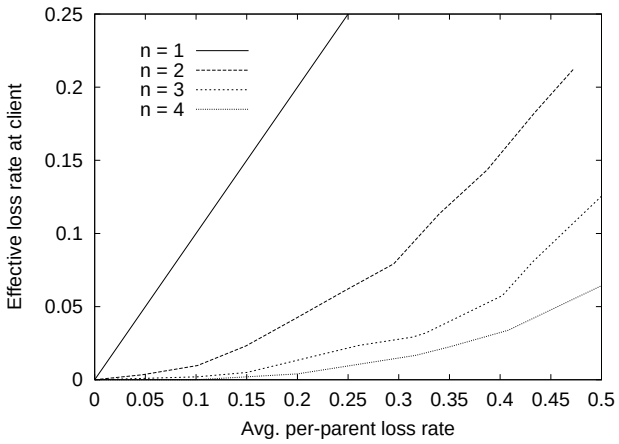

Figure 6: Loss rates experienced by a client as a function of individual parent loss rates and the number of parents.

The one-way latency between the client and parent nodes was negligible (0.1ms) with respect to the millisecond granularity of the Java-based timing mechanisms we used to measure packet latency. Variability in the observed latency of XML packets can be attributed both to the inherent non-uniformity of XML parsing times and to thread scheduling uncertainties of the JDK.

5.1.2 Redundancy reduces loss exponentially

We assume that packet loss is independent across parents. This assumption is false if a problematic portion of the communication path to a set of parents is shared. In our ideal model, if each parent has an identical loss rate, p, a node with n parents should expect a combined loss rate of p^n. Figure 6 verifies this experimentally, showing the DCP loss rate experienced at clients with 2, 3, and 4 parents where the loss rate at each parent is independently identically distributed (i.i.d.) with uniform probability p varying from $[0, 0.5]$.

For a traditional tree-based distribution network, the loss rate experienced at the client corresponds directly to the loss rate of its parent. The graph shows, however, that a mesh topology is able to provide acceptable delivery rates over even extremely lossy channels. A node with four parents can expect a loss rate of less than 5% even if each of the parents individually experiences a loss rate of up to 45%.

Most Internet links do not experience extremely high loss rates. In fact, typical long-term average loss rates are on the order of 2–5% with substantially higher burst rates [29]. In such cases, a mesh with $n = 2$ still limits the loss rate at the client to substantially less than 1%. Decreased loss rate is not the only gain from multiple parents, however.

5.1.3 Latency

Even in cases where acceptable loss rates can be provided by a tree-based network (reliability may be assured by retransmission), significant improvements in latency can be achieved by increasing redundancy. In DCP, loss is not detected until the receipt of a later packet since each individual packet is not acknowledged. However,

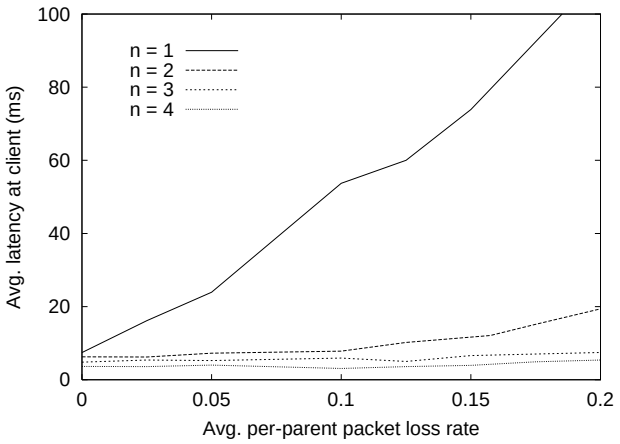

Figure 7: Average per-packet latency from root to client as a function of individual parent loss rates and the number of parents.

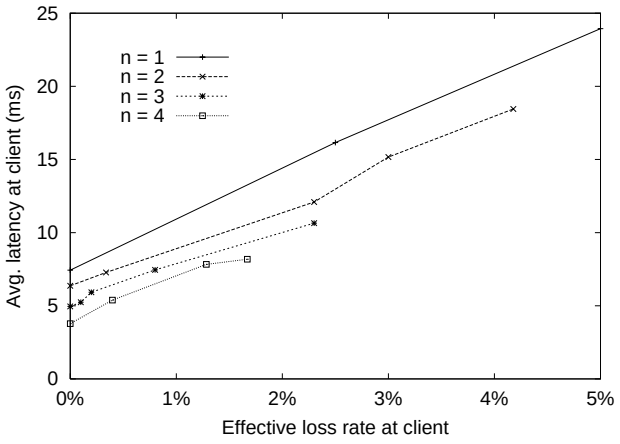

Figure 8: Average per-packet latency from root to client for a range of *effective* loss rates. Effective loss rates for each level of redundancy are taken from Figure 6.

because we expect the Internet to reorder packets [29], a packet is not assumed lost until some time after its successor arrives (currently 5ms).

In order to magnify the effects of retransmissions in a LAN environment with short round-trip times, our test XML feed was specifically constructed to have relatively long inter-packet intervals. In our experiments, a single retransmission adds approximately 300ms to the latency for the lost packet. In practice, streams are likely to have a shorter inter-arrival period but longer RTTs, resulting in a similar effect. As can be seen in Figure 7, packet loss significantly impacts the average packet latency at the client for non-redundant configurations. Meshes with higher levels of redundancy perform much better.

A redundant topology performs even better than the effective loss rate of Figure 5 suggests. This is because clients with multiple parents use the *first* copy of each XML packet they receive. In general, the expected minimum of multiple samples from any distribution

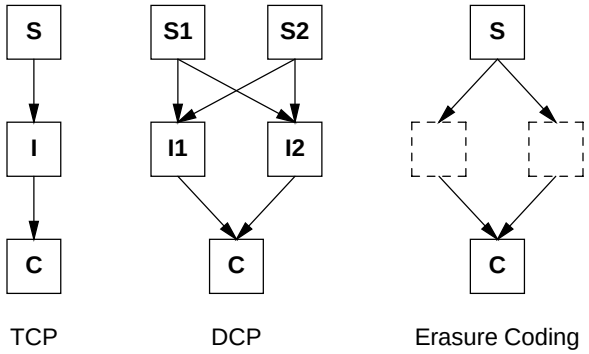

Figure 9: Experimental multi-tier topologies. A two-hop TCP path with a TCP splice in the middle, a 1-resilient DCP mesh of depth two, and an erasure code using two disjoint paths of length two with simple forwarding at the intermediate nodes. The loss rates on all links is identical.

is guaranteed to be at least as good as expected value of a single sample. Figure 8 shows the average latency for several levels of redundancy with respect to *effective* loss rates.

5.1.4 Multi-tier meshes

The improvement in latency and throughput becomes even more dramatic as the the depth of the mesh increases. We demonstrate this by measuring the throughput performance of a two-tier mesh using both DCP and TCP and analytically derive the expected performance of a carousel-based erasure code scheme. As shown in Figure 9, our experimental 1-resilient DCP mesh has five nodes: two servers delivering identical streams, two intermediate nodes, and one client. In the case of TCP, the mesh has only three nodes: a server, client, and one intermediate node that splices the two separate TCP connections. In both cases, the client, server, and intermediate nodes are connected with point-to-point Ethernet links. We analyze the performance of erasure coding over a hypothetical topology consisting of a server and client connected by two disjoint, two-hop paths. The nodes in the middle simply forward packets and, unlike TCP and DCP, do not request retransmissions of lost packets.

In our experiments we used Dummynet to limit the bandwidth of each link to 75Kbits per second and set the server to transmit data in 262-byte bursts at a rate of 19Kbits per second—significantly under link capacity. Each link in the mesh has a one-way latency of 10ms. Figure 10 shows the throughput observed at the client as the loss rate is adjusted for all links uniformly. Note that TCP's throughput drops rapidly as the loss rate increases. The redundant links are of no use to TCP as a duplicate TCP connection on a redundant path would suffer the same fate. DCP, on the other hand, is able to utilize both links at the same time to provide successful transfer at much higher loss rates.

We note that in the case of multi-hop networks with sufficient bandwidth, DCP outperforms carousel-based erasure coding techniques such as those used by Digital Fountain [9]. Such schemes do not retransmit lost packets. Instead, they encode the data stream at a fixed rate using an erasure code which enables any lost packets to be recovered by simply receiving an additional number of encoding

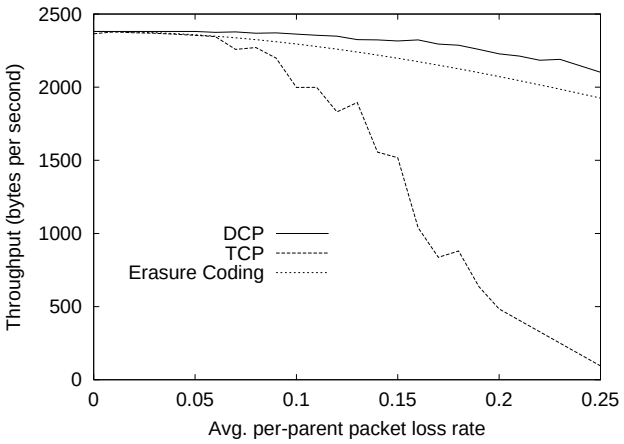

Figure 10: Observed throughput of a two-tier mesh with uniform link loss rates using both 1-resilient DCP and TCP. The stream is served in 262-byte chunks at a rate of 2381 bytes per second. DCP downloads utilize two parents at each tier while TCP can support only one at each tier. We also plot the expected performance of a simple carousel-based erasure code using two disjoint, two-hop paths.

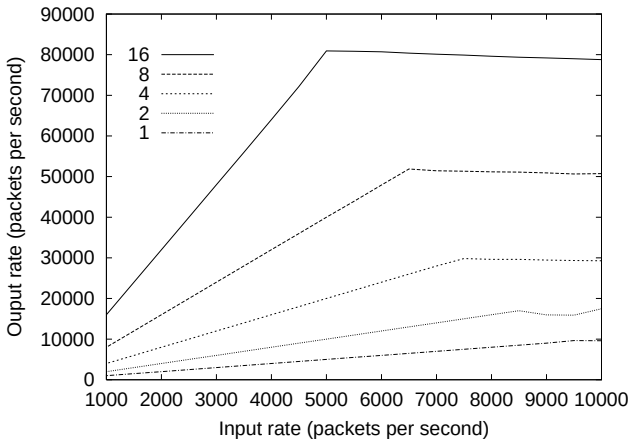

Figure 11: Forwarding rates for a Click-based XML router with a varying number of children. Each child requested the entire input XML stream specified through a trivial XPath expression.

packets. A maximum-distance-separable erasure code requires that the client simply receive as many packets as comprised the original data stream, regardless of which packets they are. In practice, many codes (including those used by Digital Fountain) are not quite maximum-distance-separable, requiring a few additional packets.

Because carousel erasure coding is typically deployed end-to-end with no retransmissions within the network, the loss rates at each hop are cumulative. Whereas, DCP reassembles the stream and retransmits a full set of redundant packets at each tier of the mesh. A naive carousel-based distribution network could support applications such as ours by ensuring each packet contains all the data necessary to decode the current input packet plus any additional redundant data required to support the erasure code.

A carousel erasure code can utilize redundant links by sending an encoded version of each data packet down all available links. Hence, we can calculate an upper bound on the performance of such an erasure code by assuming only one packet of any encoding set must be received. Given a distribution network with n separate paths, each comprised of l hops with link loss rate p, it is easy to see that each path successfully delivers the packet with probability $(1-p)^l$. In the best case, each data packet can be successfully decoded by the client if only one of the encoding packets is received, which occurs with probability $1-(1-(1-p)^l)^n$. To recover lost packets, the client must receive additional encoding packets which are lost with the same probability. Hence, the throughput of such a scheme can be computed by simply multiplying the input data rate by the effective reception rate. Using this formula, Figure 10 shows the expected performance of a sufficiently low-rate maximum-distance-separable erasure code over the two-tier topology shown in Figure 9.

5.2 XML routing performance

Figure 11 shows the forwarding rates achieved by our Click XML router installed as a kernel module. We measured the forwarding capacity by generating a constant stream of identical UDP packets containing a 262-byte XML-encoded ASDI flight update (similar to the one in Figure 12) at a fixed rate and sending them to our XML router. The router parses each XML packet, applies the appropriate child predicates, and forwards the packet to the children with matching predicates.

The tests were conducted on an 800Mhz dual-processor Intel PIII in uni-processor mode with two Ethernet controllers: an on-board Intel EtherExpress PRO 100Mbit/s PCI controller and an Intel PRO/1000 Gigabit Ethernet PCI card. We use uni-processor mode because our XML parsing code is not known to be SMP-safe. Packets were received on the 100Mbit interface and forwarded out the Gigabit interface. Because Click does not support polling on the 100Mbit controller packet input was interrupt driven.

The maximum loss-free forwarding rate varies with the number of children. Additional children add processing overhead for additional link queries. With only one client and a simple query expression, our implementation is able to forward slightly more than 9,000 262-byte packets per second, or about 19 Mbit/second. The more complicated the packet and expression, the slower the forwarding rate.

In order to better understand the impact of query complexity, we timed the XML parser and query evaluator separately. Our Click XML router uses the Gnome XML library, *libxml*. The library provides both an XML parser and an XPath (a subset of XQuery) evaluator. We have made no attempts to optimize the performance of this library. Thus, our measured performance represents a lower bound, and we expect an efficient implementation could perform much better. Table 1 shows the time taken to apply a variety of queries to the same 262-byte sample XML flight packet. We find that complex expressions can take over twice the time to evaluate in the context of an XML packet than simple ones. In all cases, packet processing cost is dominated by XML parsing time.

XQuery	Time (μs)
Parse	64.2
true()	4.5
/flight/flightleg/altitude > 300	7.1
starts-with(string(/flight/id),'TWA')	8.9
substring-before(string(/flight/flightleg \ /coordinate/lat),'N') > 2327	14.5

Table 1: Time to evaluate various queries in an 800Mhz PIII. Parsing a standard 262-byte XML flight update requires 64.2μs. The four XQuery expressions shown here select the entire feed, flights above 30,000 feet, Trans World Airlines flights, and flights currently north of the Tropic of Cancer, respectively.

5.3 Experience with air traffic control data

Our original motivation for developing XML routers was to build an infrastructure for distributing and processing real-time air traffic control data. Our laboratory receives the Aircraft Situational Display to Industry (ASDI) [40] feed via a private IP intranet connection to the U.S. Department of Transportation (DOT). The ASDI feed provides detailed information about the state of North American airspace. ASDI messages include information on flight plans, departures, flight location, and landings. A position update is received approximately once a minute for all enroute aircraft. The ASDI feed is directly distributed to most major airlines and is used for collaborative planning between the FAA and the airlines.

The ASDI feed as distributed by the DOT is encoded in ASCII with a specific compact character encoding for each ASDI message type. Efforts were made to make native ASDI messages a compressed format by virtue of their terseness. The ASDI feed is the union of feeds from multiple Air Route Traffic Control Centers (ARTCCs) and countries (USA & Canada). Unfortunately, messages that cannot be parsed using the ASDI specification arise. Thus, at the outset of our work, we built an ASDI feed parser and carefully gathered examples of non-standard messages. We slowly tuned our ASDI feed parser to handle undocumented cases and, today, still find the occasional new message format.

Early in our work, we decided to convert each ASDI message into a corresponding XML packet to create an XML packet stream. This decision guaranteed that all of our applications would have an easy to parse and well-defined XML DTD to consume. Furthermore, it centralized our interpretation of the ASDI feed so that it could be updated as new undocumented message types were identified. We call the XML stream that is created from the ASDI feed the XML ATC stream. Figure 12 shows a sample flight in both ASDI encoding and our XML encoding.

The XML-encoded ASDI packets contain widely varying amounts of data depending on the type of event being reported: flight departures, arrivals, position updates, or other auditing information. Packet size ranges from around 250 bytes to almost 1000 bytes, for an average of about 350 bytes per packet. The stream is diurnal, peaking in the early evening with an average packet inter-arrival time of about 14ms, resulting in an XML data stream of about 25 Kbytes per second.

ASDI Format:

```
153014022245CCZVTZ UAL1021 512 290 4928N/12003W
```

XML Format:

```
<?xml version="1.0"?>
<messageid>153014022245CCZVTZ</messageid>
<flight>
  <id>UAL1021</id>
  <flightleg status="active">
    <speed type="ground">512</speed>
    <altitude type="reported" mode="plain">
      290
    </altitude>
    <coordinate>
      <lat>4928N</lat>
      <lon>12003W</lon>
    </coordinate>
  </flightleg>
</flight>
```

Figure 12: The same flight data formatted in ASDI and XML. In practice, we omit the Message ID field from the XML encoding.

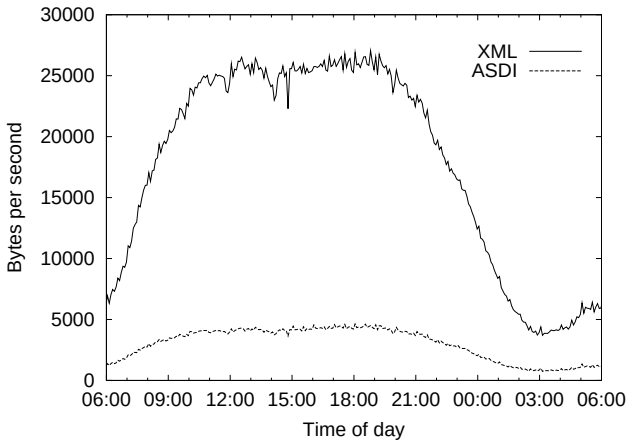

Figure 13: Average bandwidth utilization of the full XML stream and the native ASDI format vs. time of day. In both cases the Message ID field (see Figure 12) is removed from all packets at the root nodes.

Our primary concern in converting the ASDI feed to XML was potential bandwidth bloat. Figure 13 shows the bandwidth of the ASDI feed in both native and XML formats averaged over five minute intervals. Simply converting the feed to XML results in approximately a four-fold increase in bandwidth when compared to the native ASDI feed. We ran both the XML and native ASDI streams through a Lempel-Ziv [21] data compressor. Figure 14 shows the bandwidth of compressed forms of the same streams shown in Figure 13. While the ASDI feed compresses over a factor of two, the XML feed compresses over a factor of 10. The net result is a compressed XML ATC stream is only slightly larger than a compressed ASDI feed and more efficient than the raw ASDI feed.

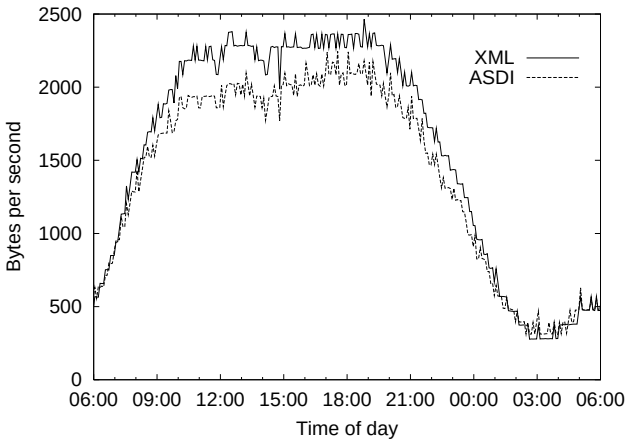

Figure 14: Average bandwidth utilization of the compressed XML stream and the compressed ASDI format vs. time of day. Again, in both cases the Message ID field is removed from all packets at the root nodes.

We have run mesh networks with two root routers and four internal routers but a single root router is more typical. This is because our DOT link is a single point of failure and terminates at our root router(s). We are adding a second communication line to the DOT to connect to their backup ASDI system. This will enable us to have two root routers with independent failure modes. Application serial numbers (ANs) in our ATC application are provided by the FAA. Hence, synchronizing multiple roots is straightforward. Each ASDI message includes a Message ID that we use as the AN of the corresponding XML packet.

Figure 15 shows one interface to the XML ATC stream. This graphical client implements DCP and connects to our XML router mesh. The panel on the left of the screen can be used to control the display of aircraft information. Different colors are used to depict aircraft altitude and the client will coast the position of an aircraft between position updates. For our particular application domain of air traffic control data, XML proved to be a robust and efficient mechanism for distribution. We anticipate adding new types of clients, including an XML stream recorder, to our current system.

6 Discussion

This section considers the strengths and weaknesses of our approach to content routing using XML. While we believe that many of the techniques we developed for our ATC application are widely applicable, we would like to make our assumptions clear.

6.1 AN generation

One difficulty in providing redundant packet sources is providing a standardized sequence space for packet streams that obey the three invariants we outlined in Section 4.2.1. Often, application-specific solutions will present themselves, such as source-derived sequence numbers or time codes. However, in the absence of application-provided sequence numbers, it is necessary to use other approaches, such as cumulative byte counts, block fingerprint matching [23, 38, 39], or other derived metrics.

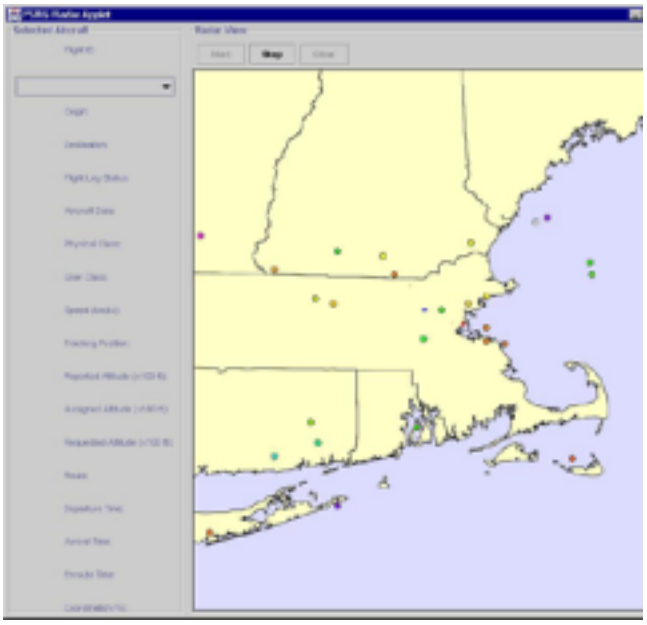

Figure 15: Java-based client for XML ATC stream showing controls and air traffic.

When a combining router merges XML streams, packets in the combined stream must have appropriate ANs. Simply using the ANs from the original uncombined packets for packets in the combined stream will typically not work as packets from different streams will in general have incomparable ANs. One solution is for the combining router to become the root of a new stream and establish its own totally ordered AN space. However, this would create a single point of failure if sequence assignments in this space are not coordinated with another combining router. Another approach is to make AN space partially ordered. For example, the AN space for a combined stream could be a pair of the AN of the source packet in its original stream along with an integer suffix that identifies the source XML stream. Packets with ANs from different source streams would not be sequenced across streams, but a client could recover the ordering of XML packets within each stream.

6.2 Flow control

Nodes are responsible for monitoring the loss rate of streams from their parent and adjusting their predicates appropriately. Limited per-child buffering is available at each node, and clients may be disconnected if they are consequently unable to consume the data stream at an acceptable rate.

The squelching mechanism of DCP allows parents to avoid wasting bandwidth sending packets to a child that the child has already received. If a child is unable to keep up with the long-term average rate of the stream, however, queues will build up and action must be taken. If the client is able to subsist with a smaller subset of the data stream, it may wish to conduct join experiments in order to determine the appropriate XML query for its bandwidth constraints [26, 42]. Otherwise, clients persistently unable to keep up with the data stream will be disconnected by their parents.

6.3 Redundancy

We expect that most mesh networks will use $n = 2$. This level of redundancy allows for single points of failure and allows mesh repair to proceed without stream flow interruption. We expect that as future networks increase in capacity a moderate amount of packet redundancy will be acceptable for high-value streams to achieve specific reliability and performance goals. Secondary storage is often replicated for similar reasons.

We have assumed in our analysis that errors from different parents are independent. This assumption can be violated in numerous ways, but the most likely reason will be shared communication path components from a child to its parents. In addition, network-wide effects, such as distributed-denial-of-service attacks, could cause independent parents to have dependent packet losses. To maximize link independence, we plan to explore using routers in distinct Internet autonomous systems (ASs) and ensuring that last-mile bandwidth is adequate to each AS. In certain applications, it may also be possible to use private intranets to better control error assumptions.

6.4 Router XML stream reassembly

Each of our routers recreates the original XML stream before it is processed by the XML switch. We do this to guarantee that every XML packet is forwarded by every router, to allow a client to ask for retransmissions from any of its parents, and to potentially allow the XML switch to keep stream-dependent state between packets that could be used by queries. The amount of buffering required is bounded by requiring positive acknowledgments as discussed in section 4.2.2.

If XML packets are forwarded out-of-order by an XML switch then a router does not necessarily need to buffer packets or recreate the original sequenced XML stream. This is, indeed, the case in our ATC application, although in our ATC application every XML router does recreate the original XML stream. If an XML router need not recreate the original XML stream, a router could process each received packet independently and would not need to process every packet in an XML stream. In this scenario, a client places increasing reliance upon the redundancy of the mesh to ensure timely delivery of packets that are not received from a particular router. In particular, since all levels may forward out of sequence, the latency induced by a retransmission request from the client may be large. Hence, we have not yet considered how to handle reliable, out-of-order delivery with bounded latency.

6.5 Packet acknowledgments

For asynchronous, variable-bandwidth data streams, packet loss can be detected either by the lack of packet acknowledgments at a sender or by a gap in packet sequence at a receiver. DCP currently relies upon the latter technique. If inter-arrival times are large, per-packet ACKs may be required to provide the appropriate level of responsiveness. Unfortunately, positive acknowledgment schemes admit a well-known implosion problem where the sender is flooded with acknowledgments from each of its children.

While our implementation currently uses unicast UDP to transport DCP packets, DCP could employ IP Multicast where available. The negative acknowledgment system we describe is capable of handling IP Multicast packet losses. If IP Multicast were employed, a DCP output component would send a single packet to an appropriate multicast group of its children based upon the children's queries.

6.6 Dynamic timer adjustment

A robust DCP implementation should be able to automatically adjust its timers to the characteristics of the link between nodes. In particular, the negative acknowledgment timer should be set only long enough to admit observed packet reordering, which clearly depends on the inter-packet arrival of the flow. Being too slow results in poor latency, being too jumpy results in wasted bandwidth. Similarly, several timers relating to mesh liveliness would benefit from automatic refinement. In particular, nodes expect to receive data from their parents every so often. If no data is available in that interval, the parent sends a keep-alive message. A similar mechanism is employed by the parent to insure the continued presence of its children. Clearly the timer should be proportional to the stream data rate, in order to avoid excessive probing. We are currently exploring applying known techniques to these problems [18].

7 Conclusions and future work

This paper presented three key ideas. First, we introduced the idea of XML routers that switch self-describing XML packets based upon any field. Second, we showed how XML routers can be organized into a resilient overlay network that can tolerate both node and link failures without reconfiguration and without interrupting real-time data transport. Finally, we introduced the Diversity Communication Protocol as a way for peers to use redundant packet transmissions to reduce latency and improve reliability.

A wide variety of extensions can be made to the work presently reported, both in protocol refinements and additional functionality. We are actively investigating methods of DCP self-tuning, both for adaptive timers and sophisticated flow control. DCP can also can be used for uninterpreted byte streams. Thus, DCP-like ideas may find application in contexts outside of XML routers. For example, contemporary work on reliable overlay networks (RONs) could use DCP as a RON communication protocol to maximize performance and reliability [2].

Just as secondary storage has become viewed as expendable in pursuit of enhanced functionality and performance [34], we believe that, for certain tightly-constrained applications, network bandwidth across multiple paths may be similarly viewed as well-spent in return for substantial gains in reliability and latency. It is unlikely that multiple disjoint paths with excess capacity will always exist on the last mile to a client. Hence, many installations may benefit from meshes that change to lower levels of redundancy at critical network points such as points-of-presence before last mile cable.

Within the scope of XML routing, our current XML routers could be extended to support.

- More sophisticated XML mesh building and maintenance algorithms.
- Combiners that integrate multiple XML streams for multicast transport as a single stream.
- Using XML routers for duplex communication.

- Other XML network components, such as stream storage and replay.
- Transcoding XML routers that produce output packets that are derivatives of input packets, based upon client queries.

Even in its current form, however, we believe our architecture demonstrates XML is a viable mechanism for content distribution, providing a natural way to encapsulate related data, and a convenient semantic framing mechanism for intelligent network transport and routing.

Acknowledgments

We would like to thank Qian Z. Wang and Micah Gutman for their work on an early version of the XML router and the graphical ASDI client shown in Figure 15. The DCP experiements were conducted at emulab.net, the Utah Network Emulation Testbed, which is primarily supported by NSF grant ANI-00-82493 and Cisco Systems. We are indebted to Benjie Chen and the members of the Click project for assistance with benchmarking our Click-based XML router. This paper greatly benfited from comments on earlier drafts by Chuck Blake, Frans Kaashoek, the anonymous reviewers, and our shepherd, Maurice Herlihy. We also remember Jochen Liedtke, Bruce Jay Nelson, and Mark Weiser as great life forces and friends.

References

[1] XMLBlaster. `http://www.xmlblaster.org/`.

[2] ANDERSEN, D. G., BALAKRISHNAN, H., KAASHOEK, M. F., AND MORRIS, R. T. Resilient overlay networks. In *Proc. ACM SOSP* (Oct. 2001).

[3] ARMSTRONG, S., ET AL. Multicast transport protocol. RFC 1301, Internet Engineering Task Force, 1992.

[4] BANAVAR, G., CHANDRA, T., MUKHERJEE, B., NAGARAJARAO, J., STROM, R., AND STURMAN, D. An efficient multicast protocol for content-based publish-subscribe systems. In *Proc. Int'l Conf. on Dist. Comp. Systems (ICDCS)* (May 1999).

[5] BANERJEA, A. Simulation study of the capacity effects of dispersity routing for fault tolerant realtime channels. In *Proc. ACM SIGCOMM* (Aug. 1996), pp. 194–205.

[6] BESTAVROS, A. An adaptive information dispersal algorithm for time-critical reliable communication. In *Network Management and Control, Volume II*, I. Frish, M. Malek, and S. Panwar, Eds. Plenum Publishing Co., New York, New York, 1994, pp. 423–438.

[7] BRAY, T., ET AL. Extensible markup language 1.0 (second edition). `http://www.w3.org/TR/REC-xml/`, W3C Recommendation, 2000.

[8] BYERS, J. W., LUBY, M., AND MITZENMACHER, M. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *Proc. IEEE Infocom* (Mar. 1999), pp. 275–283.

[9] BYERS, J. W., LUBY, M., MITZENMACHER, M., AND REGE, A. A digital fountain approach to reliable distribution of bulk data. In *Proc. ACM SIGCOMM* (Sept. 1998), pp. 56–67.

[10] CARZANIGA, A., ROSENBLUM, D. S., AND WOLF, A. L. Achieving scalability and expressiveness in an Internet-scale event notification service. In *Proc. ACM PODC* (July 2000), pp. 219–227.

[11] CHAMBERLIN, D., ET AL. XQuery 1.0: An XML query language. `http://www.w3.org/TR/xquery/`, W3C Working Draft, 2001.

[12] CHAWATHE, Y., MCCANNE, S., AND BREWER, E. RMX: Reliable multicast for heterogeneous networks. In *Proc. IEEE Infocom* (Mar. 2000), pp. 795–804.

[13] CHU, Y., RAO, S. G., AND ZHANG, H. The case for end system multicast. In *Proc. ACM SIGMETRICS* (June 2000), pp. 1–12.

[14] FLOYD, S., JACOBSON, V., MCCANNE, S., LIU, C.-G., AND ZHANG, L. A reliable multicast framework for lightweight sessions and application level framing. *IEEE/ACM Trans. on Networking 5*, 6 (Dec. 1997), 784–803.

[15] HOLBROOK, H. W., AND CHERITON, D. R. IP multicast channels: EXPRESS support for large-scale single-source applications. In *Proc. ACM SIGCOMM* (Aug. 1999), pp. 65–78.

[16] JANNOTTI, J., GIFFORD, D. K., JOHNSON, K., KAASHOEK, M. F., AND O'TOOLE, J. Overcast: Reliable multicasting with an overlay network. In *Proc. USENIX OSDI* (Oct. 2000), pp. 197–212.

[17] KADANSKY, M., CHIU, D., AND WESLEY, J. Tree-based reliable multicast [TRAM]. Technical report TR-98-66, Sun Microsystems Lab, 1998.

[18] KARN, P., AND PARTRIDGE, C. Improving round-trip time estimates in reliable transport protocols. *ACM CCR 17*, 5 (Aug. 1987), 2–7.

[19] KOHLER, E., MORRIS, R., CHEN, B., JANNOTTI, J., AND KAASHOEK, M. F. The click modular router. *ACM Trans. on Computer Systems 18*, 3 (Aug. 2000), 263–297.

[20] LABOVITZ, C., AHUJA, A., ABOSE, A., AND JAHANIAN, F. Routing stability and convergence. In *Proc. ACM SIGCOMM* (Aug. 2000), pp. 115–126.

[21] LEMPEL, A., AND ZIV, J. A universal algorithm for sequential data compression. *IEEE Trans. on Information Theory 23*, 3 (May 1977), 337–343.

[22] LIN, J.-C., AND PAUL, S. RMTP: A reliable multicast transport protocol. In *Proc. IEEE Infocom* (Mar. 1996), pp. 1414–1424.

[23] MANBER, U. Finding similar files in a large file system. In *Proc. Winter USENIX* (Jan. 1994), pp. 1–10.

[24] MAXEMCHUK, N. F. *Dispersity Routing in Store and Forward Networks*. PhD thesis, University of Pennsylvania, May 1975.

[25] MCAULEY, A. J. Reliable broadband communication using a burst erasure correcting code. In *Proc. ACM SIGCOMM* (Sept. 1990), pp. 297–306.

[26] MCCANNE, S., AND JACOBSON, V. Receiver-driven layered multicast. In *Proc. ACM SIGCOMM* (Aug. 1996), pp. 117–130.

[27] MOSER, L., MELLIAR-SMITH, P., AGARWAL, D., BUDHIA, R., AND LINGLEY-PAPADOPOULOS, C. Totem: A fault-tolerant multicast group communication system. *C. ACM 39*, 4 (Apr. 1996), 54–63.

[28] OKI, B., PFLUEGL, M., SIEGEL, A., AND SKEEN, D. The information bus — an architecture for extensible distributed systems. In *Proc. ACM SIGOPS* (Dec. 1993), pp. 58–68.

[29] PAXSON, V. End-to-end internet packet dynamics. *IEEE/ACM Trans. on Networking 7*, 3 (June 1999), 277–292.

[30] PENDARAKIS, D., SHI, S., VERMA, D., AND WALDVOGEL, M. ALMI: An application level multicast infrastructure. In *Proc. USENIX Symp. on Internet Technologies and Systems (USITS)* (Mar. 2001), pp. 49–60.

[31] RABIN, M. O. Efficient dispersal of information for security, load balancing and fault tolerance. *J. ACM 36*, 2 (Apr. 1989), 335–348.

[32] RIZZO, L. Dummynet: a simple approach to the evaluation of network protocols. *ACM CCR 27*, 1 (Jan. 1997).

[33] RIZZO, L., AND VICISANO, L. A reliable multicast data distribution protocol based on software FEC techniques. In *Proc. IEEE HPCS* (June 1997).

[34] SANTRY, D. J., FEELEY, M. J., HUTCHINSON, N. C., AND VEITCH, A. C. Elephant: The file system that never forgets. In *Proc. Workshop on Hot Topics in Operating Systems (HotOS-VII)* (Mar. 1999).

[35] SAVAGE, S., ANDERSON, T., AGGARWAL, A., BECKER, D., CARDWELL, N., COLLINS, A., HOFFMAN, E., SNELL, J., VAHDAT, A., VOELKER, J., AND ZAHORJAN, J. Detour: a case for informed internet routing and transport. *IEEE Micro 19*, 1 (Jan. 1999), 50–59.

[36] SEGALL, B., ARNOLD, D., BOOT, J., HENDERSON, M., AND PHELPS, T. Content based routing with Elvin4. In *Proc. AUUG2K* (June 2000).

[37] STOICA, I., NG, T. S. E., AND ZHANG, H. Reunite: A recursive unicast approach to multicast. In *Proc. IEEE Infocom* (Mar. 2000), pp. 1644–1653.

[38] TRIDGELL, A. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, Australian National University, Apr. 2000.

[39] TRIDGELL, A., AND MACKERRAS, P. The rsync algorithm. Tech. Rep. TR-CS-96-05, Australian National University, 1997.

[40] VOLPE NATIONAL TRANSPORTATION CENTER, AUTOMATION APPLICATIONS DIVISION. Aircraft situation display to industry functional description and interfaces. DTS-56 report, Aug. 2000.

[41] WHETTEN, B., AND TASKALE, G. An overview of reliable multicast transport protocol II. *IEEE Network 14*, 1 (Jan. 2000), 37–47.

[42] WU, L., SHARMA, R., AND SMITH, B. Thin streams: An architecture for multicasting layered video. In *ProcİEEE Int'l Workshop on Network and Operating System Support for Digital Audio and Video* (May 1997).

[43] YAVATKAR, R., GRIFFIOEN, J., AND SUDAN, M. A reliable dissemination protocol for interactive collaborative applications. In *Proc. ACM Conf. on Multimedia* (Nov. 1995), pp. 371–372.

In Proceedings of IEEE GlobeCom. Rio de Janeiro, December 1999

ADAPTIVE INVERSE MULTIPLEXING FOR WIDE-AREA WIRELESS NETWORKS

Alex C. Snoeren

snoeren@lcs.mit.edu
Laboratory for Computer Science
Massachusetts Institute of Technology

Abstract

The limited bandwidth of current wide-area wireless access networks (WWANs) is often insufficient for demanding applications, such as streaming audio or video, data mining applications, or high-resolution imaging. Inverse multiplexing is a standard application-transparent method used to provide higher end-to-end bandwidth by splitting traffic across multiple physical links, creating a single logical channel. While commonly used in ISDN and analog dialup installations, current implementations are designed for private links with stable channel characteristics.

Unfortunately, most WWAN technologies use shared channels with highly variable link characteristics, including bandwidth, latency, and loss rates. This paper presents an *adaptive* inverse multiplexing scheme for WWAN environments, termed *Link Quality Balancing*, which uses relative performance metrics to adjust traffic scheduling across bundled links. By exchanging loss rate information, we compute relative short-term available bandwidths for each link. We discuss the challenges of adaptation in a WWAN network, CDPD in particular, and present performance measurements of our current implementation of Wide-Area Multi-Link PPP (WAMP) for CDPD modems under both Constant Bit Rate (CBR) and TCP loads.

1 Introduction

A large number of wide-area wireless access network (WWAN) technologies have recently emerged, including Metricom's Ricochet Packet Radio network, *Global System Mobile* (GSM) [13], IS-95 [10], and Cellular Digital Packet Data (CDPD) [15]. The defining characteristic of WWANs is their use of a shared channel with long and variable round-trip times (RTTs), typically on the order of 500ms, coupled with a relatively low and variable bandwidth (usually in the tens of Kb/s). Additionally, many of these wireless technologies support link-level ARQ schemes for reliable transmission. While such mechanisms provide better line error characteristics, they combine with channel access contention to introduce significant delay variability.

The low bandwidths provided by WWAN technologies are often insufficient to support demanding applications, such as voice recognition and high-resolution imaging. In these contexts, one obvious solution is to spread connections over multiple physical links. By *striping* data over many physical channels (called a *bundle*), possibly by breaking packets up into fragments, it is possible to present a single logical channel with increased available bandwidth. When using such techniques, however, variations in the channel characteristics of any particular link can have catastrophic impact on the performance of the bundle as a whole. In particular, variations in the perceived transmission delay of an individual member link may cause delays in demultiplexing and packet assembly. In the absence of sophisticated scheduling techniques to balance delay, bandwidth variability between member links can cause the performance of many transport protocols to be throttled by the slowest member link, even when traffic is allocated in proportion to link speeds. Similarly, losses on one link may prevent packet reassembly, increasing loss rates markedly. These *amplification effects* motivate our design of an adaptive inverse multiplexing algorithm in WWAN networks. In this paper, we study the Bell Atlantic Mobile CDPD network, but we believe our work is applicable to all WWAN technologies, and, more generally, to any shared links with variable characteristics.

Inverse multiplexing is fairly common in current network technologies. Most implementations, however, assume physical transport mechanisms with constant bit rates and stable link characteristics such as those found in circuit switched networks [1, 6, 7]. This is not the case with CDPD. Commercial CDPD networks use an IP-based transport mechanism such as TCP or UDP to transport data. These mechanisms are subject not only to the expected loss and delay characteristics of the Internet, but additional WWAN-specific issues as well, such as increased round-trip times and delay variability. The added synchronization requirements of an inverse multiplexing scheme only compound any losses or variability in bandwidth or delay.

This work was supported in part by Defense Advanced Projects Research Agency (DARPA) contract number DAAN02-98-K0003. The author is also supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

Reliable transport protocols require reasonable limits on packet reordering, jitter, and loss in order to function optimally. In particular, TCP has been shown to be extremely sensitive to variations in delay, as they affect both its ACK-based clock and RTT estimates for packet retransmission [3, 4, 5]. The Fast Retransmit algorithm [RFC2001] also makes assumptions about the level of packet reordering, and induces spurious retransmissions in the face of excessive reordering. It is therefore important to ensure that any multi-link scheme limit packet reordering and delay jitter as much as possible.

The multiplexor must then schedule packet fragments so that they are received and reassembled in roughly the same order they arrived, with a minimum of added delay. This requires balancing the fragment distribution according to the current *bandwidth*delay* product of each individual link. Due to the depth of network queues relative to the *bandwidth*delay* product, and channel access asymmetries inherent in CDPD, it is extremely difficult to obtain accurate measurements. Instead, we use relative performance metrics to adjust the traffic distribution between links, and allow the end-to-end transport protocol's bandwidth probing algorithm to function as it would normally. We term this dynamic adaptation *Link Quality Balancing*.

The rest of this paper is organized as follows. Section 2 details related work on improving performance in similar WWAN environments. Section 3 sets forth the assumptions made concerning the CDPD network used in our study. In Section 4 describes the details of our inverse multiplexing scheme. We examine the factors affecting TCP performance over multiplexed CDPD links in Section 5, and discuss possible techniques for addressing them. Section 6 provides details of our sample implementation, built using the techniques discussed in Sections 4 and 5. Finally, we present our conclusions in Section 7.

2 Related Work

Many commercially available network devices support inverse multiplexing using special hardware at the sender and receiver. The BONDING consortium [6] specifies techniques for packet striping on 56 and 64kbps circuit switched channels. More general inverse multiplexing schemes, as surveyed in [7], often fail to provide fair bandwidth allocation in the presence of variable size packets, or introduce significant reordering. The *strIPe* protocol [1] addresses these shortcomings, providing a general mechanism for fair bandwidth allocation with limited packet reordering, but does not adapt to changing channel capacity.

Previous work has shown Wireless TCP throughput to be sub-optimal due to interactions between the link layer and transport layer protocols. Asymmetry in media access, as found in CDPD up-channels, causes ACK-compression, and large delay variations due to a reliable link layer mechanism conspire to confound the TCP congestion control mechanisms [5]. While many of these problems have previously been studied and understood, most solutions assume a *basestation* paradigm, where agents are inserted at the interface between the wired and wireless links [3, 4]. In a commercial CDPD environment, however, users do not have access to the Mobile Data Base Stations (MDBS), and therefore cannot completely control the buffering, queuing, and retransmission mechanisms being used.

Inverse multiplexing restores the basestation view, as one can consider the entire path from multiplexor to demultiplexor to be one logical link. Traffic shaping and other known techniques can (and should) be employed, considering the multiplexor as the basestation. In particular, we believe *split-connection* methods similar to I-TCP [2] may be especially well-suited; there is little additional benefit to be gained from a transparent agent in this case, since network connectivity is lost in the event of a multiplexor failure. By terminating the connection at both ends, attention can be focused exclusively on the WWAN portion of the overall path, perhaps utilizing novel transport protocols specially designed or tuned for WWANs [9, 11, 14, RFC2488]. Alternatively, mechanisms such as Snoop agents [4] could be used if a transparent solution was desired in some particular environment.

Rather than cloud our results with additional factors by considering various proposals for transport protocols, we instead use the performance of standard TCP as a benchmark for our multi-link implementation. Performance gains achieved through the use of different transport protocols over individual WWAN links should transfer directly to our multi-link environment.

3 Assumptions

We assume each of the CDPD links in the bundle traverses the same wired path. This allows us to remove congestion control from each individual link and deal with the end-to-end congestion in totality (see Section 4.2.3). This assumption is flawed; each CDPD modem in the bundle may be operating in a separate cell[1], so the paths may diverge at some point. Close examination of the topology has shown, however, that this point is well within the

[1] Bell Atlantic (and, to our knowledge, most present CDPD carriers) deployed its CDPD network with one dedicated data channel per cell. It is therefore necessary for each modem to operate in a separate cell to realize any increase in available bandwidth.

CDPD network itself, and therefore insulated from non-CDPD cross traffic.

We further assume that the CDPD modems remain stationary. This is an implementation issue caused by the standard CDPD channel acquisition algorithm, which causes each modem to select the "best" channel. Clearly when they are co-located, roaming causes the modems to converge to the same channel, competing with each other for the same bandwidth. We are currently working to modify the channel selection algorithm to support mobility.

4 Multi-Link Characteristics

The method used for inverse multiplexing IP packets is standard. We stripe each packet across some number (possibly only one) of outgoing links, encapsulating it with a PPP Multi-Link (ML) header [RFC1990]. The resulting *fragments* are then sent using a transport mechanism over IP to the demultiplexor, where they are reassembled. The reconstructed packet is then forwarded to the appropriate destination.

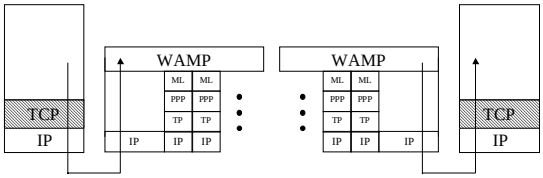

Figure 1: The WAMP Architecture

As can be seen in Figure 1, the end-to-end TCP packets are fragmented by the multiplexor and tunneled through multiple CDPD links using ML PPP over a link layer transfer protocol (TP). An inverse multiplexing strategy of this type has two main components: a fragmentation and scheduling mechanism and a transport protocol. In this section, we examine the possible choices for each, and discuss how the various alternatives affect reliable transport performance.

4.1 Scheduling Techniques

Once packets have been fragmented, the multiplexor must decide how to assign fragments to the available links: that is when and in what order to transmit them. This problem is more commonly studied in the inverse, when multiple links are multiplexed across a single, shared resource. As noted in [1], however, the problem maps directly to our situation. The obvious Round Robin approach allocates fragments amongst all the available links in an ordered fashion. In the long run, Round Robin scheduling provides a perfectly fair distribution of fragments. When these fragments are of roughly the same size, this translates into a balanced spread of bandwidth.

An entire class of scheduling mechanisms attempts to compensate for Round Robin's deficiencies in scheduling packets of varying sizes. Various fair queuing strategies select the appropriate link for the next fragment based upon the fragment's size and the queue lengths at the outgoing interfaces (see, for example, [8]). Unfortunately, many techniques of this flavor require accurate queue length information, which is difficult to obtain in a CDPD network. Furthermore, such techniques often lead to significant packet reordering when used with variable length packets.

A crucial property of WWANs, and CDPD networks in particular, is that while all of the links may be physically identical, their performance at any point in time is highly variable. Since each modem is operating in a separate cell, with possibly widely differing signal characteristics, some variation in channel throughput may be experienced even in the absence of channel contention. Furthermore, if additional CDPD modems are operating on some of the same channels, the link bandwidth available to our modems is reduced.

We propose a novel scheduling technique, similar to Weighted Round Robin, based on the ratio of short-term averages of observed throughput for each of the member links in a bundle. *Link Quality Balancing* dynamically adjusts the MTU of each link in proportion to the available bandwidth. By splitting packets into fragments that can be transmitted in roughly the same amount of time by each link, reassembly can proceed without delay.

In order to ensure traffic is actually distributed as intended, the fragmentation algorithm fragments each packet to ensure that the entire MTU of the current link is utilized before moving to the next link. If the last fragment of the previous packet did not fully utilize the relative MTU of the current link, the first fragment of the new packet is sized to fill the MTU, and scheduled accordingly. Note the two fragments are actually sent separately, so transmission of the previous packet is not delayed. In the case of uniformly sized packets, whose length is a multiple of the link MTU, this scheduling discipline reduces to Round Robin for bundles of identically performing links.

4.2 Link Layer Transport

Unlike many traditional multi-link applications, we do not have exclusive access to the physical links in question. Instead, we are forced to tunnel data over the Internet to the CDPD network, where it is sent across the cellular network to the modems. Empirical evidence shows the selection of an appropriate transport mechanism is critical to achieving acceptable performance. The fundamental design decision is whether to provide reliable transport and/or congestion control.

4.2.1 Reliability

This can be done through a reliable PPP connection, perhaps built using Numbered Mode [RFC1663], or by tunneling PPP over a reliable transport protocol. While arguments can be made for either choice, we claim an unreliable mechanism is superior in the multi-link environment, as it allows us to dispense with link layer congestion control, and reduces jitter in packet reassembly.

If a reliable transport mechanism is selected for each link, it requires a congestion control/avoidance scheme, since retransmitted fragments will travel over the shared Internet for a portion of their journey. Previous work has shown introducing an additional retransmission layer below TCP degrades overall performance [3].

4.2.2 Cached Retransmission

In some cases, however, the path between multiplexor and demultiplexor may be sufficiently lossy as to benefit from a Snoop-like [4] scheme to realize the savings of link-layer retransmission in the absence of a reliable transport mechanism. In such cases, PPP Numbered Mode [RFC1663] can be used. The multiplexor already tags each fragment with a Multi-Link (ML) identifier for reassembly. We propose to extend it with a hash of the flow ID and TCP sequence number, allowing the demultiplexor to associate it with a particular TCP packet even if reassembly fails. Then, if the end-to-end TCP requests a packet retransmission, the demultiplexor can identify exactly those fragments that are necessary to complete reassembly, and requests only their retransmission by referencing the Numbered Mode ID. We term this a *Lost Fragment Request*, or LFR.

For its part, the multiplexor simply caches each fragment. It snoops on the return path, observing ACK sequence numbers. Since TCP only transmits a *window*'s worth of data unacknowledged, the amount of data that needs to be sent before a cached fragment can be verified to have been safely received is bounded and small. Recall the *bandwidth*delay* product of CDPD links is itself small—on the order of two or three packets. Further study of the performance impact of LFRs is ongoing work; the results reported here do not reflect the benefit of fragment caching.

4.2.3 Congestion Control

If retransmissions are triggered only by the end-to-end transport protocol (either in the absence of a reliable link level, or using LFRs as described above), congestion control is unnecessary at the link level, and is effectively dealt with by end-to-end transport protocols.

Under our single path assumption, any congested bottleneck (except the final cell-specific MDBS) encountered by one link is also traversed by the other links. When one link receives notice of congestion (through packet loss), the appropriate response is for all of the links to respond by reducing their rate. Since loss is exposed to the upper level transport protocol, the logical link is determined to be congested, and the transport layer's estimate of the available bandwidth is adjusted appropriately. This adjustment affects all of the links equally (by equally, we mean in proportion to their measured throughput as a function of our scheduling policy).

Care must be taken, however, to ensure one lossy link does not limit the throughput of the bundle. In the case of transient Internet congestion, the individual links drop packets with equal probability. If congestion is occurring in a particular cell, packet losses will be concentrated on the associated link. While the individual fragment losses will indeed cause the end-to-end transport layer to slow, the demultiplexor records the relative loss rates of each link. It exchanges this information periodically with the multiplexor, allowing the scheduler to reduce its usage of each link in proportion to the estimated available channel throughput. As the end-to-end transport protocol continues to probe for bandwidth, the logical link places less demand on the physical link that previously failed, which leads to stable behavior.

4.3 Link Quality Metric

The effectiveness of such an adaptive technique hinges on the selection of a reliable metric for available link throughput. One approach might be to monitor the queue lengths at the outgoing interface, similar to weighted or fair queuing schemes described previously. While effective for the mobile multiplexor (which is directly connected to the CDPD modems), this method is fundamentally flawed if applied at the wired multiplexor. Recall that packets must first traverse some section of the Internet before entering the CDPD network. In this work, we assume the path to every CDPD modem is the same, up until some point well within the CDPD network (where it must diverge to reach separate cells). Therefore any inter-channel variability is only evident in queues far into the network, and would not be exposed at the local interface.

Instead, we passively monitor each link's performance using an extension to PPP's Link Quality Monitoring (LQM) standard [RFC1989]. By exchanging information about the loss rates and perceived throughput of each link, both multiplexors are able to make an informed and independent evaluation of channel performance (due to MAC contention on the cellular up-link and obvious discrepancies in transmitter power between the modem and tower, performance may be markedly different in opposite directions).

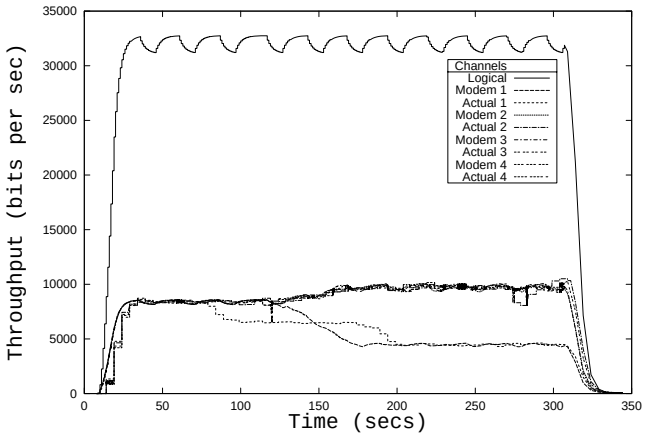

Figure 2: Link Quality Balancing across four CDPD modems

The obvious metric, as in traditional wired networks, is loss rate. Figure 2 shows an approximate Constant Bit Rate (CBR) source transmitting across the WAMP logical link. Initially the traffic is split equally across all four available CDPD links. At approximately T+75s, a fifth modem begins transmitting at a CBR of 6000bps on the same channel as modem 1. Almost immediately the throughput can be seen to drop at the demultiplexor. Due to the deep network buffers, however, no disproportional loss is perceived until T+125s, when excessive fragment loss on modem 1 causes the multiplexor to multiplicatively decrease its utilization of the link by some specified amount, δ.

The decrease in utilization of link 1 causes a proportional increase in the use of links 2-4, since the offered load is steady. The net effect, then, is a multiplicative decrease whose magnitude depends the utilization of the remaining links. WAMP continues to decrease its usage of the lossy link until the error rate returns to within an acceptable threshold, at which point utilization stabilizes.

At T+325s the CBR source terminates, and bandwidth utilization on all links drops off proportionately as traffic decreases. Note that even if the competing CBR source is removed, relative utilization of link 1 may not increase. If the remaining links are not saturated (which, in fact, they are not), and error rates remain tolerable, WAMP has no incentive to continually probe for additional bandwidth on link 1. Only when additional load arrives on the logical link (whether through a new connection or bandwidth probing of the end-to-end transport protocol), which over-saturates the remaining links, does WAMP increase utilization of link 1.

Note that WAMP never actually explicitly increases utilization of a link. Instead, as it decreases utilization of congested links, reliance on alternate links increases proportionately. This allows the end-to-end congestion control algorithms to operate as designed, without additional loss caused by bandwidth probing. While WAMP could generate synthetic traffic with which to probe links, CDPD networks do not support priority queuing, so packet loss is uniform. Hence any additional traffic, synthetic or not, may cause losses in the real traffic.

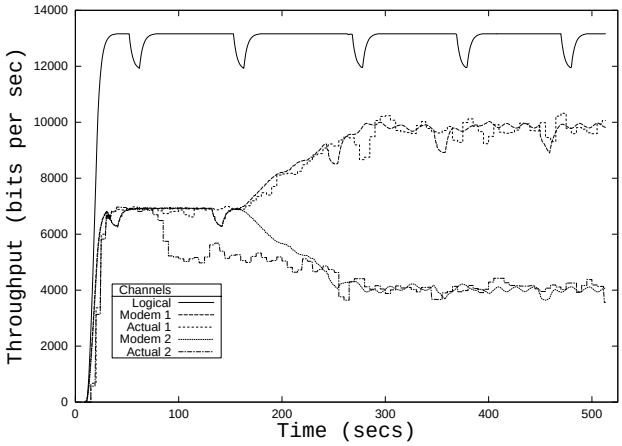

Figure 3: Link Quality Balancing across two CDPD modems

Figure 3 shows a similar experiment using only two links, which produces a much more dramatic effect. In this scenario, the CBR source provides an offered load of 13000bps. At T+75s a third modem begins to generate cross traffic on the same channel as modem 1, using a CBR stream of 5000bps. The decrease in available bandwidth becomes noticeable almost immediately, with the first loss event occurring at T+175s. WAMP then begins to decrease the utilization as before, leading to stable long-term behavior.

5 TCP Pathologies

We now turn our attention to the performance of a reliable transport protocol, TCP in particular, over our Multi-Link channel. This section identifies the difficulties in adapting link utilization for TCP flows, and suggests methods for improvement

5.1 Deep Buffering

TCP adapts to drastic changes in bandwidth by detecting losses. The assumption is that in the face of sudden changes in available bandwidth, packets will queue up and be dropped. Unfortunately, in a CDPD environment with few flows, the buffers are very deep compared to the receive windows. So sudden changes in bandwidth are often not detected by losses, but instead simply space out the ACKs from the receiver, which slows TCP's transmission rate down to the speed of the ACKs.

While TCP's bandwidth probing algorithm will continue to increase the window size one packet per (its estimation of the) RTT (which, as discussed in [3] and [5], is likely to be much larger than accurate), the decreased rate of ACKs will cause this to take longer than it should. Furthermore, the next loss event will occur with a smaller window (since TCP's congestion avoidance algorithm converges), so the network buffers are even more likely to be able to absorb the excessive traffic for long enough to slow the ACK clock down.

Figure 4 shows a single TCP sender operating over a WAMP link with two separate channels. The characteristic saw-tooth pattern of TCP's window probing algorithm is readily apparent. As before, at time T+125s, an additional modem began sending a CBR flow at 5000 bps on the same cellular channel as modem 2. The drop in available bandwidth is noticeable almost immediately, as a loss event occurs, and the saw-tooth turns downward. Finally, at T+175s the loss rates become disproportionate, and the multiplexor begins to decrease its utilization of link 2. Notice this occurs two probing periods later, after TCP has markedly decreased its sending rate due to the slower rate of returning ACKs.

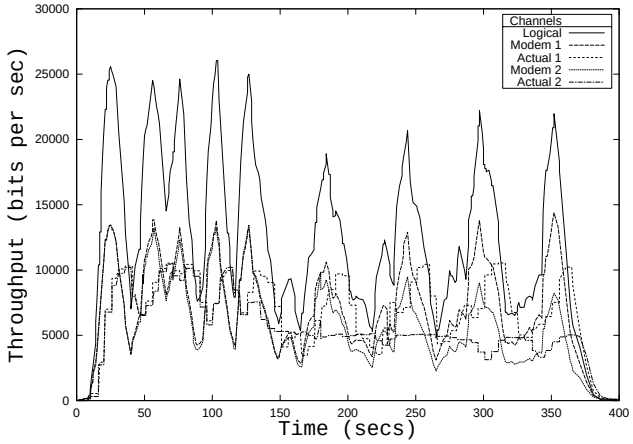

Figure 4: Link Quality Balancing across two links for a single TCP sender.

As WAMP continues to decrease utilization of the congested link, TCP can be seen to succeed in trying larger and larger window sizes. While this process will eventually converge, due to the stability properties of TCP's linear increase, multiplicative decrease congestion avoidance, it occurs over a very large time span. The transfer concludes at T+375s with an overall throughput approximately 85% of the theoretical maximum (if WAMP had adjusted instantaneously to the available channel bandwidth).

While this effect disappears rapidly as additional flows are established (since they are each asymmetrically probing for additional bandwidth), it is reasonable to assume the number of simultaneous flows will be small, given the low link bandwidth. This motivates our search for an alternate metric that could enable rapid adaptation to changes in available link throughput, without waiting for the network queues to admit a loss event.

Even if such a metric existed, considerable control issues arise when one attempts to make sudden, significant changes. In order to avoid the TCP problem, the decrease would have to be significant. Clearly false triggers would be catastrophic. Furthermore, care would need to be taken to ensure sufficient time for stabilization before additional adaptations were made. Relative short-term throughput averages, inter-packet separation averages, and inter-packet deviations have all initially proved too unstable for use. We are continuing to investigate dampening methods that might allow their use.

5.2 Packet Reordering

Depending on the scheduling policy in use, the top level TCP stack often may receive packets out of order due to the relative delay of individual channels. Our study of individual CDPD links shows that reordering frequency is similar to that expected in the general Internet [12], but *very rarely* by more than one packet.

If, as a general rule, we assume packets will be reordered at most once on each link, we can then examine the effect of striping fragments. Some scheduling policies then admit bounds on reordering. For our modified Round Robin techniques, reordering on the logical link is clearly just a function of packet size versus the number of links.

5.2.1 Fast Retransmit Modifications

TCP's fast retransmit algorithm exists to trigger rapid retransmission of lost packets. As specified, the receipt of three or more out of order packets signals packet loss, causing immediate retransmission. The value of three DUPACKs was arrived at empirically, and seems to function well for the wired Internet [12].

In our case, however, we can use any bounds provided by our scheduling mechanism to modify the retransmit algorithm. If, for instance, we are guaranteed never (actually *with high probability*, based on our assumption the individual links only reorder by one packet) to receive more than one out of order packet, we could reduce the number of DUPACKs to two, thereby causing a faster retransmission, improving throughput. If, on the other hand, our scheduling algorithm were such that packets were regularly delivered more than two packets out of order, the standard fast retransmit algorithm would incorrectly assume packet loss and cause spurious

retransmissions. In this case, it is clearly beneficial to adjust the number of DUPACKs up to the higher value.

Given the large timeout delays caused by not triggering the fast retransmit algorithm, it may be desirable to be a bit more aggressive in setting the DUPACK level than one would be on a wired network. For the purposes of this paper, however, we choose not to modify the algorithm to preserve standard TCP semantics.

5.3 Loss Amplification

The major drawback of multi-link operation is that all of the characteristics of the underlying channel, both positive and negative, are amplified. Take the case of packet loss for example. The probability of successful packet transmission for the logical link, p_t, can be expressed in terms of the loss probabilities for each of the n constituent links, p_i: $p_t = 1 - (1 - p_1)(1- p_2)\ldots(1- p_n)$. Splitting a packet across four links with 10% packet loss probability results in an incomplete packet 35% of the time. This lowest common denominator effect becomes increasingly evident when conditions deteriorate.

This effect implies there exists a point at which a link's membership in the bundle detracts from the overall bundle's performance. Consider, for example, the case of three links operating optimally, with one link dropping packets at a rate of 50% (not unheard of during peak hours in our CDPD network). Regardless of the retransmission mechanism used on the lossy link, the added delay caused by the required retransmission of lost data fragments causes TCP throughput on the logical link to be lower than it would be if it just used the three perfect links.

We are already monitoring link quality for use in our scheduling mechanism, so the data is available. The difficult part is determining at exactly what point to remove a link. There are obviously many factors, including the relative performance of a link to the others in the bundle, the absolute performance of the link, and so on. It is not clear how to calculate its exact value on-line. Instead, we suggest empirically computing a reasonable default threshold, and basing a link's membership in the bundle on that specified value. There is no danger in setting the value too low, as the resulting performance is no worse than the unmodified system. Care needs to be taken, however, not to set the value too high. Determining the optimum value for this threshold is ongoing work.

6 Implementation

The inverse multiplexor described herein was deployed using Sierra Wireless MP200 CDPD modems. The modems are connected via a Specialix SIO serial board to a PII/400 running FreeBSD 2.2.7. The other end of the virtual link is a Pentium-class machine running FreeBSD 3.2. This machine connects directly to the Bell Atlantic CDPD network over a 56K PVC Frame-Relay circuit.

WAMP is implemented as a user-level PPP daemon that runs PPP [RFC1661] with HDLC framing [RFC1662] over UDP. The CDPD modems operate on Bell Atlantic Mobile's digital cellular network. During initialization, the mobile multiplexor scans for available CDPD channels, and allocates the four best channels to the MP200s. Each modem locks on its prescribed channel, and begins PPP link establishment negotiation with the wired multiplexor/demultiplexor.

6.1 Tuning

Several aspects of our inverse multiplexing algorithm require constants to be set at appropriate values for the current operating environment.

6.1.1 LQM Interval

The current WAMP implementation exchanges LQM information every 5 seconds. Each LQM packet is 56 bytes, so this translates to an overhead of ~10 bytes/sec per link, on the order of 1% of the channel capacity. Shorter intervals not only increase overhead, but also provide less stability in the scheduling mechanism. Internet traffic is by nature bursty. When one considers the typical user's traffic on the CDPD network (mail clients, web browsers, and the like), it can be expected to be especially bursty. We clearly don't want to be too quick to predict a link's performance.

Given current CDPD speeds, one 576-byte packet (our MTU) requires on the order of half a second, five seconds allows for approximately 10 packets to be transmitted between LQM exchanges. Loss rates become less useful at lower intervals, as the granularity becomes too coarse. Furthermore, when multiple flows are active, we are likely to have significant buffering, so any changes will take several seconds to propagate through the channel anyway.

6.1.2 Loss Threshold & Decrease Delta

Loss thresholds and decrease deltas must be tightly correlated if stability is to be achieved. The higher the loss rate threshold, the slower WAMP adapts to long-term changes in available throughput. At the same time, it will be more resilient to transient bursts. Due to the asymmetric nature of CDPD's channel access, different values are required for each direction. For the down channel, where there is no media contention, empirical evidence suggests that during off hours, most bursts can be accommodated in the network and modem buffers, and any loss exposed to the link layer is significant. For the up channel, on the other hand, losses are much more common, due to the CSMA/CD channel access method.

At present, we manually adjust the loss threshold to current operating conditions, but are investigating methods for separating load-induced loss with load-independent loss. We have found thresholds of 2% during evening hours and 5% during peak hours work reasonably well.

Optimum values of the decrease parameter, δ, however, have proved to be considerably more stable over long time frames. A decrease factor of .85 is the largest factor that leads to stable behavior with reasonable reaction times. While this is a considerably smaller decrease than found in TCP Reno, one must recall that scheduling is relative. Hence any decrease in one link leads to an increase in the others. Small values of δ lead to huge oscillations in link scheduling, especially when few links are involved, as well-behaved links are driven into overload by the increased load caused by the sudden decrease in utilization of a lossy link.

7 Conclusions

We have presented an adaptive approach to inverse multiplexing reliable transport protocols in WWAN environments based upon three key observations. First, when the component links share a path to the CDPD network, we note that congestion control is appropriately and effectively handled by the upper level transport protocol, and is impeded by additional control at the link layer. This does not, however, imply the individual links could not provide some form of enhanced reliability, as proposed by our LFR scheme.

Secondly, we argue that optimum fragment scheduling requires knowledge of the current channel characteristics of each link. By adapting fragment size to the current effective throughput of each link, we enable packet reassembly and delivery with a minimum of delay, thereby preventing slow links from throttling the performance of the entire bundle.

Finally, while obtaining accurate measurements of instantaneous channel transmission delay is problematic, we note that relative channel performance is sufficient, since the transport protocol's congestion control algorithm will appropriately increase or decrease bandwidth utilization. LQM loss data sent from the receiver at regular intervals is sufficient to maintain a stable short-term approximation of relative throughput. Determining optimum sampling intervals is ongoing work.

8 Acknowledgements

Rusty Hemenway and his colleagues at Bell Atlantic Mobile provided invaluable assistance in setting up the experimental testbed. Ty Sealy and Ron Wiken aided in the installation of the mobile equipment. We are indebted to John Wroclawski and Hari Balakrishnan for their constructive criticism and willingness to support this work.

9 References

[1] H. ADISESHU, G. PARULKAR, AND G. VARGHESE. A Reliable and Scalable Striping Protocol. In *Proc.of ACM SIGCOMM*, August 1996

[2] A. BAKRE AND B. BADRINATH. I-TCP: Indirect TCP for Mobile Hosts. In *Proc. of ICDCS*, May 1995.

[3] B. BAKSHI, P. KRISHNA, N. VAIDYA, AND D. K. PRADHAN. Improving Performance of TCP over Wireless Networks. In *Proc. of ICDCS*, May 1997.

[4] H. BALAKRISHNAN, S. SESHAN, E. AMIR, AND R. KATZ. Improving TCP/IP Performance over Wireless Networks. In *Proc. of ACM MOBICOM*, November 1995.

[5] H. BALAKRISHNAN, V. PADMANABHAN, AND R. KATZ. The Effects of Asymmetry on TCP Performance. In *Proc. of ACM/IEEE MOBICOM*, September 1997.

[6] Bandwidth ON Demand Interoperability Group. Interoperability Requirements for Nx56/64 kbit/s Calls, September 1992.

[7] C. BRENDAN, S. TRAW, AND J. SMITH. Striping within the Network Subsystem. *IEEE Network*, 1995.

[8] A. DEMERS, S. KESHAV, AND S. SHENKER. Analysis of a Fair Queuing Algorithm, *Journal of Internetworking Research and Experience*, September 1989.

[9] R. DURST, G. MILLER, AND E. TRAVIS. TCP Extensions for Space Communications. In *Proc. of ACM/IEEE MOBICOM*, September 1996.

[10] Electronic Industry Alliance/Telecommunications Industry Association. IS-95: Mobile Station-Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular System, 1993.

[11] M. MEHTA AND N. VAIDYA. Delayed Duplicate-Acknowledgements: A Proposal to Improve Performance of TCP on Wireless Links. Technical Report, Computer Science Dept., Texas A&M University, February 1999.

[12] V. PAXSON. End-to-End Internet Packet Dynamics. In *Proc. of ACM SIGCOMM*, September 1997.

[13] M. RAHNEMA. An Overview of the GSM System and Protocol Architecture. *IEEE Communications Magazine*: 31, April 1993.

[14] P. SINHA, N. VENKITARAMAN, R. SIVAKUMAR, AND V. BHARGHAVAN. WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks. In *Proc. of ACM/IEEE MOBICOM*, August 1999.

[15] Wireless Data Forum. Cellular Digital Packet Data System Specification, Release 1.1, January 1995.