Nick Feamster

Assistant Professor

School of Computer Science
Klaus Advanced Computing Building
Georgia Institute of Technology
266 Ferst Drive, Room 3348
Atlanta, GA 30332-0765, USA
+1 404 385 1944
feamster@cc.gatech.edu

Table of Contents

EDUCATIONAL BACKGROUND

Degree	Year	University	Field
Ph.D.	2005	Massachusetts Institute of Technology Cambridge, MA *Dissertation:* Proactive Techniques for Correct and Predictable Internet Routing *Sprowls Honorable Mention for best MIT Ph.D. dissertation in computer science* *Advisor:* Hari Balakrishnan Minor in Game Theory	Computer Science
M.Eng.	2001	Massachusetts Institute of Technology Cambridge, MA *Dissertation:* Adaptive Delivery of Real-Time Streaming Video *Advisor:* Hari Balakrishnan *William A. Martin Memorial Thesis Award* (MIT M.Eng. thesis award)	Computer Science
S.B.	2000	Massachusetts Institute of Technology Cambridge, MA	Electrical Engineering and Computer Science

EMPLOYMENT HISTORY

Title	Organization	Years
Assistant Professor	Georgia Institute of Technology	*2006–Present*
Postdoctoral Research Staff	Princeton University	*Fall 2005*
Research Assistant	Massachusetts Institute of Technology	*2000–2005*
Intern/Consultant	AT&T Labs–Research	*2001–2005*
Technical Associate	Bell Laboratories	*1999*
Intern	Hewlett-Packard Laboratories	*1999*
Technical Staff	LookSmart, Ltd.	*1997*

CURRENT FIELDS OF INTEREST

My research focuses on networked computer systems, with a strong emphasis on network architecture and protocol design; network security, management and measurement; routing; and anti-censorship techniques. The primary goal of my research is to help network operators run their networks better, and to enable users of these networks (both public and private) to experience high availability and good end-to-end performance. I have a strong interest in tackling practical problems using a "first principles" approach, designing systems based on these principles, and implementing and deploying these systems in practice.

I. TEACHING

A. Courses Taught

Term	Year	Number of Course Number & Title	Students	Comments
Spring	2010	CS 3251 Computing Networking I	53	
Spring	2010	CS 8803 NGN Next Generation Networking	50	New Course
Fall	2009	CS 7001 Introduction to Graduate Studies	39	
Spring	2009	CS 6262 Network Security	45	Updated Syllabus
Fall	2008	CS 4251 Computer Networking II	16	
Fall	2008	CS 7001 Introduction to Graduate Studies	44	
Spring	2008	CS 4251 Computer Networking II	14	New Syllabus
Fall	2007	CS 7001 Introduction to Graduate Studies	53	
Spring	2007	CS 7260 Internetworking Protocols and Architectures	29	
Fall	2006	CS 7001 Introduction to Graduate Studies	74	New Syllabus
Fall	2006	CS 8001 Networking Research Seminar	30	New Syllabus
Fall	2006	CS 1100 Freshman Leap Seminar	15	
Spring	2006	CS 7260 Internetworking Protocols and Architectures	27	New Syllabus

Tutorials on BGP Multiplexer at GENI Experimenters Workshop and
GENI Engineering Conference in Summer 2010.
Tutorial on network security at African Network Operators Group (AfNOG) in Summer 2009. Tutorial on internet routing at Simposio Brasileiro de Redes de Computadores (SBRC) in Summer 2008.
Lecture for DIMACS Tutorial on Next-Generation Internet Routing Algorithms in August 2007.
Guest lecture for CS 6250 (Advanced Computer Networks) in Fall 2007.
Guest lecture for CS 3251 (Computer Networks I) in Fall 2006.
Multiple guest lectures for CS 4251 (Computer Networks II) in Spring 2006.
Guest lecture for MIT Course 6.829 (Computer Networking) in Fall 2005.

B. Curriculum Development

CS 8803 Next-Generation Networking: I have developed a brand-new graduate course that gives students practical experience with a variety of tools for next-generation networking, ranging from the Click software router to the OpenFlow switch framework. The course also teaches students about the state of the art in networking research—students read papers about research and industry trends and do a course project that incorporates aspects of these new technologies. This course relates to the larger nationwide effort on Global Environment for Network Innovations (GENI), which is building infrastructure for researchers to provide the next generation of networking protocols and technologies.

CS 7001 Introduction to Graduate Studies: With Professor Alex Gray, I have developed a new course syllabus and structure to CS 7001 around the larger goal of introducing new students to *how to do great research* as soon as their first term at Georgia Tech. In contrast with previous terms, where CS 7001 consisted of faculty "advertisements" for their research and projects consisted of short "mini-projects" where little research could be accomplished in a short time span of 3 weeks, we have improved the syllabus by bringing in faculty members to talk about research philosophy, exciting new directions, etc. We have also given the students the option to do a research project that is a term-long project in conjunction with CS 8903; our goal is to give students the flexilibty to select meaningful research problems based on their research assistantships while helping them learn the skills required for writing papers, finding and evaluating research ideas, and performing other tasks associated with doing great research. Alex Gray and I wrote a conference paper on our development of this course, which appeared at *ACM SIGCSE 2008*.

College of Computing Research Day and Seminar Series: In addition to the course itself, to fulfill some of the functions of the former 7001 course, Alex Gray and I financed and organized a college-wide seminar series and research day in Fall 2009. Throughout the term, faculty speakers from across the college gave one-hour talks about their research; we raised money from Yahoo to support this event. The research day brings together students and faculty from around the college to see talks, demonstrations, and posters from around the collage to exchange ideas.

CS 4251 Computer Networking II: *Spring/Fall 2008.* I developed new hands-on assignments to give students experience with real-world networking tools and software (*e.g.*, Emulab, Quagga, Click). I also revamped the course around various high-level themese in networking, including layering, resource sharing, tree formation (routing and forwarding, etc.). Finally, I developed over 20 new lectures for the syllabus, as well as new problem sets which can be re-used for future offerings of the course.

CS 6262 Network Security: *Spring 2009.* I updated the syllabus to include recent network attacks (*e.g.*, spam, botnets, reflection attacks, etc.) and also to integrate more hands-on assignments. Updating problem sets and project lists.

CS 7260 Internetworking Architectures and Protocols: *Spring 2006.* I developed a new project-based graduate course with substantial programming assignments using a wide variety of state-of-the-art networking software tools and platforms (*e.g.*, rcc, PlanetLab, scriptroute, NetFlow, etc.). I contributed questions to a larger bank of questions also used in graduate-level networking courses at Carnegie Mellon and MIT. Finally, I developed 24 new lectures, many based on current "hot topics" in computer networking (*e.g.*, spam, botnets, traffic anomaly detection, etc.)

Spring 2007. I developed two new course modules: (1) sound techniques for network measurement; and (2) evaluation platforms (Emulab, VINI, etc.). I designed new problem sets on these topics. With faculty at Carnegie Mellon, I instituted the use of a cross-institutional online forum for paper discussion. Students read papers from the CS 7260 syllabus and commented on papers before class to help stimulate paper discussion; students also read the discussion blog and could comment on papers being discussed in networking classes at other schools.

C. Individual Student Guidance

C.1. Postdocs Supervised

Cristian Lumezanu College of Computing
Fall 2009 - Present
Research on Internet measurement and economics.

C.2. Ph.D. Students Supervised

Hyojoon Kim College of Computing
Fall 2009 - Present
Research on prgrammable networks and network configuration.

Robert Lychev College of Computing
Fall 2008 - Present
Research on contract enforcement for transit markets.
Taken qualifier (grading in progress).

Sam Burnett College of Computing
Fall 2008 - Present
Publications: *D.0.2, F.5.2, F.5.7*
Design and implementation of anti-censorship systems.
Taken qualifier (grading in progress).

Srikanth Sundaresan College of Computing
Fall 2008 - Present
Publications: *F.5.3*
Research on access network performance and online traffic engineering.
Taken qualifier (grading in progress).

Bilal Anwer College of Computing
Fall 2008 - Present
Publications: *D.0.1, E.0.1, E.0.4*
Research on support for hardware forwarding in virtual network environments.
Taken qualifier (grading in progress).

Shuang Hao College of Computing
Fall 2007 - Present
Publications: *F.3.7, D.0.9*
Research on botnet detection, network monitoring, and spam filtering.
Taken qualifier (grading in progress).

Maria Konte College of Computing
Fall 2007 - Present
Publications: *D.0.10*
Measurement study of fast-flux networks.
Taken qualifier (grading in progress).

Yogesh Mundada College of Computing
Fall 2007 - Present
Publications: *F.3.8, F.5.9*
Development of experiment specification for VINI and integration of VINI with Emulab.
Taken qualifier (grading in progress).

Vytautas Valancius College of Computing
Summer 2007 - Present
Publications: *D.0.4 E.0.7, E.0.6 F.3.5, F.3.8, F.3.13, F.5.1 F.5.9, F.5.10*
Research on interdomain routing and network virtualization.
Passed qualifier.

Mukarram Bin Tariq College of Computing
Spring 2007 - Spring 2010
Publications: *D.0.13, D.0.7, D.0.6, E.0.8, F.1.1, F.3.6*
Co-advised with Mostafa Ammar. Research on statistical inference methods for network planning and troubleshooting problems. Mukarram's dissertation work is now part of operational systems at Google.
Graduated. Now works full-time at Google in the network monitoring group.

Murtaza Motiwala College of Computing
Fall 2006 - Present
Publications: *D.0.12, E.0.11, F.3.8, F.3.9, F.3.12, F.3.16, F.5.9*
Research on (1) in-band troubleshooting and (2) scalable network architectures for path diversity, including path splicing.
Passed qualifier.

Anirudh Ramachandran College of Computing
Spring 2006 - Present
Publications: *C.0.1, D.0.16, D.0.21, E.0.10, E.0.13, E.0.14, E.0.15, F.3.4, F.3.6, F.5.8, F.5.13*
Research on network-level behavior of spammers and passive botnet detection.
Passed qualifier and proposal defense. Expected graduation Fall 2010.

C.3. Masters Students Supervised

Ankur Nayak College of Computing
Spring 2009 - Spring 2010
Publications: *E.0.5*
Dynamic access control with programmable switches.

Umayr Hassan College of Computing
Fall 2008 - Spring 2010
Research on the design of a market for Internet transit, and on home network configuration.
Umayr now works full-time at Bloomberg.

Nadeem Syed College of Computing
Spring 2007 - Spring 2008
Publications: *F.5.11*
Co-advised with Alex Gray. Developing and implementing new machine learning techniques for fast disruption detection.
Nadeem is in the MBA program at Georgia Tech.

Kaushik Bhandakar College of Computing
Spring 2007 - Summer 2008
Publications: *F.5.8*
Experiments for VINI performance benchmarking; implementation and prototyping for the "Pedigree" packet provenance project; research on incentives in BitTorrent.
Kaushik now works full-time at Google.

Samantha Lo Hong Kong Polytechnic University
Spring 2007
Research on market-based network architectures and inbound traffic engineering.
Samantha is now a Ph.D. student at Georgia Tech.

Manas Khadilkar College of Computing
Fall 2006 - Spring 2007
Publications: *D.0.17*
Research on efficient settings of lease times for DHCP address allocation. Algorithm in development, to be used on the Georgia Tech campus network for optimizing lease time settings.
Manas now works full-time for Expedia.

Han Lu College of Computing
Fall 2006 - Spring 2007
Research on spam traffic patterns by IP address space.

Chris Kelly College of Computing
Fall 2006 - Fall 2007
Developing new software features for the Campus-Wide Performance Monitoring and Recovery (CPR) project.
Chris now works full-time for SugarCRM, an Atlanta-based startup.

Yiyi Huang College of Computing
Spring 2006 - Fall 2009
Publications: *D.0.18, F.3.10*
Co-advised with Jim Xu. Research on fast, distributed network anomaly detection.
Yiyi now works full-time at Microsoft.

Winston Wang M.I.T. EECS
Fall 2002 - Spring 2003
Publications: *E.0.19*
Thesis on an implementation of the Infranet anti-censorship system received MIT's Charles and Jennifer Johnson Thesis Prize.

C.4. Undergraduate Students Supervised

Alex Reimers College of Computing
Spring 2009
Publications: *E.0.5*
Worked on dynamic access control (a replacement of Georgia Tech OIT's current authentication system) with programmable switches. Alex now works full time at Cisco.

Megan Elmore College of Computing
Fall 2007 - Spring 2009
Publications: *D.0.12*
Experiments for interdomain path splicing; design and implementation of the path splicing prototype. Work received 2nd prize in 2008 Georgia Tech College of Computing undergraduate research competition. Megan was also the winner of the 2009 College of Computing Undergraduate Research Award, and the 2009 Sigma Xi Best Undergraduate Researcher Award. Megan is now a Ph.D. student at Stanford University.

Hongyi Hu M.I.T. EECS
Spring 2005 - Fall 2005
Extensions to the *rcc* router configuration checker tool for static configuration analysis of internal routing protocol configurations.

C.5. Special Projects

Mona Chitnis College of Computing
Spring 2010
Research on OpenFlow network architectures.

Sravanthi Gondhi College of Computing
Spring 2010
Research on online traffic engineering.

Shruti Gupta College of Computing
Spring 2010
Research on online traffic engineering.

Utkarsh Shrivastava College of Computing
Spring 2010
Research on network-level behavior of spammers.

Pooja Rajanna College of Computing
Spring 2010
Research on network-level behavior of spammers.

Luxmi Saha College of Computing
Spring 2010
Research on data-center scheduling algorithms.

Dongchan Kim College of Computing
Fall 2009
Research on spam filtering.

Sonali Batra College of Computing
Summer 2009
Research on anti-phishing techniques.

Radhika Partharathy College of Computing
Fall 2008
Research on anti-phishing techniques.

Sagar Mehta College of Computing
Fall 2006 - Spring 2008
Research on anti-phishing techniques.

Bhairav Dutia College of Computing
Fall 2006
Research on anti-censorhip techniques and countermeasures.

Megan Benoit College of Computing
Fall 2006
Research on spammers' email address harvesting practices.

Amit Khanna College of Computing
Fall 2006
Implemented Secure BGP (S-BGP) in the Quagga software router. Software publically available and operators are using the codebase for ongoing work on certificates for secure routing.

Daniel Mentz College of Computing
Spring 2006
Research on campus network security troubleshooting.

Buddy Moore College of Computing
Summer 2006
Implemented distributed version of the Infranet anti-censorship software. Publicly available.

II. RESEARCH AND CREATIVE SCHOLARSHIP

A. Theses

A.0.1 Nick Feamster. *Proactive Techniques for Correct and Predictable Internet Routing*. PhD thesis, Massachusetts Institute of Technology, February 2006. Winner of the MIT George M. Sprowls Honorable Mention for Best MIT Ph.D. Dissertation in Computer Science.

A.0.2 Nick Feamster. Adaptive delivery of real-time streaming video. Master's thesis, Massachusetts Institute of Technology, May 2001. Winner of the MIT EECS William A. Martin Memorial Thesis Award.

B. Journal Publications

B.0.1 Bilal Anwer and Nick Feamster. Building a fast, virtualized data plane with programmable hardware. *ACM SIGCOMM Computer Communication Review*, April 2010.

B.0.2 Nick Feamster, Ramesh Johari, and Hari Balakrishnan. Stable Policy Routing with Provider Independence. *IEEE/ACM Transactions on Networking*, December 2007.

B.0.3 Nick Feamster and Jennifer Rexford. Network-Wide Prediction of BGP Routes. *IEEE/ACM Transactions on Networking*, June 2007.

B.0.4 Nick Feamster, Jaeyeon Jung, and Hari Balakrishnan. An Empirical Study of "Bogon" Route Advertisements. *ACM Computer Communications Review*, 35(1):63–70, November 2004.

B.0.5 Nick Feamster, Jay Borkenhagen, and Jennifer Rexford. Guidelines for Interdomain Traffic Engineering. *ACM Computer Communications Review*, 33(5):19–30, October 2003.

C. Books and Book Chapters

C.0.1 Anirudh Ramachandran, Nick Feamster, and David Dagon. *Botnet Detection: Countering the Largest Security Threat*. Springer, 2008. *Chapter:* Revealing Botnet Membership with DNSBL Counterintelligence.

D. Refereed Conference Publications

D.0.1 Bilal Anwer, Murtaza Motiwala, Mukarram bin Tariq, and Nick Feamster. SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware. In *Proc. ACM SIGCOMM*, New Delhi, India, August 2010.
Acceptance rate: 12%

D.0.2 Sam Burnett, Nick Feamster, and Santosh Vempala. Chipping Away at Censorship Firewalls with Collage. In *Proc. 19th USENIX Security Symposium*, Washington, DC, August 2010.
Acceptance rate: 15%

D.0.3 Manos Antonakakis and Roberto Perdisci and David Dagon and Wenke Lee and Nick Feamster. Building a Dynamic Reputation System for DNS. In *Proc. 19th USENIX Security Symposium*, Washington, DC, August 2010.
Acceptance rate: 15%

D.0.4 Vytautas Valancius, Nick Feamster, Jennifer Rexford, and Akihiro Nakao. Wide-Area Routing for Distributed Services. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2010.
Acceptance rate: 17%

D.0.5 Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral Clustering of HTTP-Based Malware. In *Proc. 7th ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, April 2010.
Acceptance rate: 16%

D.0.6 Mohammed Mukarram bin Tariq, Murtaza Motiwala, Nick Feamster, and Mostafa Ammar. Detecting General Network Neutrality Violations with Causal Inference. In *4th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, Rome, Italy, December 2009.
Acceptance rate: 17%

D.0.7 Mohammed Mukarram bin Tariq, Ahmed Mansy, Nick Feamster, and Mostafa Ammar. Measuring VLAN-Induced Sharing on a Campus Network. In *Proc. ACM SIGCOMM Internet Measurement Conference*, Chicago, Illinois, October 2009.
Acceptance rate: 22%

D.0.8 Italo Cunha, Renata Teixeira, Nick Feamster, and Christophe Diot. Techniques for Fast and Accurate Network Tomography. In *Proc. ACM SIGCOMM Internet Measurement Conference*, Chicago, Illinois, October 2009.
Acceptance rate: 22%

D.0.9 Shuang Hao, Nadeem Syed, Nick Feamster, Alexander Gray, and Sven Krasser. Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine. In *Proc. 18th USENIX Security Symposium*, Montreal, Quebec, Canada, August 2009.
Acceptance rate: 15%

D.0.10 Maria Konte, Nick Feamster, and Jaeyeon Jung. Dynamics of Online Scam Infrastructure. In *Proc. Passive and Active Measurement Conference*, Seoul, Korea, March 2009.
Acceptance rate: 20% **Best paper award.**

D.0.11 Anirudh Ramachandran, Srinivasan Seetharaman, Nick Feamster, and Vijay Vazirani. Fast Monitoring of Traffic Subpopulations. In *Proc. ACM SIGCOMM Internet Measurement Conference*, Vouliagmeni, Greece, October 2008.
Acceptance rate: 17%

D.0.12 Murtaza Motiwala, Megan Elmore, Nick Feamster, and Santosh Vempala. Path Splicing. In *Proc. ACM SIGCOMM*, Seattle, WA, August 2008.
Acceptance rate: 12%

D.0.13 Mohammed Mukarram bin Tariq, Amgad Zeitoun, Nick Feamster, and Mostafa Ammar. Answering What-If Deployment and Configuration Questions with WISE. In *Proc. ACM SIGCOMM*, Seattle, WA, August 2008.
Acceptance rate: 12%

D.0.14 David Andersen, Hari Balakrishnan, Nick Feamster, and Scott Shenker. Accountable Internet Protocol (AIP). In *Proc. ACM SIGCOMM*, Seattle, WA, August 2008.
Acceptance rate: 12%

D.0.15 Nick Feamster and Alexander Gray. Can Great Research Be Taught? Independent Research with Cross-Disciplinary Thinking and Broader Impact. In *ACM SIGCSE Technical Symposium on Computer Science Eudation (SIGCSE)*, Portland, OR, March 2008.

D.0.16 Anirudh Ramachandran, Nick Feamster, and Santosh Vempala. Filtering Spam with Behavioral Blacklisting. In *Proc. 14th ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, October 2007.
Acceptance rate: 24%

D.0.17 Manas Khadilkar, Nick Feamster, Russ Clark, and Matt Sanders. Usage-Based DHCP Lease Time Optimization. In *Proc. ACM SIGCOMM Internet Measurement Conference*, San Diego, CA, October 2007.
Acceptance rate: 24%

D.0.18 Yiyi Huang, Nick Feamster, Anukool Lakhina, and Jim Xu. Exposing Routing Problems with Network-Wide analysis. In *Proc. ACM SIGMETRICS*, San Diego, CA, June 2007.
Acceptance rate: 17%

D.0.19 Feng Wang, Nick Feamster, and Lixin Gao. Measuring the contributions of routing dynamics to prolonged end-to-end internet path failures. In *Proc. IEEE Conference on Global Communications (GlobeCom)*, Washington, DC, November 2007.
Acceptance rate: 40%

D.0.20 Christopher P. Lee, Keshav Attrey, Carlos Caballero, Nick Feamster, Milena Mihail, and John A. Copeland. MobCast: Overlay Architecture for Seamless IP Mobility using Scalable Anycast Proxies. In *IEEE Wireless Communications and Networking Conference*, Hong Kong, March 2007.
Acceptance rate: 47%

D.0.21 Anirudh Ramachandran and Nick Feamster. Understanding the Network-Level Behavior of Spammers. In *Proc. ACM SIGCOMM*, Pisa, Italy, August 2006. An earlier version appeared as Georgia Tech TR GT-CSS-2006-001.
Acceptance rate: 12% **Best student paper award.**

D.0.22 Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In VINI Veritas: Realistic and controlled network experimentation. In *Proc. ACM SIGCOMM*, Pisa, Italy, August 2006.
Acceptance rate: 12%

D.0.23 Nick Feamster and Hari Balakrishnan. Correctness Properties for Internet Routing. In *Forty-third Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, September 2005.

D.0.24 Nick Feamster, Ramesh Johari, and Hari Balakrishnan. The Implications of Autonomy for Stable Policy Routing. In *Proc. ACM SIGCOMM*, pages 25–36, Philadelphia, PA, August 2005.
Acceptance rate: 11%

D.0.25 Michael Freedman, Mythili Vutukuru, Nick Feamster, and Hari Balakrishnan. Geographic Locality of IP Prefixes. In *Proc. ACM SIGCOMM Internet Measurement Conference*, New Orleans, LA, October 2005.
Acceptance rate: 24%

D.0.26 Nick Feamster and Hari Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 43–56, Boston, MA, May 2005.
Acceptance rate: 22% **Best paper award.**

D.0.27 Matthew Caesar, Don Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Kobus van der Merwe. Design and Implementation of a Routing Control Platform. In *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–28, Boston, MA, May 2005.
Acceptance rate: 22%

D.0.28 Nick Feamster, Zhuoqing Morley Mao, and Jennifer Rexford. BorderGuard: Detecting Cold Potatoes from Peers. In *Proc. ACM SIGCOMM Internet Measurement Conference*, pages 213–218, Taormina, Sicily, Italy, October 2004.
Acceptance rate: 25%

D.0.29 Nick Feamster, Jared Winick, and Jennifer Rexford. A Model of BGP Routing for Network Engineering. In *Proc. ACM SIGMETRICS*, pages 331–342, New York, NY, June 2004.
Acceptance rate: 12%

D.0.30 Nick Feamster, David Andersen, Hari Balakrishnan, and M. Frans Kaashoek. Measuring the Effects of Internet Path Faults on Reactive Routing. In *Proc. ACM SIGMETRICS*, pages 126–137, San Diego, CA, June 2003.
Acceptance rate: 12%

D.0.31 Nick Feamster, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan, and David Karger. Infranet: Circumventing Web censorship and surveillance. In *Proc. 11th USENIX Security Symposium*, San Francisco, CA, August 2002.
Acceptance rate: 17% **Best student paper award.**

D.0.32 Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. Dos and don'ts of client authentication on the Web. In *Proc. 10th USENIX Security Symposium*, Washington, DC, August 2001.
Acceptance rate: 28% **Best student paper award.**

D.0.33 Susie Wee, John Apostolopoulos, and Nick Feamster. Field-to-frame transcoding with temporal and spatial downsampling. In *IEEE International Conference on Image Processing*, October 1999.
Acceptance rate: 45%

D.0.34 Nick Feamster and Susie Wee. An MPEG-2 to H.263 transcoder. In *SPIE Voice, Video, and Data Communications Conference*, Boston, MA, September 1999.

E. Workshop Publications

E.0.1 Bilal Anwer, Ankur Nayak, Nick Feamster, and Ling Liu. Network I/O Fairness in Virtual Machines. In *ACM SIGCOMM Workshop on Virtualized Infrastrastructure, Services, and Architectures (VISA)*, New Delhi, India, September 2010.

E.0.2 Nick Feamster. Outsourcing Home Network Security. In *ACM SIGCOMM Workshop on Home Networking (HomeNets)*, New Delhi, India, September 2010.

E.0.3 Ken Calvert, W. Keith Edwards, Nick Feamster, Rebecca Grinter, Ye Deng, and Xuzi Zhou. Instrumenting Home Networks. In *ACM SIGCOMM Workshop on Home Networking (HomeNets)*, New Delhi, India, September 2010.

E.0.4 Bilal Anwer and Nick Feamster. Building a Fast, Virtualized Data Plane with Programmable Hardware. In *ACM SIGCOMM Workshop on Virtualized Infrastrastructure, Services, and Architectures (VISA)*, Barcelona, Spain, August 2009.

E.0.5 Ankur Nayak, Alex Reimers, Russ Clark, and Nick Feamster. Resonance: Dynamic Access Control for Enterprise Networks. In *ACM SIGCOMM Workshop on Research in Enterprise Networks (WREN)*, Barcelona, Spain, August 2009.

E.0.6 Sapan Bhatia, Murtaza Motiwala, Wolfgang Muhlbauer, Yogesh Mundada, Vytautas Valancius, Andy Bavior, Nick Feamster, Larry Peterson, and Jennifer Rexford. Trellis: A Platform for Building Flexible, Fast Virtual Networks on Commodity Hardware. In *3rd International Workshop on Real Overlays & Distributed Systems*, December 2008.

E.0.7 Vytautas Valancius, Nick Feamster, Ramesh Johari, and Vijay Vazirani. MINT: A Market for Internet Transit. In *ACM SIGCOMM CoNext Workshop on Re-Architecting the Internet*, December 2009.

E.0.8 Mohammed Mukarram bin Tariq, Murtaza Motiwala, and Nick Feamster. NANO: Network Access Neutrality Observatory. In *Proc. 7th ACM Workshop on Hot Topics in Networks (Hotnets-VII)*, Calgary, Alberta, Canada, October 2008.
Acceptance rate: 20%

E.0.9 S. Yardi, N. Feamster, and A. Bruckman. Photo-Based Authentication Using Social Networks. In *Proc. ACM SIGCOMM Workshop on Online Social Networks*, Seattle, WA, August 2008.

E.0.10 Anirudh Ramachandran and Nick Feamster. Authenticated Out-of-Band Communication over Social Links. In *Proc. ACM SIGCOMM Workshop on Online Social Networks*, Seattle, WA, August 2008.

E.0.11 Murtaza Motiwala, Nick Feamster, and Santosh Vempala. Path Splicing: Reliable Connectivity with Rapid Recovery. In *Proc. 6th ACM Workshop on Hot Topics in Networks (Hotnets-VI)*, Atlanta, GA, November 2007.
Acceptance rate: 18%

E.0.12 David G. Andersen, Hari Balakrishnan, Nick Feamster, and Scott Shenker. Holding the Internet Accountable. In *Proc. 6th ACM Workshop on Hot Topics in Networks (Hotnets-VI)*, Atlanta, GA, November 2007.
Acceptance rate: 18%

E.0.13 Anirudh Ramachandran, Atish das Sarma, and Nick Feamster. BitStore: An Incentive-Compatible Solution for Blocked Downloads in Bittorrent. In *ACM Joint Workshop on The Economics of Networked Systems and Incentive-Based Computing (NetEcon)*, San Diego, CA, June 2007.

E.0.14 Anirudh Ramachandran, Nick Feamster, and David Dagon. Revealing Botnet Membership with DNSBL Counter-Intelligence. In *2nd USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, San Jose, CA, July 2006.

E.0.15 Anirudh Ramachandran, David Dagon, and Nick Feamster. Can DNSBLs Keep Up with Bots? In *3rd Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, July 2006.

E.0.16 Nick Feamster, Hari Balakrishnan, and Jennifer Rexford. Some foundational problems in interdomain routing. In *Proc. 3nd ACM Workshop on Hot Topics in Networks (Hotnets-III)*, San Diego, CA, November 2004.

E.0.17 Nick Feamster, Hari Balakrishnan, Jennifer Rexford, Aman Shaikh, and Kobus van der Merwe. The Case for Separating Routing from Routers. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, pages 5–12, Portland, OR, September 2004.

E.0.18 Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In *ACM Workshop on Privacy in the Electronic Society*, Washington, DC, October 2004.

E.0.19 Nick Feamster, Magdalena Balazinska, Winston Wang, Hari Balakrishnan, and David Karger. Thwarting Web censorship with untrusted messenger discovery. In *3rd Workshop on Privacy Enhancing Technologies*, Dresden, Germany, March 2003.

E.0.20 Nick Feamster. Practical Verification Techniques for Wide-Area Routing. In *Proc. 2nd ACM Workshop on Hot Topics in Networks (Hotnets-II)*, pages 87–92, Cambridge, MA, November 2003.

E.0.21 Nick Feamster and Hari Balakrishnan. Towards a Logic for Wide-Area Internet Routing. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, pages 289–300, Karlsruhe, Germany, August 2003.

E.0.22 David G. Andersen, Nick Feamster, Steve Bauer, and Hari Balakrishnan. Topology inference from BGP routing dynamics. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.
Acceptance rate: 42%

E.0.23 Nick Feamster and Hari Balakrishnan. Packet loss recovery for streaming video. In *Proc. 12th International Packet Video Workshop (PV 2002)*, Pittsburgh, PA, April 2002.

E.0.24 Nick Feamster, Deepak Bansal, and Hari Balakrishnan. On the interactions between congestion control and layered quality adaptation for streaming video. In *11th International Packet Video Workshop*, Kyongju, Korea, May 2001.

F. Other

F.1. Submitted Journal Papers

F.1.1 Mohammed Mukarram bin Tariq, Vytautas Valancius, Kaushik Bhandakar, Amgad Zeitoun, Nick Feamster, and Mostafa Ammar. Answering "What-If" Deployment and Configuration Questions with WISE: Techniques and Deployment Experience.

F.1.2 Nick Feamster and Hari Balakrishnan. Correctness Properties for Internet Routing. *IEEE/ACM Transactions on Networking*, December 2005. *In Submission*.

F.2. Submitted Conference and Workshop Papers

F.2.1 Mukarram bin Tariq, Jake Brutlag, Natalia Sutin, Nick Feamster, and Mostafa Ammar. Answering How-to Questions for Mitigating High-latency Web Transactions with HIP. *In Submission.*

F.2.2 Nick Feamster, Brighten Godfrey, Nick McKeown, Guru Parulkar, Jennifer Rexford, and Scott Shenker. Architecting for Innovation. *In Submission.*

F.2.3 Srikanth Sundaresan, Cristian Lumezanu, and Nick Feamster. Autonomous Traffic Engineering with Self-Configuring Topologies. *In Submission.*

F.2.4 Yogesh Mundada, Nick Feamster, and Rob Sherwood. Distributed Implementation of Coordinated, Network-Wide Policies and Protocols with FlowFlex. *In Submission.*

F.2.5 Srikanth Sundaresan, Lucas Di Cioccio, Nick Feamster, and Renata Teixeira. Which Factors Affect Access Network Performance?, November 2009. *In Preparation.*

F.3. Other Technical Reports, Unrefereed Papers, and Drafts in Preparation

F.3.1 Vytautas Valancius, Nick Feamster, Ramesh Johari, and Vijay Vazirani. A Market for Internet Connectivity, March 2010. *In Preparation.*

F.3.2 Robert Lychev and Nick Feamster. Derailing Depeering: Incentives for Interdomain Contracts, March 2010. *In Preparation.*

F.3.3 Anirudh Ramachandran, Hitesh Khandelwal, Shuang Hao, Nick Feamster, and Santosh Vempala. A Dynamic Reputation Service for Spotting Spammers, March 2010. *In Preparation.*

F.3.4 Anirudh Ramachandran, Nick Feamster, Kobus van der Merwe, Balachander Krishnamurthy, and Oliver Spatschek. Fishing for Phishing in the Network Stream. March 2010. *In Preparation.*

F.3.5 Vytautas Valancius and Nick Feamster. Managing BGP Routes with a BGP Session Multiplexer. Technical Report GT-CS-08-05, Georgia Tech School of Computer Science, July 2008.

F.3.6 Anirudh Ramachandran, Kaushik Bhandakar, Mohammed Mukarram bin Tariq, and Nick Feamster. Packets with Provenance. Technical Report GT-CS-08-02, Georgia Tech School of Computer Science, February 2008.

F.3.7 Shuang Hao and Nick Feamster. Estimating Botnet Populations from Attack Traffic. April 2008.

F.3.8 Sapan Bhatia, Murtaza Motiwala, Wolfgang Muhlbauer, Vytautas Valancius, Andy Bavior, Nick Feamster, Larry Peterson, and Jennifer Rexford. Hosting Virtual Networks on Commodity Hardware. Technical Report GT-CS-07-10, Georgia Institute of Technology, Atlanta, GA, October 2007.

F.3.9 Nick Feamster, Murtaza Motiwala, and Andy Bavier. In-Band Network Path Diagnosis. *In Submission.*

F.3.10 Yiyi Huang, Nick Feamster, Renata Teixeira, and Christophe Diot. Making Tomography Practical: Scalable Network Monitoring for Fault Diagnosis. *In submission.*

F.3.11 Nick Feamster, Ramesh Johari, and Vijay Vazirani. AGORA: A Market for Internet Connectivity. In *Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*, Princeton, NJ, May 2007.

F.3.12 Murtaza Motiwala, Nick Feamster, and Santosh Vempala. Improving Interdomain Routing Security with BGP Path Splicing. In *Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*, Princeton, NJ, May 2007.

F.3.13 Vytautas Valancius and Nick Feamster. Layering the Interdomain Routing Layer. In *Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*, Princeton, NJ, May 2007.

F.3.14 Nick Feamster, Lixin Gao, and Jennifer Rexford. How to Lease the Internet in Your Spare Time. *ACM Computer Communications Review*, 37(1), January 2007. Editorial section.

F.3.15 Nick Feamster. Can Information from End Systems Improve Routing? In *Workshop on Internet Routing Evolution and Design (WIRED)*, Atlanta, GA, October 2006.

F.3.16 Murtaza Motiwala and Nick Feamster. Network Troubleshooting on Data Plane Coattails. In *Workshop on Internet Routing Evolution and Design (WIRED)*, Atlanta, GA, October 2006.

F.3.17 David G. Andersen and Nick Feamster. Challenges and opportunities in Internet data mining. Technical Report CMU-PDL-06-102, Carnegie Mellon University, January 2006.

F.3.18 Nick Feamster and Hari Balakrishnan. Verifying the correctness of wide-area Internet routing. Technical Report MIT-LCS-TR-948, Massachusetts Institute of Technology, May 2004.

F.3.19 Nick Feamster. Rethinking routing configuration: Beyond stimulus-response reasoning. In *Workshop on Internet Routing Evolution and Design (WIRED)*, Mt. Hood, OR, October 2003.

F.3.20 Nick Feamster and Jennifer Rexford. Network-Wide BGP Route Prediction for Traffic Engineering. In *Proc. SPIE ITCom*, volume 4868, pages 55–68, Boston, MA, August 2002.

F.3.21 Nick Feamster, Jennifer Rexford, and Jay Borkenhagen. Controlling the impact of BGP policy changes on IP traffic. Technical Report 011106-02, AT&T Labs–Research, Florham Park, NJ, November 2001.

F.3.22 Russ White, B. Akyol, and Nick Feamster. *Considerations in Validating the Path in Routing Protocols*. IETF, June 2005.

F.4. Software

F.4.1 *Campus-Wide OpenFlow Deployment: Access and Information Flow Control for Enterprise Networks.* Resonance is a system for controlling access and information flow in an enterprise network. Network operators currently use access control systems that are coarse-grained (i.e., it is difficult to apply specialized policy to individual users) and static (i.e., it is difficult to quickly change the extent of a user's access). Towards fixing these problems, we have developed a system that allows network operators to program network policy using a controller that is distinct from the switch itself and can be programmed to implement network-wide policy. We have implemented and deployed this system in an operational network that spans two buildings on the Georgia Tech campus; the network sees regular use, and a deployment in Georgia Tech dormitories or the wireless network is planned for the near future. We demonstrated the function of this network at the 7th GENI Engineering Conference in March 2010. See `http://groups.geni.net/geni/wiki/DTunnels` for details.

F.4.2 *NANO: Network Access Neutrality Observatory.* The Network Access Neutrality Observatory (NANO) is a system to help users determine whether their traffic is being discriminated against by an access ISP. In constrast to existing systems for detecting network neutrality violations, NANO makes no assumptions about the mechanism for discrimination or the services that the ISP might be discriminated against. NANO has been released in collaboration with Google as part of the Measurement Lab project. A preliminary version of the software was released to a small group of users in March 2009 for testing; a complete release is available for download at: `http://gtnose.net/nano/`.

F.4.3 *Implenentation of GENI Prototype: Virtual Networks and BGP Session Multiplexer.* In the process of developing software for the NSF-Sponsored GENI Project Office. This project (1) adds facilities and functions to the VINI testbed to enable experiments to carry traffic from real users; and (2) increases the experimental use of the VINI testbed by providing a familiar experiment management facility. The deliverables for this project all comprise software for supporting external connectivity and flexible, facile experimentation on the GENI testbed. The primary deliverables are a BGP session multiplexer—a router based on the Quagga software routing suite, software support for virtual tunnel and node creation, and integration of the above functionality with clearinghouse services developed as part of the ProtoGENI project. See `http://groups.geni.net/geni/wiki/DTunnels`.

This project contributes to GENI design and prototyping through BGP mux development integration with ISPs; tunnel and topology establishment and management; ProtoGENI clearinghouse integration; and support for isolation and resource swapout. With researchers at Princeton, we have also built VINI, a large distributed testbed for specifying virtual network topologies and experimenting with routing protocols and architectures in a controlled, realistic emulation environment. See `http://vini-veritas.net/` for details.

F.4.4 *rcc: router configuration checker*. Static configuration analysis tool for Border Gateway Protocol (BGP) routing configuraions. Downloaded by over 100 network operators and many large, nationwide backbone ISPs around the world. See `http://gtnoise.net/rcc/` for details.

F.4.5 *Infranet*. System for circumventing Web censorship firewalls (*e.g.*, those in China, Saudia Arabia, etc.). Available on Sourceforge. Featured in articles in *Technology Review*, *New Scientist*, and *Slashdot*. See `http://nms.lcs.mit.edu/projects/infranet/`.

F.4.6 *The Datapository*. Originally the "MIT BGP Monitor", the Datapository is growing to support multiple data feeds (*e.g.*, spam, end-to-end measurement probes, traceroutes, Abilene data, etc.). Currently used by researchers at Georgia Tech, Carnegie Mellon, University of Michigan, Princeton, MIT. See `http://www.datapository.net/` for details.

F.4.7 *Secure BGP Implementation*. Implementation of S-BGP in the Quagga software router. Our implementation may be used by Randy Bush and Geoff Huston in their project to develop a certificate infrastructure for secure routing protocols.

F.4.8 *SR-RTP*. Transport protocol for selective retransmission of packets in an MPEG video stream. Incorporated into "Oxygen TV" for MIT Project Oxygen. Some ideas incorporated into the OpenDivX video transport protocol.

F.5. Conference Posters and Demos

F.5.1 Vytautas Valancius, Hyojoon Kim, and Nick Feamster. Transit Portal: BGP Connectivity as a Service. In *Proc. ACM SIGCOMM*, New Delhi, India, August 2010. Demo.

F.5.2 Sam Burnett, Nick Feamster, and Santosh Vempala. Circumventing Censorship with Collage. In *Proc. ACM SIGCOMM*, New Delhi, India, August 2010. Demo.

F.5.3 Srikanth Sundaresan, Cristian Lumezanu, Nick Feamster, and Pierre Francois. Traffic Engineering with Self-Configuring Topologies. In *Proc. ACM SIGCOMM*, New Delhi, India, August 2010.

F.5.4 Anirudh Ramachandran, Yogesh Mundada, Mukarram bin Tariq, and Nick Feamster. Securing Enterprise Networks with Traffic Tainting. In *Proc. ACM SIGCOMM*, Barcelona, Spain, August 2009. Demo.

F.5.5 Vytautas Valancius, Nick Feamster, Jennifer Rexford, and Aki Nakao. Transit Portal: Bringing Connectivity to the Cloud. In *Proc. ACM SIGCOMM*, Barcelona, Spain, August 2009. Demo.

F.5.6 Murtaza Motiwala, Megan Elmore, Yogesh Mundada, and Nick Feamster. Network and End-System Support for Transparent Use of Multiple Paths. In *Proc. ACM SIGCOMM*, Barcelona, Spain, August 2009. Demo.

F.5.7 Sam Burnett, Nick Feamster, and Santosh Vempala. Circumventing Internet Censorship with Collage. In *Proc. 6th Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, April 2009. Demo.

F.5.8 Anirudh Ramachandran, Kaushik Bhandakar, Mohammed Mukarram bin Tariq, and Nick Feamster. Packets with Provenance. In *Proc. ACM SIGCOMM*, Seattle, WA, August 2008.

F.5.9 Yogesh Mundada, Murtaza Motiwala, , Vytautas Valancius, Andy Bavier, Nick Feamster, Larry Peterson, Sapan Bhatia, and Jennifer Rexford. Trinity: A Framework for Managing Wide-Area Virtual Networks. In *Proc. 5th Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, April 2008.

F.5.10 Vytautas Valancius and Nick Feamster. Multiplexing BGP Sessions with BGP-Mux. In *3rd International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, New York, NY, December 2007.

F.5.11 Nadeem Syed, Nick Feamster, and Alex Gray. Predicting bad behavior. In *NIPS Workshop on Machine Learning in Adversarial Environments for Computer Security*, Whistler, Canada, December 2007.

F.5.12 Murtaza Motiwala, Andy Bavier, and Nick Feamster. In-Band Network Troubleshooting. In *Proc. Fourth Symposium on Networked Systems Design and Implementation (NSDI)*, Cambridge, MA, April 2007.

F.5.13 Anirudh Ramachandran and Nick Feamster. Understanding the Network-Level Behavior of Spammers. In *Proc. Third Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, May 2006.

F.5.14 Nick Feamster and Hari Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In *Proc. First Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

G. Research Proposals and Grants (Principal Investigator)

1. Approved and Funded

G.1.1 **Network-Wide Configuration Testing and Synthesis**
Sponsor: National Science Foundation
Investigator(s): N. Feamster (PI), A. Akella
Amount: *$500,000 for 3 years*
Awarded: June 2010

G.1.2 **MEDITA - Multi-layer Enterprise-wide Dynamic Information-flow Tracking & Assurance**
Sponsor: National Science Foundation
Investigator(s): N. Feamster, A. Orso (PI), M. Prvulovic
Amount: *$900,000 for 3 years*
Awarded: March 2010

G.1.3 **Campus Network Access and Admission Control with Openflow**
Sponsor: National Science Foundation
Investigator(s): N. Feamster (PI), R. Clark
Amount: *$300,000 for 3 years*
Awarded: January 2010

G.1.4 **Studying DNS Traffic Patterns**
Sponsor: Verisign
Investigator(s): N. Feamster
Amount: *$30,000 for 1 year*
Awarded: November 2009

G.1.5 **CIFellowship for Cristian Lumezanu**
Sponsor: National Science Foundation
Investigator(s): C. Lumezanu, N. Feamster (PI)

Amount: *$140,000 for 1 year*
Awarded: November 2009

G.1.6 **Military Network Protocol**
Sponsor: DARPA Subcontract
Investigator(s): N. Feamster
Amount: *$37,000 for 1 year*
Awarded: November 2009

G.1.7 **Botnet Attribution and Removal: From Axioms to Theories to Practice**
Sponsor: Office of Naval Research
Investigator(s): W. Lee (PI), D. Dagon, J. Giffin, N. Feamster, K. Shin, F. Jahanian, M. Bailey, J. Mitchell, G. Vigna, C. Kruegel
Amount: *$7,500,000 for 5 years*
Awarded: August 2009

G.1.8 **Taint-based Information Tracking in Networked Systems**
Sponsor: National Science Foundation Trusted Computing Program
Investigator(s): N. Feamster
Amount: *$450,000 for 3 years*
Awarded: August 2009

G.1.9 **Towards a Market for Internet Connectivity**
Sponsor: Office of Naval Research
Investigator(s): N. Feamster (PI), R. Johari, V. Vazirani
Amount: *$350,000 for 1 year*
Awarded: March 2009

G.1.10 **Bringing Experimenters and External Connectivity to GENI**
Sponsor: GENI Project Office
Investigator(s): N. Feamster
Amount: *$320,000 for 3 years*
Awarded: September 2008

G.1.11 **Routing Without Recomputation**
Sponsor: Cisco Systems
Investigator(s): N. Feamster
Amount: *$96,019 for 1 year*
Awarded: September 2008

G.1.12 **CLEANSE: Cross-Layer Large-Scale Efficient Analysis of Network Activities to Secure the Internet**
Sponsor: National Science Foundation Cybertrust Program
Investigator(s): W. Lee (PI), N. Feamster and others
Amount: *$1,200,000 for 5 years*
Awarded: September 2008

G.1.13 **Virtual Center for Network and Security Data**
Sponsor: Department of Homeland Security
Investigator(s): N. Feamster
Amount: *$48,000 for 2 years*
Awarded: March 2008

G.1.14 **Sloan Research Fellowship**
Sponsor: Alfred P. Sloan Foundation

Investigator(s): N. Feamster
Amount: *$45,000 for 2 years*
Awarded: February 2008

G.1.15 **Enabling Security and Network Management Research for Future Networks**
Sponsor: National Science Foundation CRI-IAD Program
Investigator(s): N. Feamster (PI), Z. Mao, W. Lee
Amount: *$397,426 for 3 years*
Awarded: February 2008

G.1.16 **SMITE: Scalable Monitoring in the Extreme**
Sponsor: DARPA BAA 07-52: Scalable Network Monitoring
Investigator(s): N. Feamster (PI), W. Lee
Amount: *$250,000 for 2 years*
Awarded: January 2008

G.1.17 **Countering Botnets: Anomaly-Based Detection, Comprehensive Analysis, and Efficient Mitigation**
Sponsor: Department of Homeland Security BAA07-09
Investigator(s): W. Lee (PI), N. Feamster, J. Giffin
Amount: *$1,050,730 for 2 years*
Awarded: January 2008

G.1.18 **Spam Filtering Research**
Sponsor: IBM Faculty Award
Investigator(s): N. Feamster
Amount: *$ 7,500 (unrestricted gift)*
Awarded: June 2007

G.1.19 **SCAN: Statistical Collaborative Analysis of Networks**
Sponsor: National Science Foundation NeTS-NBD Program
Investigator(s): N. Feamster (PI), A. Gray, J. Hellerstein, C. Guestrin
Amount: *$ 95,000 for 3 years.*
Awarded: June 2007

G.1.20 **Towards an Accountable Internet Architecture**
Sponsor: National Science Foundation CyberTrust Program (Team Proposal)
Investigator(s): D. Andersen, H. Balakrishnan, N. Feamster (PI), S. Shenker
Amount: *$ 300,000 for 3 years.*
Awarded: May 2007

G.1.21 **Fish4Phish: Fishing for Phishing in a Large Pond**
Sponsor: AT&T Labs—Research
Investigator(s): N. Feamster (PI), O. Spatscheck, K. van der Merwe
Amount: *Funding for summer intern.*
Awarded: February 2007

G.1.22 **Improving Network Operations with a View from the Edge.**
Sponsor: National Science Foundation CAREER Program
Investigator(s): N. Feamster (PI)
Amount: *$400,000 for 5 years.*
Awarded: January 2007

G.1.23 **Equipment Donation for Network Operations Research**
Sponsor: Intel Corporation

Investigator(s): N. Feamster
Amount: *$30,000*
Awarded: October 2006

G.1.24 **CABO: Concurrent Architectures are Better than One**
Sponsor: National Science Foundation NeTS-FIND Program
Investigator(s): N. Feamster (PI), L. Gao, J. Rexford
Amount: *$ 300,000 for 4 years*
Awarded: June 2006

G.1.25 **Verification and Modeling of Wide-Area Internet Routing**
Sponsor: Cisco Systems University Research Program
Investigator(s): N. Feamster and H. Balakrishnan (PI)
Amount: *$ 95,500 for 1 year.*
Awarded: June 2004

2. Pending

G.2.1 **Automatic Configuration Generation for Data Center Configurations**
Sponsor: Yahoo Faculty Research and Engagement Program
Investigator(s): N. Feamster
Amount: *$25,000 for 1 year*
Submitted: May 2010

G.2.2 **Managing the Cost of Network Traffic**
Sponsor: Yahoo Faculty Research and Engagement Program
Investigator(s): N. Feamster
Amount: *$25,000 for 1 year*
Submitted: May 2010

G.2.3 **Architecting for Innovation**
Sponsor: National Science Foundation
Investigator(s): H. Balakrishnan, N. Feamster, B. Godfrey, N. McKeown, J. Rexford, S. Shenker (PI)
Amount: *$9,000,000 for 3 years*
Submitted: April 2010

3. Not Funded

Available upon request.

H. Research Proposals and Grants (Contributor)

1. Approved and Funded

H.1.1 **Development of a shared network measurement storage and analysis infrastructure**
Sponsor: National Science Foundation Major Research Infrastructure (MRI)
Investigator(s): D. Andersen, D. Song, C. Wang, H. Zhang
Amount: *$ 101,488.96*
User and developer for proposed infrastructure; possible equipment money to Georgia Tech .
Submitted: February 2006.

2. Pending

3. Not Funded

Available upon request.

I. Other

I.0.1 **In-Band, Bottom-Up Support for Network Accountability**
Sponsor: N. Feamster, W. Lee, M. Ahamad
Investigator(s): DARPA Strategic Technology Office
Amount: *White paper / Request for Information.*
Submitted: February 2007

I.0.2 **Towards an Accountable Internet Architecture**
Sponsor: D. Andersen, H. Balakrishnan, N. Feamster, S. Shenker
Investigator(s): DARPA Strategic Technology Office
Amount: *White paper / Request for Information.*
Submitted: February 2007

J. Research Honors and Awards

2010	Georgia Tech Sigma Xi Young Faculty Award
2009	Best Paper, *Passive and Active Measurement Conference*
2009	Georgia Tech Sigma Xi Best Undergraduate Research Advisor
2008	**NSF Presidential Early Career Award for Scientists and Engineers (PECASE)**
2008	**Alfred P. Sloan Fellowship**
2008	Georgia Tech College of Computing Outstanding Junior Faculty Research Award
2007	IBM Faculty Award
2007	NSF CAREER Award
2006	Best Student Paper (Advisor), *ACM SIGCOMM* (Premier Networking Conference)
2006	George M. Sprowls honorable mention for best Ph.D. thesis in computer science, MIT
2005	Best Paper, *2nd USENIX Symposium on Networked Systems Design and Implementation*
2002	Best Student Paper, *11th USENIX Security Symposium*
2001	Best Student Paper, *10th USENIX Security Symposium*
2002–2005	NSF Graduate Research Fellow
2001	MIT William A. Martin Memorial Thesis Award for Best EECS Master's Thesis

III. SERVICE

A. Professional Activities

A.1. Memberships and Activities in Professional Societies

Member, ACM Special Interest Group on Data Communications (SIGCOMM)

Member, USENIX Technical Association

A.2. Conference Committee Activities

2010 Program Committee, *ACM SIGCOMM Poster and Demo Session*

2010 Program Committee, *ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures (VISA)*

2010 **Program Committee Co-Chair**, *USENIX Workshop on Internet Network Management (INM/WREN)*

2010 Program Committee, *ACM SIGMETRICS*

2010 Program Committee, *IEEE Symposium on Internet Security and Privacy*

2010 Program Committee, *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*

2010 Program Committee *3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '10)*

2009 Program Committee, *ACM SIGCOMM Workshop on Research on Enterprise Networks (WREN)*

2009 Program Committee, *ACM SIGCOMM Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*

2009 Program Committee, *USENIX Technical Conference*

2009 Program Committee, *ACM SIGCOMM Workshop on Economics of Networked Systems (NetEcon)*

2009 Program Committee *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '09)*

2009 **Poster/Demo Co-Chair**, *ACM SIGCOMM*

2009 Program Committee, *ACM SIGMETRICS*

2009 Program Committee, *6th ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*

2008 Program Committee, *3rd International Workshop on Real Overlays & Distributed Systems*

2008 Program Committee, *ACM SIGCOMM Internet Measurement Conference*

2008 Program Committee, *ACM Conference on Computer and Communications Security (CCS)*

2008 Program Committee, *ACM SIGCOMM*

2008 Program Committee, *ACM SIGMETRICS*

2008 Program Committee, *ACM SIGCOMM Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*

2008 Program Committee, *ACM SIGCOMM Workshop on Economics of Networked Systems (NetEcon)*

2008	Program Committee, *17th International World Wide Web Conference* (Security/Privacy Track)
2008	Program Committee, *ACM SIGMETRICS Workshop on Internet Network Management*
2008	Program Committee, *16th IEEE LAN/MAN Workshop*
2007	Program Committee, *3rd Annual CoNext Conference*
2007	Program Committee, *ACM SIGCOMM Workshop on Internet Management (INM)*
2007	Program Committee, *ACM SIGCOMM Student Poster Session*
2007	Program Committee, *ACM SIGMETRICS Workshop on Mining Internet Data (MineNet)*
2007	**Co-Organizer**, DIMACS Workshop Series on Internet Security
2007	Program Committee, *4th Conference on Email and Anti-Spam (CEAS)*
2007	Program Committee, *3rd USENIX Workshop on Steps to Reduce Unwanted Traffic on the Internet (SRUTI)*
2007	Program Committee, *USENIX Technical Conference*
2007	Program Committee, *CoNext*
2007	Program Committee, *16th International World Wide Web Conference* (Security/Privacy Track)
2007	Program Committee Co-Chair, *ACM/USENIX Workshop on Networks meet Databases (NetDB)*
2006–	Program Committee, *North American Network Operators Group (NANOG)*
2006	Organizer, *Workshop on Internet Routing Evolution and Design*
2006	Program Committee, *ACM SIGCOMM Internet Measurement Conference*
2006	**Program Committee Co-Chair**, *Workshop on the Economics of Networked Systems (NetEcon)*
2006	**Program Committee Co-Chair**, *CoNext Student Workshop*
2006	Program Committee, *2nd Annual CoNext Conference*
2006	Program Committee, *ACM SIGCOMM Workshop on Internet Network Management*
2006	Program Committee, *IEEE Symposium on Security and Privacy*
2006	Program Committee, *IEEE Infocom Student Poster Session*
2006	Program Committee, *IEEE International Conference on Internet Surveillance and Protection*

External reviewer for *IEEE/ACM Transactions on Networking*, *SIGCOMM* (2002, 2003, 2004, 2006, 2007), *SOSP* (2001, 2003), *Infocom* (2004, 2006), *HotNets* (2003), *HotOS* (2001), *USENIX Security Symposium* (2002), *ACM Computer Communication Review*, *IEEE Network Magazine*, *IEEE Journal on Selected Areas in Communications*, *Image Communication* (EURASIP), *ASPLOS* (2004), *MobiSys* (2004), *USENIX* (2005, 2006), *NSDI* (2005, 2006), *IPTPS* (2005), *Workshop on Privacy Enhancing Technologies* (2006).

A.3. Workshops and External Courses

Unless otherwise noted, all listed activities are invited speaking invitations for workshops, tutorials, and symposia.

June 2010	African Network Operators Group Tutorial, Kigali, Rwanda

March 2010	**Co-Organizer**, DIMACS Workshop on Secure Internet Routing, New Brunswick, NJ
June 2009	**Organizer**, NSF Security Driven Architectures Workshop, Arlington, VA
March 2009	Workshop on Re-Architecting the Internet (NetArch), Monte Verita, Switzerland
February 2009	CAIDA Active Internet Measurement Systems (AIMS) Workshop, San Diego, CA
October 2008	Panelist at IEEE CCW, Steamboat Springs, Colorado
October 2008	Panelist at ACM SIGCOMM Internet Measurement Conference, Athens, Greece
June 2008	Google Workshop on Internet Measurement, Mountain View, CA
May 2008	Tutorial at 26th Brazilian Symposium on Computer Networks and Distributed Systems, Rio de Janeiro, Brazil
March 2008	Co-Organizer for DIMACS Workshop on Secure Internet Routing
November 2007	NSF/DARPA/ARO NCDI Workshop, College Park, MD
August 2007	DIMACS Tutorial on Algorithms for Next Generation Networks, New Brunswick, NJ
July 2007	INTIMATE Workshop on Methods and Tools for Network Analysis, Paris, France
May 2007	Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO), Princeton, NJ
April 2007	NeXtworking, the Second COST-NSF Workshop on Future Internet, Berlin, Germany
February 2007	GIG Routing and Addressing Workshop, Washington, DC
February 2007	DARPA Workshop on Assurable Global Networking, Washington, DC
December 2006	Next-Generation Internet Workshop, Lisbon, Portugal
September 2006	Clean Slate Network Research Symposium, Cambridge, UK
October 2006	**Co-organizer**, Workshop on Internet Routing Evolution and Design (WIRED), Atlanta, GA
August 2006	Cisco Routing Research Symposium, San Jose, CA
June 2006	ARO-DARPA-DHS Special Workshop on Botnets, Arlington, VA
June 2006	Microsoft Research EdgeNet Workshop, Snoqualmie, WA
June 2006	Microsoft Research "Networking on the Edge" Workshop on Network Management, Seattle, WA
February 2006	NSF Workshop on Theory of Networked Computation, Princeton, NJ
January 2006	Cisco Network Management Summit, San Jose, CA

B. On-Campus Georgia Tech Committees

Spring 2010	Faculty Recruiting Committee
Fall 2009	Strategic Planning Committee
Spring 2009–10	Dean Search Committee
Spring 2009	Faculty Recruiting Committee
Spring 2008	Faculty Recruiting Committee
Fall 2006–	Area Coordinator, Networking Area Group

Fall 2006	Co-Organizer, Networking and Telecommunications Group Open House
Spring 2006–	College of Computing Strategic Planning Committee
Spring 2006	Ph.D. Recruiting Weekend Organizing Committee, CSS Division Leader

C. Member of Ph.D. Examining Committees

Ph.D. Thesis Committee – Georgia Tech

1. David Dagon, College of Computing, Georgia Tech., Spring 2010.
   Principal Advisor: Professor Wenke Lee.
2. Junjie Zhang, College of Computing, Georgia Tech., Spring 2010.
   Principal Advisor: Professor Wenke Lee.
3. Amogh Dhamdhere, College of Computing, Georgia Tech, Spring 2009.
   Principal Advisor: Professor Constantine Dovrolis.
4. Guofei Gu, College of Computing, Georgia Tech, Spring 2008.
   Principal Advisor: Professor Wenke Lee.
5. Steve Webb, College of Computing, Georgia Tech, Spring 2008.
   Principal Advisor: Professor Calton Pu.
6. Vibhore Kumar, College of Computing, Georgia Tech, Spring 2008.
   Principal Advisor: Professor Karsten Schwan.
7. Prahlad Fogla, College of Computing, Georgia Tech, Spring 2007.
   Principal Advisor: Professor Wenke Lee.
8. Srinivasan Seetharaman, College of Computing, Georgia Tech, Spring 2007.
   Principal Advisor: Professor Mostafa Ammar.
9. Sridhar Srinivasan, College of Computing, Georgia Tech., Spring 2006.
   Principal Advisor: Professor Ellen Zegura.
10. Xenofontas Dimitropoulos, Electrical and Computer Engineering, Georgia Tech, Spring 2006.
    Principal Advisor: Professor George Riley.
11. Qi Zhao, College of Computing, Georgia Tech, Spring 2006.
    Principal Advisor: Professor Jim Xu.
12. Ruomei Gao, College of Computing, Georgia Tech., Fall 2005.
    Principal Advisor: Professors Ellen Zegura and Constantine Dovrolis.

Other — Question writer for Ph.D. Qualifying Exam

1. Bilal Anwer, College of Computing, Georgia Tech., Spring 2010.
   Principal Advisor: Professor Nick Feamster.
2. Sam Burnett, College of Computing, Georgia Tech., Spring 2010.
   Principal Advisor: Professor Nick Feamster.
3. Shuang Hao, College of Computing, Georgia Tech., Spring 2010.
   Principal Advisor: Professor Nick Feamster.
4. Partha Kanuparthy, College of Computing, Georgia Tech., Spring 2010.
   Principal Advisor: Professor Constantine Dovrolis.

5. Maria Konte, College of Computing, Georgia Tech., Spring 2010.
   Principal Advisor: Professor Nick Feamster.

6. Robert Lychev, College of Computing, Georgia Tech., Spring 2010.
   Principal Advisor: Professor Nick Feamster.

7. Yogesh Mundada, College of Computing, Georgia Tech., Spring 2010.
   Principal Advisor: Professor Nick Feamster.

8. Cong Shi, College of Computing, Georgia Tech., Spring 2010.
   Principal Advisor: Professor Mostafa Ammar.

9. Yiyi Huang, College of Computing, Georgia Tech., Spring 2008.
   Principal Advisor: Professor Nick Feamster.

10. Anirudh Ramachandran, College of Computing, Georgia Tech., Spring 2008.
    Principal Advisor: Professor Nick Feamster.

11. Murtaza Motiwala, College of Computing, Georgia Tech., Spring 2009.
    Principal Advisor: Professor Nick Feamster.

12. Vytautas Valancius, College of Computing, Georgia Tech., Spring 2009.
    Principal Advisor: Professor Nick Feamster.

13. Mehmet Demirci, College of Computing, Georgia Tech., Spring 2009.
    Principal Advisor: Professor Mostafa Ammar.

14. Ahmed Mansy, College of Computing, Georgia Tech., Spring 2009.
    Principal Advisor: Professor Mostafa Ammar.

15. Tonqing Qiu, College of Computing, Georgia Tech., Spring 2009.
    Principal Advisor: Professor Jim Xu.

16. Nan Hua, College of Computing, Georgia Tech., Spring 2009.
    Principal Advisor: Professor Jim Xu.

17. Chuck Zhao, College of Computing, Georgia Tech., Spring 2009.
    Principal Advisor: Professor Jim Xu.

18. Mukarram Bin Tariq, College of Computing, Georgia Tech., Spring 2007.
    Principal Advisor: Professor Nick Feamster.

19. Zhenshun Zhang, College of Computing, Georgia Tech., Spring 2007.
    Principal Advisor: Professor Raghupathy Sivakumar.

20. Yang Chen, College of Computing, Georgia Tech., Spring 2006.
    Principal Advisor: Professor Jim Xu.

D. Consulting, Advisory, and Other External Appointments

1. Technical Advisory Board, Guavus, Inc. `http://www.guavus.com/`

2. Technical Advisory Board, Anchor Intelligence, Inc. `http://www.fraudwall.net/`

3. Consultant, Damballa, Inc. `http://www.anchorintelligence.com/`

E. Research Project Reviewer

1. National Science Foundation.
   May 2009

2. National Science Foundation.
   July 2008

3. National Science Foundation.
   June 2008

4. National Science Foundation.
   October 2007

5. National Science Foundation.
   March 2007

6. National Science Foundation.
   February 2007

7. Department of Homeland Security.
   Fall 2006 - Present
   Department of Homeland Security Predict Review Board

IV. NATIONAL AND INTERNATIONAL RECOGNITION

A. Invited Conference Session Chair

April 2010 — Session chair, *ACM/USENIX Networked Systems Design and Implementation (NSDI)*, San Jose, CA

October 2008 — Session chair, *ACM SIGCOMM HotNets-VII Workshop*, Calgary, Alberta, Canada

October 2008 — Session chair, *ACM SIGCOMM Internet Measurement Conference*, Athens, Greece

March 2008 — Session organizer, Security for the Future Internet, *NSF Cybertrust PI Meeting*, Arlington, VA

June 2007 — Session chair, Network Virtualization, *NSF NeTS-FIND PI Meeting*, Arlington, VA

June 2006 — Session chair, Mechanism Design and Networking, *First Workshop on the Economics of Networked Systems (NetEcon06)*, Ann Arbor, MI

B. Patents

March 2007 — "Method and System for Detecting and Responding to Attacking Networks", U.S. Patent Application # 11/538,212

C. Editorial and Reviewer Work for Technical Journals and Publishers

External reviewer for *IEEE/ACM Transactions on Networking*

External reviewer for *IEEE Journal on Selected Areas in Communications*

External reviewer for *IEEE Computer Networks*

External reviewer for *ACM Transactions on Information and Systems Security*

V. OTHER CONTRIBUTIONS

A. Seminar Presentations

A.0.1 — Network-Level Spam and Scam Defenses. In *African Network Operators Group*, Kigali, Rwanda, June 2010.

A.0.2 — SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware. In *National Institute of Information and Communications Technology*, Tokyo, Japan, June 2010.

A.0.3 — Network-Level Spam and Scam Defenses. In *National Institute of Information and Communications Technology*, Tokyo, Japan, June 2010.

A.0.4 — Wide-Area Routing for Distributed Services. In *University of Tokyo*, Tokyo, Japan, June 2010.

A.0.5 — SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware. In *Stanford Networking Seminar*, Stanford, CA, June 2010.

A.0.6 — Network-Level Spam and Scam Defenses. In *MIT Computer Science Department Colloquium*, Cambridge, MA, May 2010.

A.0.7 — Network-Level Spam and Scam Defenses. In *Harvard University Computer Science Department Colloquium*, Cambridge, MA, April 2010.

A.0.8 Network-Layer Support for Secure Routing and Access Control. In *DIMACS Workshop on Secure Routing*, New Brunswick, NJ, March 2010.

A.0.9 Detecting General Network Neutrality Violations with Causal Inference. In *ACM SIGCOMM CoNext*, Rome, Italy, December 2009.

A.0.10 Dynamic Access Control with Resonance. In *DIMACS Workshop on Network Management*, New Brunswick, NJ, November 2009.

A.0.11 Network-Level Spam and Scam Defenses. In *Princeton University Computer Science Department Colloquium*, Princeton, NJ, October 2009.

A.0.12 SNARE: Spatio-Temporal Network-Level Automated Reputation Engine. In *Message Anti-Abuse Working Group (MAAWG) Plenary Session*, Philadelphia, PA, October 2009.

A.0.13 A Platform for Building Flexible, Fast Virtual Networks on Commodity Hardware. In *Cisco Routing Architecture Workshop*, San Jose, CA, October 2009.

A.0.14 Network-Level Spam and Scam Defenses. In *EPFL Summer Research Institute*, Lausanne, Switzerland, June 2009.

A.0.15 Bringing Experimenters and External Connectivity to GENI. In *GENI Measurement Workshop*, Madison, WI, June 2009.

A.0.16 Network-Level Spam Defenses. In *University of Wisconsin Computer Science Department Colloquium*, Madison, WI, April 2009.

A.0.17 Trellis: A Platform for Building Flexible, Fast Virtual Networks on Commodity Hardware. In *NSF NeTS FIND PI Meeting*, Arlington, VA, April 2009.

A.0.18 Dynamics of Online Scam Hosting Infrastructure. In *Passive and Active Measurement Conference*, Seoul, Korea, April 2009.

A.0.19 Demonstration of Topology Creation Service and BGP Session Multiplexer. In *4th GENI Engineering Conference*, Miami, Florida, April 2009.

A.0.20 Network Security Research. In *Georgia Tech College of Computing Wine & Cheese Talk*, Atlanta, GA, April 2009.

A.0.21 Network-Level Spam Defenses. In *Cisco Systems Security Seminar*, San Jose, CA, March 2009.

A.0.22 Network-Level Spam Defenses. In *Stanford University Networking Seminar*, Stanford, CA, March 2009.

A.0.23 Network-Level Spam Defenses. In *University of North Carolina Computer Science Department Colloquium*, Chapel Hill, NC, March 2009.

A.0.24 Dynamics of Online Scam Hosting Infrastructure. In *CAIDA Active Internet Measurement Systems (AIMS) Workshop*, San Diego, CA, February 2009.

A.0.25 Outsourcing Network Security with Programmable Hardware. In *GENI Security Workshop*, Davis, CA, April 2009.

A.0.26 Spam, Phishing, and Online Scams:
A View from the Network-Level. In *University of Toronto Computer Science Department Colloquium*, Toronto, Ontario, Canada, December 2008.

A.0.27 Detecting and Diagnosing Network Performance Degradations with Statistical Methods. In *IPAM Workshop on New Mathematical Frontiers in Network Multi-Resolution Analysis*, Los Angeles, CA, October 2008.

A.0.28 Fighting Spam, Phishing, and Online Scams at the Network Level. In *Asian Internet Engineering Conference*, Bangkok, Thailand, November 2008.

A.0.29 Multiplexing BGP Sessions with a BGP Session Multiplexer. In *3rd GENI Engineering Conference*, Palo Alto, CA, October 2008.

A.0.30 Towards an Accountable Internet Architecture. In *The IEEE 22nd Annual Computer Communications Workshop*, Steamboat Springs, Colorado, October 2008.

A.0.31 Network-Level Spam Filtering. In *IPAM Workshop on Applications of Internet MRA to Cyber-Security*, Los Angeles, CA, October 2008.

A.0.32 Spam, Phishing, and Online Scams:
A View from the Network-Level. In *Yahoo Tech Talk*, Sunnyvale, CA, August 2008.

A.0.33 Path Splicing. In *EPFL Summer Research Institute*, Lausanne, Switzerland, July 2008.

A.0.34 Spam, Phishing, and Online Scams:
A View from the Network-Level. In *Google Tech Talk*, Mountain View, CA, June 2008. `http://www.youtube.com/watch?v=IBPg9Lyta3A`.

A.0.35 Making Tomography Practical: Scalable Network Monitoring for Fault Diagnosis. In *DIMACS Workshop on Network Tomography*, New Brunswick, NJ, May 2008.

A.0.36 Network-Based Spam Filtering. In *Internet2 Members Meeting*, Arlington, VA, April 2008.

A.0.37 Path Splicing. In *Carnegie Mellon University Computer Science Seminar*, Pittsburgh, PA, April 2008.

A.0.38 BGP, Bogons, and Spam. In *DIMACS Workshop on Secure Internet Routing*, New Brunswick, NJ, March 2008.

A.0.39 Can Great Research Be Taught? Independent Research with Cross-Disciplinary Thinking and Broader Impact. In *ACM SIGCSE Technical Symposium on Computer Science Edudation (SIGCSE)*, Portland, OR, March 2008.

A.0.40 Path Splicing. In *Colgate University Computer Science Seminar*, Hamilton, NY, November 2007.

A.0.41 Path Splicing. In *Cornell University Computer Science Symposium*, Ithaca, NY, November 2007.

A.0.42 Path Splicing. In *Bell Labs Networking Seminar*, Murray Hill, NJ, November 2007.

A.0.43 Usage-Based DHCP Lease-Time Optimization. In *ACM/USENIX Internet Measurement Conference*, San Diego, CA, October 2007.

A.0.44 Network-Based Spam Filtering. In *University of Southern California Computer Science Symposium*, Los Angeles, CA, October 2007.

A.0.45 Path Splicing with Network Slicing. In *Georgia Tech Algorithms and Randomness Center (ARC) Symposium*, Atlanta, GA, October 2007.

A.0.46 Internet Availability. In *DIMACS Tutorial on Algorithms for Next-Generation Networks*, New Brunswick, NJ, August 2007.

A.0.47 Towards an Accountable Internet Architecture. In *INTIMATE Workshop on Methods and Tools for Network Analysis*, Paris, France, July 2007.

A.0.48 Path Splicing with Network Slicing. In *UCL Networking Seminar*, Louvain La Neuve, Belgium, July 2007.

A.0.49 Composing Virtual Networks. In *Third NSF-NeTS FIND PI Meeting*, Arlington, VA, June 2007.

A.0.50 Path Splicing with Network Slicing. In *UCSD Networking and Systems Seminar*, San Diego, CA, June 2007.

A.0.51 Path Splicing with Network Slicing. In *NANOG 40*, Bellevue, WA, June 2007.

A.0.52 AGORA: A Market for Internet Connectivity. In *Workshop on Programmable Routers for Extensible Services of Tomorrow*, Princeton, NJ, May 2007.

A.0.53 Path Splicing with Network Slicing. In *NeXtworking 2007, the Second COST-NSF Workshop on Future Internet*, Berlin, Germany, April 2007.

A.0.54 Towards an Accountable Internet Architecture. In *DARPA GIG Routing and Addressing Workshop*, Arlington, VA, February 2007.

A.0.55 Towards an Accountable Internet Architecture. In *DARPA STO Assurable Global Networking Workshop*, Arlington, VA, February 2007.

A.0.56 Improving Network Operations with a View from the Edge. In *National Science Foundation CyberTrust PI Meeting*, Atlanta, GA, January 2007.

A.0.57 Network Virtualization for Experimentation and Profit. In *University of Michigan Networking Seminar*, Ann Arbor, MI, December 2006.

A.0.58 Network Virtualization for Experimentation and Profit. In *Stanford Networking Seminar*, Stanford, CA, November 2006.

A.0.59 Network Architecture Research. In *National Science Foundation NeTS-FIND Informational Meeting*, Reston, VA, November 2006.

A.0.60 Network Virtualization. In *National Science Foundation NeTS-FIND PI Meeting*, Reston, VA, November 2006.

A.0.61 Understanding the Network-Level Behavior of Spammers. In *Johns Hopkins Security and Applied Research Lab Seminar*, Baltimore, MD, October 2006.

A.0.62 Revealing Botnet Membership with DNSBL Counter-Intelligence. In *NANOG 38*, St. Louis, MO, October 2006.

A.0.63 VINI: A Virtual Network Infrastructure. In *Centre national de la recherche scientifique (CNRS)*, Paris, France, September 2006.

A.0.64 Botnets and Spam. In *Seminar: Laboratoire d'informatique de Paris 6*, Paris, France, September 2006.

A.0.65 Network Virtualization and Graph Embedding. In *Georgia Tech Algorithms and Randomness Center Seminar*, Atlanta, GA, September 2006.

A.0.66 Cabo: Concurrent Architectures are Better than One. In *Cisco Internet Routing Research Symposium*, San Jose, CA, August 2006.

A.0.67 Botnets and Spam. In *ARO-DARPA-DHS Special Workshop on Botnets*, Arlington, VA, June 2006.

A.0.68 Understanding the Network-Level Behavior of Spammers. In *CS 548: Internet and Distributed Systems Seminar*, Stanford, CA, June 2006.

A.0.69 Understanding the Network-Level Behavior of Spammers. In *NANOG 37*, San Jose, CA, June 2006.

A.0.70 Campus and Personal Network Troubleshooting. In *Microsoft Research EdgeNet 2006*, Snoqualmie, WA, June 2006.

A.0.71 Infranet: Circumventing Web censorship and surveillance. In *Georgia Tech NTG Seminar*, Atlanta, GA, April 2006.

A.0.72 Filtering: Sharpening Two Sides of a Double-Edged Sword. In *Interz0ne 5*, Tucker, GA, March 2006.

A.0.73 Theory of Measurement and Monitoring. In *Workshop on the Theory of Networked Computation*, Princeton, NJ, February 2006.

A.0.74 Challenges and Opportunities in Internet Data Mining. In *Cisco Network Management Summit*, San Jose, CA, February 2006.

A.0.75 BGP Spamming Agility. In *NANOG 36*, Dallas, TX, February 2006.

A.0.76 Spam Forensics. In *IEEE Security and Privacy Crystal Ball Workshop*, Berkeley, CA, February 2006.

A.0.77 Network Security. In *Georgia Tech Information Security Center Lunch*, Atlanta, GA, January 2006.

A.0.78 Geographic Locality of IP Prefixes. In *NANOG 35*, Los Angeles, CA, October 2005.

A.0.79 rcc demonstration. In *NANOG 35*, Los Angeles, CA, October 2005.

A.0.80 Detecting BGP Configuration Faults with Static Analysis. In *Princeton University Systems Reading Group*, Princeton, NJ, October 2005.

A.0.81 Proactive Techniques for Correct and Predictable Internet Routing. In *University of Cambridge Symposium*, England, United Kingdom, June 2005.

A.0.82 Proactive Techniques for Correct and Predictable Internet Routing. In *Hewlett-Packard Laboratories*, Palo Alto, CA, October 2005.

A.0.83 Open problems in BGP anomaly detection. In *CAIDA Workshop on Internet Signal Processing*, San Diego, CA, November 2004.

A.0.84 On end-to-end path failures and routing instability. In *CAIDA Workshop on Internet Signal Processing*, San Diego, CA, November 2004.

A.0.85 Wide-area network data and analysis at MIT. In *CAIDA Internet Measurement Data Catalog Workshop*, San Diego, CA, June 2004.

A.0.86 Verifying wide-area routing configuration. In *NANOG 31*, San Francisco, CA, May 2004.

A.0.87 Verifying wide-area routing configuration with **rcc**. AT&T Labs; Florham Park, NJ, February 2004.

A.0.88 Verifying wide-area routing configuration with **rcc**. In *NYU Systems Reading Group*, New York, NY, February 2004.

A.0.89 A systematic approach to BGP configuration checking. In *NANOG 29*, Chicago, IL, October 2003.

A.0.90 Characterizing path failures: Location, characterization, correlation. In *CAIDA Internet Statistics, Metrics, and Analysis Workshop*, Leiden, The Netherlands, October 2002.

A.0.91 Infranet: Circumventing Web censorship and surveillance. In *Harvard Kennedy School of Government Seminar*, Cambridge, MA, November 2002.

A.0.92 Infranet: Circumventing Web censorship and surveillance. Hewlett-Packard Laboratories; Palo Alto, CA, August 2002.

A.0.93 Infranet: Circumventing Web censorship and surveillance. In *Carnegie Mellon University SDI Seminar*, Pittsburgh, PA, April 2002.

A.0.94 Controlling the impact of BGP policy changes on IP traffic. In *NANOG 25*, Toronto, Ontario, Canada, June 2002.

VI. PERSONAL DATA

Born:	Stanford, CA
Home Address:	Atlanta, GA 30308-1490
Citizenship:	USA
Email:	`feamster - cc.gatech.edu`
WWW:	`http://www.cc.gatech.edu/~feamster`

Teaching Statement

Nick Feamster

I believe that students learn best by doing. Hands-on experience and real-world exercises not only make classes more exciting, but they also provide memorable examples and analogies. Our ability to understand abstract concepts is more limited than our ability to process concrete examples that relate to familiar ideas. Some of my own memorable classroom experiences were lectures with concrete examples; I apply a similar approach in my courses, tying abstract concepts to relevant concrete examples and hands-on experience. For example, "route hijacking" is more concrete when students can actually see network traffic going where it isn't supposed to go. Routing protocols make more sense when students can configure routers, induce failures on an experimental network, and watch the behavior of traffic over that network.

To this end, my first goal in teaching both networking and security classes is to connect textbook material with (1) real-world examples; and (2) hands-on experience. At both the undergraduate and graduate levels, I have incorporated course material that familiarizes students with the state of the art in network design, implementation, and experimentation; for example, I have students run experiments on network testbeds like Emulab and VINI, with real software routers that they can configure. Where appropriate, I have also allowed students to shape the course themselves by bringing in real-world examples, from which I will design a lecture. For example, in my undergraduate networking course, I maintain a wiki where students could post relevant "current events" in networking and vote on which topics they would like to see covered. Based on that input, I incorporate new material into the syllabus—teaching not just the current event itself, but also the relevant foundational material. I also bring my own current events to lecture and relate them to the textbook being covered. This takes more effort than dusting off old notes, but it keeps students engaged and helps me stay abreast of what is happening both in industry and in research.

My second goal is to elevate course material so that students are not just learning mechanics of protocols and systems, but also gaining a deeper understanding of the *concepts* that underlie their design. My reasoning here is two-fold. First, I believe that the traditional classroom lecture is "going the way of the blackboard". With so many computing and communications tools for aggregating and processing information, conventional lectures are no longer always the most efficient way to convey textbook information. In my lectures, I try to go beyond what is taught in the textbook—rather than teaching only mechanics, I ask students about design rationales, and whether they would make the same choices today, given the changing roles of communications networks. Given that networking is still maturing as a field, there is a tendency to focus on protocol details, which may change over time. Ten years from now, I would like someone who took my class to be able to say that the concepts they learned have remained applicable. I try to organize topics around higher-level concepts (*e.g.*, randomization, caching, tree formation, identity), so that even as protocol details continue to change, the concepts that they learn remain valuable.

I enjoy teaching students how to think. With Alex Gray, I have designed a graduate course that focuses on teaching first-year graduate students how to do research. The course includes topics ranging from paper reading to fellowship applications to generating (and executing) research ideas. The value of the course is evident from student response: Students at various points along their Ph.D.—even more senior students—sit in on lectures. We presented a paper on the course at *SIGCSE* in 2008, and other computer science departments (*e.g.*, Duke, Princeton) are now starting to adopt some of the material.

I have a strong mentoring and advising record. My graduate students have received best paper awards at top conferences (including *SIGCOMM*), and my undergraduate students I have advised have garnered publications in top conferences and have gone to the best computer science graduate schools in the country. One of my most rewarding advising experiences was with undergraduate student Megan Elmore. When I first began working with undergraduate student Megan, she lacked confidence that she could be good at research. I taught her not only about research and computer networking, but I also continually encouraged her to continue in research and helped her realize her talents. Under my advisement, she received the Sigma Xi Undergraduate Research Award, the CoC undergraduate research award, and an NSF Graduate Fellowship; she is now a Ph.D. student at Stanford University.

Research Statement

Nick Feamster

Summary

My research focuses on developing tools, algorithms, and protocols that make the network easier to manage, more secure, and more available.

Nobody notices when the network works well, but everyone suffers when it doesn't. Thus, communications networks should be both secure and available. Network *security* has many facets, ranging from the ability to stop "unwanted traffic" (e.g., spam and denial-of-service attacks) to the ability to trace back attacks to their perpetrators ("accountability"). *Availability* means that the network must provide good performance for users whenever they want to use it—unfortunately, the increasing complexity of the network, coupled with hardware faults, software bugs, misconfigurations, and malice, make it difficult to achieve this goal. Unfortunately, these two important goals have also been among the most evasive. Breakthroughs require not only extensive domain knowledge, but also the ability to techniques from a wide range of areas, ranging from economics to machine learning. My work combines domain knowledge, extensive interactions with network operators, techniques from a wide range of disciplines, and—perhaps most importantly—the competence and tenacity to implement and deploy these systems in practice. This unique combination has allowed me to build one of the few networking research groups in the world that interacts directly with network operators to deploy fundamentally new systems and technologies in real-world networks.

I discover interesting and challenging practical problems through frequent discussions and meetings with network operators and people in industry. I then tackle these problems from first principles, develop new methods, and transfer these solutions back to practice in the form of working systems. I have tackled a variety of problems in network operations, ranging from real-time network diagnosis to stemming unwanted traffic like spam to architectures for fast failure recovery. Many people—most notably, operators "in the trenches"—are also working on these problems. Unfortunately, many of the people who have the domain knowledge that best equip them to solve these problems are busy with day-to-day operations, putting out fires as they arise but rarely taking time to think about fundamental changes to the network that might eradicate these problems. My research fills this niche. I first devise methods to understand the nature of the problem in practice. I tackle domain-specific problems with tools and techniques from other disciplines—ranging from machine learning to economics to program analysis—whose principles might provide insights into a new, previously undiscovered solution. I then devise a new approach or solution, and I transfer it to practice through implementation and deployment of real-world systems.

My work in this broad area follows three specific themes: (1) making the network more secure; (2) improving network availability and performance by making the network easier to operate and manage; (3) designing platforms for virtual networks that facilitate technical innovation in both network security and operations. The first two themes involve developing solutions that make the network more robust and resilient in the face of faults, misconfiguration, and malice; the third theme provides an avenue to evaluate and deploy these solutions in practice.

Theme 1: Network Security

Summary of work. My research explores the role that communications networks—in particular the network layer—can play in improving computer and communications security. This line of research began with my arrival at Georgia Tech in 2006. A cornerstone of this research is a system that was published in August 2009 called "SNARE" (Spatio-temporal Network-level Automated Reputation Engine). This work appeared at the *USENIX Security Symposium*, a top-tier security conference. The main idea behind

SNARE—and the key insight behind my research in spam filtering—is that spammers have different sending behavior than legitimate senders. Therefore, filters can distinguish spammers from legitimate senders by examining their *sending behavior* (i.e., how they send traffic), rather than what is in the messages themselves.

Prior to my research, conventional spam filters attempted to distinguish spam from legitimate email by looking at message contents: that is, they would look at the words or language used in the messages themselves and try to detect spam based on what the message said. This approach has become increasingly untenable, since spammers have begun to embed their messages in all sorts of media, ranging from images to PDFs to audio files to spreadsheets—by the time developers perfected their content filters for one type of medium, spammers moved onto the next. My line of work has taken an entirely different, but complementary approach: I look at features of the senders' *behavior* (e.g., the time of day they are sending, whether there are other "nearby" senders on the network, whether and how the sizes of the messages of the senders vary over time) to distinguish spamming behavior from legitimate email use. The method is harder for spammers to evade, it is more flexible because it can be deployed anywhere in the network, and it can work at much higher traffic rates than conventional approaches. This idea was first laid out in the initial award paper at *SIGCOMM* and finally realized in the SNARE paper from August 2009 at *USENIX Security*.

I have also worked on sweeping changes to the Internet architecture that could improve *accountability*, thus making it more difficult for malicious parties to operate unfettered in the first place. The current Internet architecture provides little to no accountability. Malicious end systems can conceal the source of their traffic ("spoofing"), and edge networks can provide false information about their reachability to various Internet destinations ("route hijacking"); both of these attacks make it difficult to track down perpetrators of attacks. Current approaches to solving these problems require manual configuration and operator vigilance, which make them weak and error-prone. Towards building networks that are inherently accountable, I have developed the Accountable Internet Protocol (AIP). One of my contributions to the design was to make the addresses in this protocol self-certifying, which forms the cornerstone of the basic design. I also demonstrated how to apply AIP to secure BGP, the Internet's interdomain routing protocol.

Impact. My research in network security has had impact in research, in industry, and on the national level. My research on this topic has earned the Presidential Early Career Award for Scientists and Engineers (PECASE), a Sloan fellowship, and the Best Paper Award at *ACM SIGCOMM* (the premier computer networking conference). Aspects of my work have also been incorporated into commercial spam filtering products and Web mail clients at companies including Yahoo, Cisco/Ironport, and McAfee, as well as a project for the Department of Defense on high-speed network monitoring. My paper on understanding the network-level behavior of spammers—which won the Best Student Paper award at *SIGCOMM* in 2006—has been cited over 200 times since its initial publication in August 2006—it spawned a variety of high-impact follow-on work, including looking at network-level behavior not only to develop better spam filters, but also to detect botnets more effectively and defend against phishing attacks, click fraud, and other serious threats to the Internet infrastructure. I have also been working on similar approaches to help detect and dismantle the Internet's scam hosting infrastructure (e.g., Web sites that attempt to steal user passwords, money, and so forth). My initial paper on this topic ("Dynamics of Online Scam-Hosting Infrastructure") won the Best Paper award at the *Passive and Active Measurement* conference in April 2009.

My work on SNARE has also garnered significant attention in industry. This work was featured in *Technology Review* and on Slashdot (a popular, high-traffic site for news in information technology). Several companies including Yahoo have incorporated the network-level features that SNARE identifies into its spam filters, and companies that develop spam filtering appliances, such as McAfee, are also using these features to improve the accuracy and performance of their spam filtering appliances.

AIP appeared in *ACM SIGCOMM* in 2008; an early version of the design also appeared in *ACM Workshop on Hot Topics in Networking (HotNets)*. I am incorporating a version of this technology into a working system and transferring them to practice. I am working with BBN on a DARPA project that will ultimately result in incorporating AIP's mechanisms into a military network protocol that allows attribution of traffic to sources

(the details may ultimately be classified).

My impact on the broader field of cybersecurity goes beyond my own research. I am also having impact in the national arena in several ways. Last year, I was involved in setting the nation's agenda for cyber security, through multiple additional activities. First, with leadership from Jeannette Wing at NSF, I led a community-wide effort to develop a "wish list" document that describes the security community's needs for access to better data—ranging from network traffic data, to data about our country's infrastructure. This report was ultimately delivered to Tom Kalil, the deputy director for policy in the Office of Science and Technology Policy. Second, with program managers Karl Levitt and Lenore Zuck at NSF, I organized a community-wide, multi-agency workshop on "Security-Driven Architectures". The workshop included participants from all areas of computer science (ranging from theory and cryptography to networking to systems to hardware), with an eye towards setting a research agenda for developing more holistic approaches to computer security that consider *all* aspects of computer and communications systems, rather than just a single piece (like the network). Finally, my work on developing next-generation Internet protocols to improve accountability (which could eradicate spam in the first place), based on work that appeared at *ACM SIGCOMM* in 2008, were also included in reports for the National Cyber Leap Year.

Representative Publication

> S. Hao, N. Syed, N. Feamster, A. Gray and S. Krasser. "Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine". *USENIX Security Symposium*, August 2009. Montreal, Quebec, Canada.

Most Cited Publication (236 Citations)

> A. Ramachandran and N. Feamster. "Understanding the Network-Level Behavior of Spammers". *ACM SIGCOMM*, August 2006. Pisa, Italy. **Best Student Paper Award.**

Theme 2: Network Operations

Summary of work. A second major theme of my work is *network operations*, which is what I call the field of designing networks so that they are easier to run and manage. Much of my work in this area has focused on fault detection and troubleshooting. Prior to my dissertation work, operators relied on detecting problems with networks "at runtime" on a live network. My dissertation work demonstrated that, in fact, many routing problems could be detected simply by examining the configuration of the routing protocols, before the configuration is even deployed. I applied techniques from static program analysis to routing configuration to help network operators catch mistakes and predict dynamic network behavior before the configurations are deployed on a live network, preventing costly and catastrophic network downtime.

Beyond predicting behavior and proactively detecting configuration faults, operators must understand the network's behavior *as it is running* (e.g., to detect equipment failures, attacks, or unplanned shifts in network traffic). Unfortunately, operators are drowning in heterogeneous data. To help operators better understand network faults "at runtime", I have applied unsupervised learning techniques to Internet routing data to help them efficiently mine the data for network events that require corrective action. This work appeared in *ACM SIGMETRICS* in 2007. My work has also applied statistical inference techniques to help network operators determine the answers to "what if" configuration questions in content distribution networks; we developed a system called "WISE" (What-If Scenario Evaluator) to help network operators determine the effects of configuration changes on network response time. A paper on this system appeared at *ACM SIGCOMM* in 2008 and is now used by operators and network designers at Google. A more mature version of this work that also describes deployment experiences at Google is in submission to *IEEE/ACM Transactions on Networking*.

Users of communications networks also face the potential of intentional performance degradation or manipulation by Internet Service Providers (ISPs); these problems are popularly referred to as "network neutrality violations". This transparency can help users determine whether their network is the cause of performance degradation, or whether performance problems that they are seeing are due to some other cause. With students, I designed, built, and deployed the *Network Neutrality Access Observatory (NANO)*, a system that aggregates measurements from end systems to help users and operators of edge networks infer when transit networks may be discriminating against certain types of traffic. This work appeared in *ACM SIGCOMM CoNext* in 2009, and we have deployed the system on Google's Measurement Lab (`http://www.measurementlab.net/`). More recently, we have been looking at methods for helping users diagnose general problems with access network performance and examining which factors have the most influence on access network performance.

I have developed new network protocols and architectures that improve availability and accountability in communications networks in the face of both faults and malice. Networks face the continual threat of failures and attacks that disrupt end-to-end connectivity. Prior to my work, one promising approach to improving connectivity involved routing traffic along multiple paths between two endpoints ("multipath routing"); despite the promise of this approach, previous approaches encountered two significant challenges: First, previous approaches for disseminating information about multiple paths through the network did not scale to large networks. Second, end systems had no way to signal to the network that an end-to-end path had failed or was providing inadequate performance. My research applied a new perspective to this problem: rather than simply routing traffic on one of multiple paths to a destination, allow traffic to switch paths at intermediate points en route to the destination, and allow end systems to signal to the network when it should attempt to use a different path to the destination using a small number of bits that can be carried in the traffic itself. This system, called *path splicing*, provides up to an exponential improvement in reliability for only a linear increase in the amount of state that each router in the network must store.

Impact. The foundation of this research theme comes from a system I built called called "rcc" (router configuration checker). This system was the centerpiece of my doctoral dissertation and has had significant impact in both research and industry. The work received the Best Paper Award at *ACM/USENIX Networked Systems Design and Implementation (NSDI)* in 2005 and has been used by hundreds of Internet Service Providers (ISPs) around the world to check their network configurations for errors.

The NANO project is among the most visible of my research projects at Georgia Tech: The project page receives about 2,000 unique visitors every month, and, between January and March 2010, about 300 users have downloaded the code. The system is deployed on Google's Measurement Lab, and we are exploring a version of NANO that could be incorporated into the Google Toolbar with Google's Broadband Access Transparency (BAT) team.

The path splicing work resulted in a Sigma Xi undergraduate research award for Megan Elmore. The work was funded by Cisco, and they have considered the possibility of extending their existing multiple routing configuration (MRC) function to support path splicing. A more likely deployment scenario, however, may be the incorporation of path splicing into a network where network elements are more programmable (I discuss the promise of programmable networking in "Future Challenges" below.) We have published an open-source implementation of path splicing on several programmable networking platforms.

Representative Publication

M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar. "Answering What-if Deployment and Configuration Questions with 'WISE'". *ACM SIGCOMM*, August 2008. Seattle, WA.

Most Cited Publication (137 Citations)

> N. Feamster and H. Balakrishnan "Detecting BGP Configuration Faults with Static Analysis" *2nd Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005. **Best Paper Award.**

Theme 3: Virtual Networking

Network virtualization allows multiple networks to operate in parallel on the same physical infrastructure. Although this concept is not new (commonly used Virtual Private Networks, or "VPNs", come to mind as a prominent real-world example of network virtualization), virtualizing all aspects of the network infrastructure—in particular, both the links *and* the routers themselves—holds great promise for enabling innovation. In 2002, Larry Peterson, Scott Shenker, and Jon Turner argued that networking research had "ossified", because researchers faced a huge deployment hurdle for deploying their research in production environments, and also because the large stakeholders had little incentive to allow disruptive innovation to take place. Their argument was essentially that, by "letting a thousand flowers bloom", multiple networking technologies could be deployed in parallel, thereby providing researchers a path to innovation. The main research challenge was how to design and implement a virtual network infrastructure that supported this philosophy.

Towards solving this challenge, I began working on network virtualization during my postdoc at Princeton. Jennifer Rexford and I wanted to implement a new network protocol we had designed at the end of my graduate career. Our plan was to use PlanetLab—a large testbed with virtualized servers distributed around the world—to do it. Unfortunately, we quickly realized that PlanetLab did not have the necessary functions to instantiate test *networks*; in particular, PlanetLab offered no functions for building virtual routers and links, and also had no support for forwarding traffic at high rates for virtual routers (e.g., every packet needed to be copied several times at each node, significantly slowing the packet forwarding rates). These shortcomings caused us to pursue a larger project to build such a testbed that would support the kinds of experiments that we wanted to run. With Andy Bavier and Larry Peterson, we built a Virtual Network Infrastructure (VINI), a testbed that allows researchers to build virtual networks. This work appeared in *ACM SIGCOMM* in 2006. Although we still strive for more widespread adoption, the testbed is regularly used by several research groups around the country.

Since this initial work, I have focused on two aspects of network virtualization: (1) providing Internet connectivity and routing control to virtual networks; (2) designing very fast packet forwarding technologies for virtual networks. A virtual network—either an experiment or a distributed "cloud" service—typically needs connectivity to the rest of the Internet so that users can actually exchange traffic with it. To provide such connectivity, and to give each virtual network direct control over how user traffic reaches it, I designed, implemented and deployed the Transit Portal. This work will appear in *USENIX Annual Technical Conference* in June 2010; it is also a cornerstone of the larger nationwide GENI effort (featured here, for example: `http://www.geni.net/?p=1682`). Our work on designing faster packet forwarding technologies for virtual networks started with the Trellis project, which moved packet forwarding for virtual networks into the kernel; although this work resulted only in a workshop publication, the software itself was adopted by University of Utah's Emulab, the most prominent emulation-based testbed for networking research. Our current efforts have focused on accelerating packet forwarding further by supporting custom packet forwarding for virtual networks in Field Programmable Gate Arrays (FPGAs); our work on SwitchBlade, a platform for rapidly developing and deploying custom forwarding engines in hardware for virtual networks, will appear at *ACM SIGCOMM* in August 2010.

Impact. The impact of this work thus far has been to support network experimentation for researchers; many other virtual network technologies and platforms have built on this work. Our work on virtual

networks has been cited nearly 300 times (the VINI paper has been cited 200 times, and our work describing a network architecture based around network virtualization has been cited over 100 times).

The Transit Portal is currently deployed in five locations, and I am using it in my courses to provide students with hands-on experience configuring networks of routers and connecting them to real BGP-speaking routers on the Internet. The course I have developed that uses this technology is likely serves as the first course where students can directly configure networks of routers that are connected to the global Internet.

Representative Publication

> B. Anwer, M. Tariq, M. Motiwala, N. Feamster. "SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware" *ACM SIGCOMM*, New Delhi, India, August 2010.

Most Cited Publication (200 citations)

> A. Bavier, N. Feamster, M. Huang, L. Peterson, J. Rexford. "In VINI Veritas: Realisitic and Controlled Network Experimentation". *ACM SIGCOMM*, August 2006. Pisa, Italy.

Future Challenges

In my future work, I plan to take a proactive approach to network security and operations by developing protocols from the ground up that make networks fundamentally more secure and easier to manage. Networks are large, distributed systems that operators "program" with network configuration. Unfortunately, this configuration still takes place at the granularity of individual network devices and relies on low-level, mechanistic, and unintuitive commands. Improving network security and reliability ultimately requires raising this level of abstraction, which will most likely require developing protocols and languages that allow network operators to configure networks from a much higher layer of abstraction than they do today. It also entails fundamental changes to both the network devices and protocols. To achieve this goal, I see opportunities to draw on techniques from other disciplines. For example, my most recently awarded NSF grant proposes applying programming language and software engineering techniques to help operators configure networks.

Along these lines, my long-term goal is to *design networks that permit a range of policies but still have provable correctness and security properties*. In recent years, I have been redesigning network protocols so that operators can more easily control which users have access to which parts of the network ("access control"); I have also been building host and network support to prevent unauthorized access and data leaks ("information flow control"). My goal is to make it easier for operators to express—and verify—these policies in an operational network. I have made some early progress on these problems: we have built a system called Resonance that allows operators to express access control policies at a higher level of abstraction and may ultimately replace the current campus access control system. (A working version of this system was demonstrated at the GENI Engineering Conference in March 2010.) We are also building a system called Pedigree, which allows network operators to prevent certain types of data leaks. Finally, I have begun to design new protocols and systems to make home networks easier to configure and more secure.

My continued efforts will not only help advance network operations, but also to help shape the future of networking research towards more "hands on", operationally relevant problems. My approach will not only define new areas for networking research, but will also take better advantage of Georgia Tech's unique facilities and research-minded network operators. I aim to make Georgia Tech the top destination for performing experimental, systems-focused research on network operations and security.

Suggested Letter Writers

Nick Feamster

1. Jennifer Rexford (Princeton), Full Professor (postdoc mentor and regular collaborator)
2. Scott Shenker (UC Berkeley), Full Professor (occasional collaborator)
3. Larry Peterson (Princeton), Full Professor (previous collaborator)
4. Farnam Jahanian (Michigan), Full Professor
5. Jim Kurose (UMass Amherst), Full Professor
6. David Clark (MIT), Senior Research Scientist
7. Nick McKeown (Stanford), Associate Professor
8. Vern Paxson (UC Berkeley), Associate Professor
9. Stefan Savage (UCSD), Associate Professor
10. David Wetherall (Univ. of Washington), Associate Professor

Representative Publications

Nick Feamster

1. S. Hao, N. Syed, N. Feamster, A. Gray and S. Krasser. "Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine". *USENIX Security Symposium*, August 2009. Montreal, Quebec, Canada.

2. M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar. "Answering What-if Deployment and Configuration Questions with 'WISE'". *ACM SIGCOMM*, August 2008. Seattle, WA.

   Note: A more recent version of this paper is submission to *IEEE/ACM Transactions on Networking*; it was submitted in January 2010. A version of the *Transactions on Networking* submission is available at `http://www.cc.gatech.edu/~feamster/rpt/papers/wise:ton10.pdf`

3. B. Anwer, M. Tariq, M. Motiwala, N. Feamster. "SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware" *ACM SIGCOMM*, New Delhi, India, August 2010.

Proceedings of 18th USENIX Security Symposium
Montreal, Quebec, Canada. August 2009.

Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine

Shuang Hao, Nadeem Ahmed Syed, Nick Feamster, Alexander G. Gray, Sven Krasser *
College of Computing, Georgia Tech *McAfee, Inc.
{shao, nadeem, feamster, agray}@cc.gatech.edu, sven_krasser@mcafee.com

Abstract

Users and network administrators need ways to filter email messages based primarily on the reputation of the sender. Unfortunately, conventional mechanisms for sender reputation—notably, IP blacklists—are cumbersome to maintain and evadable. This paper investigates ways to infer the reputation of an email sender based solely on network-level features, without looking at the contents of a message. First, we study first-order properties of network-level features that may help distinguish spammers from legitimate senders. We examine features that can be ascertained without ever looking at a packet's contents, such as the distance in IP space to other email senders or the geographic distance between sender and receiver. We derive features that are *lightweight*, since they do not require seeing a large amount of email from a single IP address and can be gleaned without looking at an email's contents—many such features are apparent from even a single packet. Second, we incorporate these features into a classification algorithm and evaluate the classifier's ability to automatically classify email senders as spammers or legitimate senders. We build an *automated* reputation engine, *SNARE*, based on these features using labeled data from a deployed commercial spam-filtering system. We demonstrate that *SNARE* can achieve comparable accuracy to existing static IP blacklists: about a 70% detection rate for less than a 0.3% false positive rate. Third, we show how *SNARE* can be integrated into existing blacklists, essentially as a first-pass filter.

1 Introduction

Spam filtering systems use two mechanisms to filter spam: content filters, which classify messages based on the contents of a message; and sender reputation, which maintains information about the IP address of a sender as an input to filtering. Content filters (e.g., [22, 23]) can block certain types of unwanted email messages, but they can be brittle and evadable, and they require analyzing the contents of email messages, which can be expensive. Hence, spam filters also rely on *sender reputation* to filter messages; the idea is that a mail server may be able to reject a message purely based on the reputation of the sender, rather than the message contents. DNS-based blacklists (DNSBLs) such as Spamhaus [7] maintain lists of IP addresses that are known to send spam. Unfortunately, these blacklists can be both incomplete and slow-to-respond to new spammers [32]. This unresponsiveness will only become more serious as both botnets and BGP route hijacking make it easier for spammers to dynamically obtain new, unlisted IP addresses [33, 34]. Indeed, network administrators are still searching for spam-filtering mechanisms that are both *lightweight* (i.e., they do not require detailed message or content analysis) and *automated* (i.e., they do not require manual update, inspection, or verification).

Towards this goal, this paper presents *SNARE* (Spatio-temporal Network-level Automatic Reputation Engine), a sender reputation engine that can accurately and automatically classify email senders based on lightweight, network-level features that can be determined early in a sender's history—sometimes even upon seeing only a single packet. *SNARE* relies on the intuition that about 95% of all email is spam, and, of this, $75 - 95\%$ can be attributed to botnets, which often exhibit unusual sending patterns that differ from those of legitimate email senders. *SNARE* classifies senders based on *how* they are sending messages (i.e., traffic patterns), rather than *who* the senders are (i.e., their IP addresses). In other words, *SNARE* rests on the assumption that there are lightweight network-level features that can differentiate spammers from legitimate senders; this paper finds such features and uses them to build a system for automatically determining an email sender's reputation.

SNARE bears some similarity to other approaches that classify senders based on network-level behavior [12, 21,

24, 27, 34], but these approaches rely on inspecting the message contents, gathering information across a large number of recipients, or both. In contrast, *SNARE* is based on *lightweight* network-level features, which could allow it to scale better and also to operate on higher traffic rates. In addition, *SNARE* is *more accurate* than previous reputation systems that use network-level behavioral features to classify senders: for example, *SNARE*'s false positive rate is an order of magnitude less than that in our previous work [34] for a similar detection rate. It is the first reputation system that is both as accurate as existing static IP blacklists and automated to keep up with changing sender behavior.

Despite the advantages of automatically inferring sender reputation based on "network-level" features, a major hurdle remains: We must identify *which features* effectively and efficiently distinguish spammers from legitimate senders. Given the massive space of possible features, finding a collection of features that classifies senders with both low false positive and low false negative rates is challenging. This paper identifies thirteen such network-level features that require varying levels of information about senders' history.

Different features impose different levels of overhead. Thus, we begin by evaluating features that can be computed purely locally at the receiver, with no information from other receivers, no previous sending history, and no inspection of the message itself. We found several features that fall into this category are surprisingly effective for classifying senders, including: The AS of the sender, the geographic distance between the IP address of the sender and that of the receiver, the density of email senders in the surrounding IP address space, and the time of day the message was sent. We also looked at various aggregate statistics across messages and receivers (e.g., the mean and standard deviations of messages sent from a single IP address) and found that, while these features require slightly more computation and message overhead, they do help distinguish spammers from legitimate senders as well. After identifying these features, we analyze the relative importance of these features and incorporate them into an automated reputation engine, based on the *RuleFit* [19] ensemble learning algorithm.

In addition to presenting the first automated classifier based on network-level features, this paper presents several additional contributions. First, we presented a detailed study of various network-level characteristics of both spammers and legitimate senders, a detailed study of how well each feature distinguishes spammers from legitimate senders, and explanations of why these features are likely to exhibit differences between spammers and legitimate senders. Second, we use state-of-the-art ensemble learning techniques to build a classifier using these features. Our results show that *SNARE*'s performance is at least as good as static DNS-based blacklists, achieving a 70% detection rate for about a 0.2% false positive rate. Using features extracted from a single message and aggregates of these features provides slight improvements, and adding an AS "whitelist" of the ASes that host the most commonly misclassified senders reduces the false positive rate to 0.14%. This accuracy is roughly equivalent to that of existing static IP blacklists like SpamHaus [7]; the advantage, however, is that *SNARE* is *automated*, and it characterizes a sender based on its sending *behavior*, rather than its IP address, which may change due to dynamic addressing, newly compromised hosts, or route hijacks. Although *SNARE*'s performance is still not perfect, we believe that the benefits are clear: Unlike other email sender reputation systems, *SNARE* is both automated and lightweight enough to operate solely on network-level information. Third, we provide a deployment scenario for *SNARE*. Even if others do not deploy *SNARE*'s algorithms exactly as we have described, we believe that the collection of network-level features themselves may provide useful inputs to other commercial and open-source spam filtering appliances.

The rest of this paper is organized as follows. Section 2 presents background on existing sender reputation systems and a possible deployment scenario for *SNARE* and introduces the ensemble learning algorithm. Section 3 describes the network-level behavioral properties of email senders and measures first-order statistics related to these features concerning both spammers and legitimate senders. Section 4 evaluates *SNARE*'s performance using different feature subsets, ranging from those that can be determined from a single packet to those that require some amount of history. We investigate the potential to incorporate the classifier into a spam-filtering system in Section 5. Section 6 discusses evasion and other limitations, Section 7 describes related work, and Section 8 concludes.

2 Background

In this section, we provide background on existing sender reputation mechanisms, present motivation for improved sender reputation mechanisms (we survey other related work in Section 7), and describe a classification algorithm called *RuleFit* to build the reputation engine. We also describe McAfee's TrustedSource system, which is both the source of the data used for our analysis and a possible deployment scenario for *SNARE*.

2.1 Email Sender Reputation Systems

Today's spam filters look up IP addresses in DNS-based blacklists (DNSBLs) to determine whether an IP address is a known source of spam at the time

of lookup. One commonly used public blacklist is Spamhaus [7]; other blacklist operators include SpamCop [6] and SORBS [5]. Current blacklists have three main shortcomings. First, they only provide reputation at the granularity of IP addresses. Unfortunately, as our earlier work observed [34], IP addresses of senders are dynamic: roughly 10% of spam senders on any given day have not been previously observed. This study also observed that many spamming IP addresses will go inactive for several weeks, presumably until they are removed from IP blacklists. This dynamism makes maintaining responsive IP blacklists a manual, tedious, and inaccurate process; they are also often coarse-grained, blacklisting entire prefixes—sometimes too aggressively—rather than individual senders. Second, IP blacklists are typically incomplete: A previous study has noted that as much as 20% of spam received at spam traps is not listed in any blacklists [33]. Finally, they are sometimes inaccurate: Anecdotal evidence is rife with stories of IP addresses of legitimate mail servers being incorrectly blacklisted (e.g., because they were reflecting spam to mailing lists). To account for these shortcomings, commercial reputation systems typically incorporate additional data such as SMTP metadata or message fingerprints to mitigate these shortcomings [11]. Our previous work introduced "behavioral blacklisting" and developed a spam classifier based on a single behavioral feature: the number of messages that a particular IP address sends to each recipient domain [34]. This paper builds on the main theme of behavioral blacklisting by finding better features that can classify senders earlier and are more resistant to evasion.

2.2 Data and Deployment Scenario

This section describes McAfee's TrustedSource email sender reputation system. We describe how we use the data from this system to study the network-level features of email senders and to evaluate *SNARE*'s classification. We also describe how *SNARE*'s features and classification algorithms could be incorporated into a real-time sender reputation system such as TrustedSource.

Data source TrustedSource is a commercial reputation system that allows lookups on various Internet identifiers such as IP addresses, URLs, domains, or message fingerprints. It receives query feedback from various different device types such as mail gateways, Web gateways, and firewalls. We evaluated *SNARE* using the query logs from McAfee's TrustedSource system over a fourteen-day period from October 22–November 4, 2007. Each received email generates a lookup to the TrustedSource database, so each entry in the query log represents a single email that was sent from some sender to one of McAfee's TrustedSource appliances. Due to the volume

Field	*Description*
timestamp	UNIX timestamp
ts_server_name	Name of server that handles the query
score	Score for the message based on a combination of anti-spam filters
source_ip	Source IP in the packet (DNS server relaying the query to us)
query_ip	The IP being queried
body_length	Length of message body
count_taddr	Number of To-addresses

Figure 1: Description of data used from the McAfee dataset.

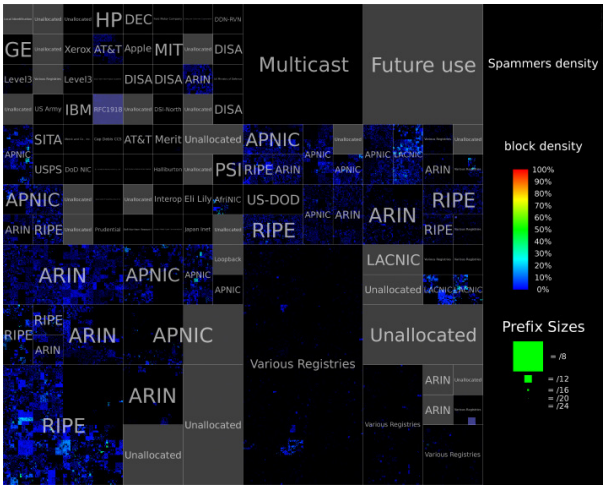


Figure 2: Distribution of senders' IP addresses in Hilbert space for the one-week period (October 22–28, 2007) of our feature study. (The grey blocks are unused IP space.)

of the full set of logs, we focused on logs from a single TrustedSource server, which reflects about 25 million email messages as received from over 1.3 million IP addresses each day. These messages were reported from approximately 2,500 distinct TrustedSource appliances geographically distributed around the world. While there is not a precise one-to-one mapping between domains and appliances, and we do not have a precise count for the number of unique domains, the number of domains is roughly of the same order of magnitude.

The logs contain many fields with *metadata for each email message*; Figure 1 shows a subset of the fields that we ultimately use to develop and evaluate *SNARE*'s classification algorithms. The `timestamp` field reflects the time at which the message was received at a TrustedSource appliance in some domain; the `source_ip` field reflects the source IP of the machine that issued the DNS query (i.e., the recipient of the email). The `query_ip`

field is the IP address being queried (i.e., the IP address of the email sender). The IP addresses of the senders are shown in the Hilbert space, as in Figure 2[1], where each pixel represents a /24 network prefix and the intensity indicates the observed IP density in each block. The distribution of the senders' IP addresses shows that the TrustedSource database collocated a representative set of email across the Internet. We use many of the other features in Figure 1 as input to *SNARE*'s classification algorithms.

To help us label senders as either spammers or legitimate senders for both our feature analysis (Section 3) and training (Sections 2.3 and 4), the logs also contain *scores* for each email message that indicate how McAfee scored the email sender based on its current system. The `score` field indicates McAfee's sender reputation score, which we stratify into five labels: certain ham, likely ham, certain spam, likely ham, and uncertain. Although these scores are not perfect ground truth, they do represent the output of both manual classification and continually tuned algorithms that also operate on more heavy-weight features (e.g., packet payloads). Our goal is to develop a fully automated classifier that is as accurate as TrustedSource but (1) classifies senders *automatically* and (2) relies only on lightweight, evasion-resistant network-level features.

Deployment and data aggregation scenario Because it operates only on network-level features of email messages, *SNARE* could be deployed either as part of TrustedSource or as a standalone DNSBL. Some of the features that *SNARE* uses rely on aggregating sender behavior across a wide variety of senders. To aggregate these features, a monitor could collect information about the global behavior of a sender across a wide variety of recipient domains. Aggregating this information is a reasonably lightweight operation: Since the features that *SNARE* uses are based on simple features (i.e., the IP address, plus auxiliary information), they can be piggybacked in small control messages or in DNS messages (as with McAfee's TrustedSource deployment).

2.3 Supervised Learning: RuleFit

Ensemble learning: *RuleFit* Learning ensembles have been among the popular predictive learning methods over the last decade. Their structural model takes the form

$$F(\mathbf{x}) = a_0 + \sum_{m=1}^{M} a_m f_m(\mathbf{x}) \qquad (1)$$

Where $\mathbf{x}$ are input variables derived form the training data (spatio-temporal features); $f_m(\mathbf{x})$ are different functions called ensemble members ("base learner") and M is the size of the ensemble; and $F(\mathbf{x})$ is the predictive output (labels for "spam" or "ham"), which takes a linear combination of ensemble members. Given the base learners, the technique determines the parameters for the learners by regularized linear regression with a "lasso" penalty (to penalize large coefficients a_m).

Friedman and Popescu proposed *RuleFit* [19] to construct regression and classification problems as linear combinations of simple rules. Because the number of base learners in this case can be large, the authors propose using the rules in a decision tree as the base learners. Further, to improve the accuracy, the variables themselves are also included as basis functions. Moreover, fast algorithms for minimizing the loss function [18] and the strategy to control the tree size can greatly reduce the computational complexity.

Variable importance Another advantage of *RuleFit* is the interpretation. Because of its simple form, each rule is easy to understand. The relative importance of the respective variables can be assessed after the predictive model is built. Input variables that frequently appear in important rules or basic functions are deemed more relevant. The importance of a variable x_i is given as importance of the basis functions that correspond directly to the variable, plus the average importance of all the other rules that involve x_i. The *RuleFit* paper has more details [19]. In Section 4.3, we show the relative importance of these features.

Comparison to other algorithms There exist two other classic classifier candidates, both of which we tested on our dataset and both of which yielded poorer performance (i.e., higher false positive and lower detection rates) than *RuleFit*. Support Vector Machine (SVM) [15] has been shown empirically to give good generalization performance on a wide variety of problems such as handwriting recognition, face detection, text categorization, etc. On the other hand, they do require significant parameter tuning before the best performance can be obtained. If the training set is large, the classifier itself can take up a lot of storage space and classifying new data points will be correspondingly slower since the classification cost is $O(S)$ for each test point, where S is the number of support vectors. The computational complexity of SVM conflicts with *SNARE*'s goal to make decision quickly (at line rate). Decision trees [30] are another type of popular classification method. The resulting classifier is simple to understand and faster, with the prediction on a new test point taking $O(\log(N))$, where N is the number of nodes in the trained tree. Unfortunately, decision trees compromise accuracy: its high false positive rates make it less than ideal for our purpose.

[1] A larger figure is available at `http://www.gtnoise.net/snare/hilbert-ip.png`.

3 Network-level Features

In this section, we explore various spatio-temporal features of email senders and discuss why these properties are relevant and useful for differentiating spammers from legitimate senders. We categorize the features we analyze by increasing level of overhead:

- *Single-packet features* are those that can be determined with no previous history from the IP address that *SNARE* is trying to classify, and given only a *single packet* from the IP address in question (Section 3.1).
- *Single-header and single-message features* can be gleaned from a single SMTP message header or email message (Section 3.2).
- *Aggregate features* can be computed with varying amounts of history (i.e., aggregates of other features) (Section 3.3).

Each class of features contains those that may be either purely local to a single receiver or aggregated across multiple receivers; the latter implies that the reputation system must have some mechanism for aggregating features in the network. In the following sections, we describe features in each of these classes, explain the intuition behind selecting that feature, and compare the feature in terms of spammers vs. legitimate senders.

No single feature needs to be perfectly discriminative between ham and spam. The analysis below shows that it is unrealistic to have a single perfect feature to make optimal resolution. As we describe in Section 2.3, *SNARE*'s classification algorithm uses a *combination* of these features to build the best classifier. We do, however, evaluate *SNARE*'s classifier using these three different classes of features to see how well it can perform using these different classes. Specifically, we evaluate how well *SNARE*'s classification works using only single-packet features to determine how well such a lightweight classifier would perform; we then see whether using additional features improves classification.

3.1 Single-Packet Features

In this section, we discuss some properties for identifying a spammer that rely only on a single packet from the sender IP address. In some cases, we also rely on auxiliary information, such as routing table information, sending history from neighboring IP addresses, etc., not solely information in the packet itself. We first discuss the features that can be extracted from just a single IP packet: the geodesic distance between the sender and receiver, sender neighborhood density, probability ratio of spam to ham at the time-of-day the IP packet arrives, AS number of the sender and the status of open ports on the machine that sent the email. The analysis is based on the McAfee's data from October 22–28, 2007 inclusive (7 days).[2]

3.1.1 Sender-receiver geodesic distance: Spam travels further

Recent studies suggest that social structure between communicating parties could be used to effectively isolate spammers [13, 20]. Based on the findings in these studies, we hypothesized that legitimate emails tend to travel shorter geographic distances, whereas the distance traveled by spam will be closer to random. In other words, a spam message may be just as likely to travel a short distance as across the world.

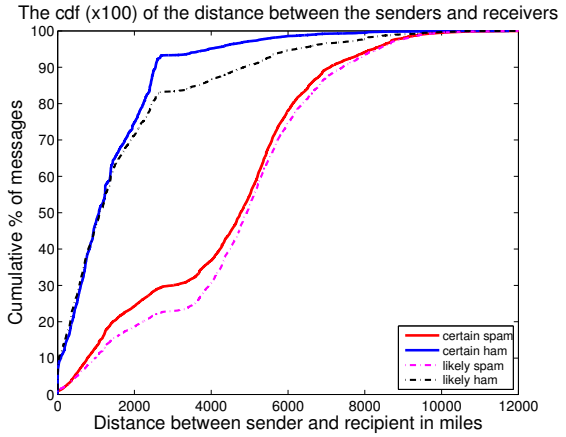
Figure 3(a) shows that our intuition is roughly correct: the distribution of the distance between the sender and the target IP addresses for each of the four categories of messages. The distance used in these plots is the geodesic distance, that is, the distance along the surface of the earth. It is computed by first finding the physical latitude and longitude of the source and target IP using the MaxMind's GeoIP database [8] and then computing the distance between these two points. These distance calculations assume that the earth is a perfect sphere. For *certain ham*, 90% of the messages travel about 2,500 miles or less. On the other hand, for *certain spam*, only 28% of messages stay within this range. In fact, about 10% of spam travels more than 7,000 miles, which is a quarter of the earth's circumference at the equator. These results indicate that geodesic distance is a promising metric for distinguishing spam from ham, which is also encouraging, since it can be computed quickly using just a single IP packet.

3.1.2 Sender IP neighborhood density: Spammers are surrounded by other spammers

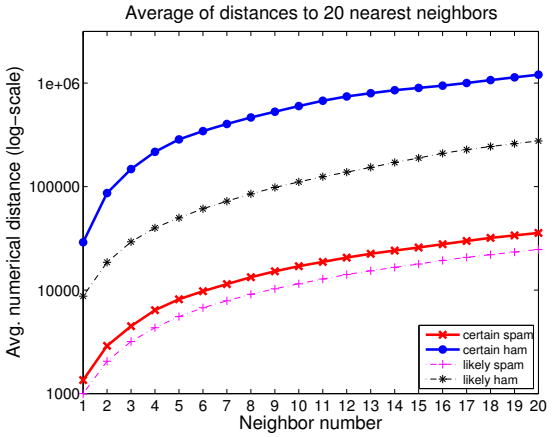
Most spam messages today are generated by botnets [33, 37]. For messages originating from the same botnet, the infected IP addresses may all lie close to one another in numerical space, often even within the same subnet. One way to detect whether an IP address belongs to a botnet is to look at the past history and determine if messages have been received from other IPs in the same subnet as the current sender, where the subnet size can be determined experimentally. If many different IPs from the same subnet are sending email, the likelihood that the whole subnet is infested with bots is high.

The problem with simply using subnet density is that the frame of reference does not transcend the subnet

[2]The evaluation in Section 4 uses the data from October 22–November 4, 2007 (14 days), some of which are not included in the data trace used for measurement study.



(a) Geodesic distance between the sender and recipient's geographic location.



(b) Average of numerical distances to the 20 nearest neighbors in the IP space.

Figure 3: Spatial differences between spammers and legitimate senders.

boundaries. A more flexible measure of *email sender density* in an IP's neighborhood is the distances to its k nearest neighbors. The distance to the k nearest neighbors can be computed by treating the IPs as set of numbers from 0 to $2^{32} - 1$ (for IPv4) and finding the nearest neighbors in this single dimensional space. We can expect these distances to exhibit different patterns for spam and ham. If the neighborhood is *crowded*, these neighbor distances will be small, indicating the possible presence of a botnet. In normal circumstances, it would be unusual to see a large number of IP addresses sending email in a small IP address space range (one exception might be a cluster of outbound mail servers, so choosing a proper threshold is important, and an operator may need to evaluate which threshold works best on the specific network where *SNARE* is running).

The average distances to the 20 nearest neighbors of the senders are shown in Figure 3(b). The x-axis indicates how many nearest neighbors we consider in IP space, and the y-axis shows the average distance in the sample to that many neighbors. The figure reflects the fact that a large majority of spam originates from hosts have high email sender density in a given IP region. The distance to the k^{th} nearest neighbor for spam tends to be much shorter on average than it is for legitimate senders, indicating that spammers generally reside in areas with higher densities of email senders (in terms of IP address space).

3.1.3 Time-of-day: Spammers send messages according to machine off/on patterns

Another feature that can be extracted using information from a single packet is the time of day when the message was sent. We use the *local* time of day at the sender's physical location, as opposed to Coordinated Universal Time (UTC). The intuition behind this feature is that local legitimate email sending patterns may more closely track "conventional" diurnal patterns, as opposed to spam sending patterns.

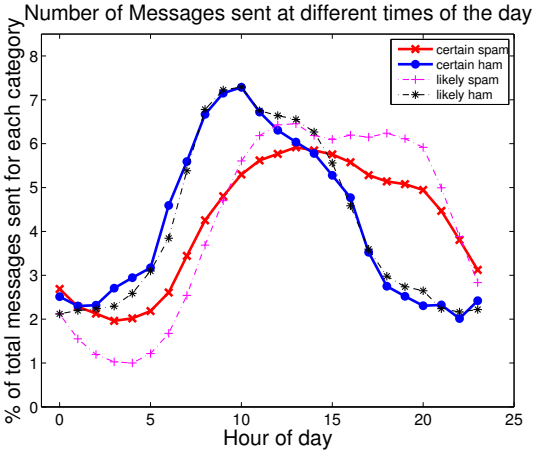


Figure 4: Differences in diurnal sending patterns of spammers and legitimate senders.

Figure 4 shows the relative percentage of messages of each type at different times of the day. The legitimate senders and the spam senders show different diurnal patterns. Two times of day are particularly striking: the relative amount of ham tends to ramp up quickly at the start of the workday and peaks in the early morning. Volumes decrease relatively quickly as well at the end of the workday. On the other hand spam increases at a slower, steadier pace, probably as machines are switched on in the morning. The spam volume stays steady throughout the day and starts dropping around 9:00 p.m., probably when machines are switched off again. In summary, legitimate

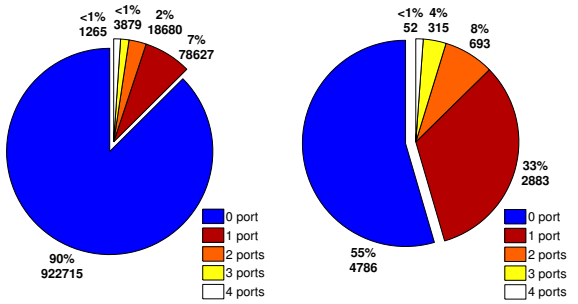
senders tend to follow workday cycles, and spammers tend to follow machine power cycles.

To use the timestamp as a feature, we compute the probability ratio of spam to ham at the time of the day when the message is received. First, we compute the *a priori* spam probability $p_{s,t}$ during some hour of the day t, as $p_{s,t} = n_{s,t}/n_s$, where $n_{s,t}$ is the number of spam messages received in hour t, and n_s is the number of spam messages received over the entire day. We can compute the *a priori* ham probability for some hour t, $p_{h,t}$ in a similar fashion. The probability ratio, r_t is then simply $p_{s,t}/p_{h,t}$. When a new message is received, the precomputed spam to ham probability ratio for the corresponding hour of the day at the senders timezone, r_t can be used as a feature; this ratio can be recomputed on a daily basis.

3.1.4 AS number of sender: A small number of ASes send a large fraction of spam

As previously mentioned, using IP addresses to identify spammers has become less effective for several reasons. First, IP addresses of senders are often transient. The compromised machines could be from dial-up users, which depend on dynamic IP assignment. If spam comes form mobile devices (like laptops), the IP addresses will be changed once the people carry the devices to a different place. In addition, spammers have been known to adopt stealthy spamming strategies where each bot only sends several spam to a single target domain, but overall the botnets can launch a huge amount of spam to many domains [33]. The low emission-rate and distributed attack requires to share information across domains for detection.

On the other hand, our previous study revealed that a significant portion of spammers come from a relatively small collection of ASes [33]. More importantly, the ASes responsible for spam differ from those that send legitimate email. As a result, the AS numbers of email senders could be a promising feature for evaluating the senders' reputation. Over the course of the seven days in our trace, more than 10% of unique spamming IPs (those sending certain spam) originated from only 3 ASes; the top 20 ASes host 42% of spamming IPs. Although our previous work noticed that a small number of ASes originated a large fraction of spam [33], we believe that this is the first work to suggest using the AS number of the email sender as input to an automated classifier for sender reputation.



(a) spam (b) legitimate mail

Figure 5: Distribution of number of open ports on hosts sending spam and legitimate mail.

3.1.5 Status of service ports: Legitimate mail tends to originate from machines with open ports

We hypothesized that legitimate mail senders may also listen on other ports besides the SMTP port, while bots might not; our intuition is that the bots usually send spam directly to the victim domain's mail servers, while the legitimate email is handed over from other domains' MSA (Mail Submission Agent). The techniques of reverse DNS (rDNS) and Forward Confirmed Reverse DNS (FCrDNS) have been widely used to check whether the email is from dial-up users or dynamically assigned addresses, and mail servers will refuse email from such sources [1].

We propose an additional feature that is orthogonal to DNSBL or rDNS checking. Outgoing mail servers open specific ports to accept users' connections, while the bots are compromised hosts, where the well-known service ports are closed (require root privilege to open). When packets reach the mail server, the server issues an active probe sent to the source host to scan the following four ports that are commonly used for outgoing mail service: 25 (SMTP), 465 (SSL SMTP), 80 (HTTP) and 443 (HTTPS), which are associated with outgoing mail services. Because neither the current mail servers nor the McAfee's data offer email senders' port information, we need to probe back sender's IP to check out what service ports might be open. The probe process was performed during both October 2008 and January 2009, well after the time when the email was received. Despite this delay, the status of open ports still exposes a striking difference between legitimate senders and spammers. Figure 5 shows the percentages and the numbers of opening ports for spam and ham categories respectively. The statistics are calculated on the senders' IPs from the evaluation dataset we used in Section 4 (October 22–28, 2007). In the spam case, 90% of spamming IP addresses have *none* of the standard mail service ports open; in contrast,

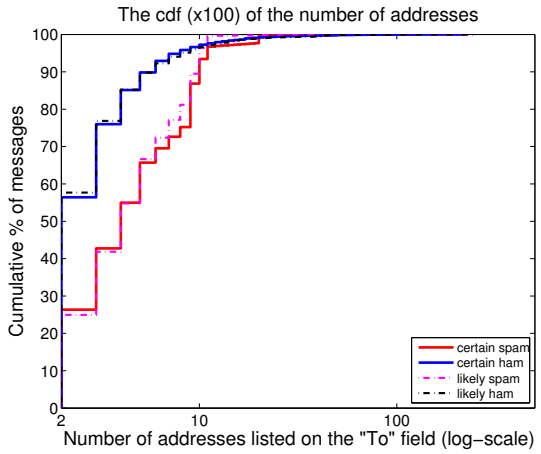


Figure 6: Distribution of number of addresses listed on the "To" field for each category (ignoring single-recipient messages).

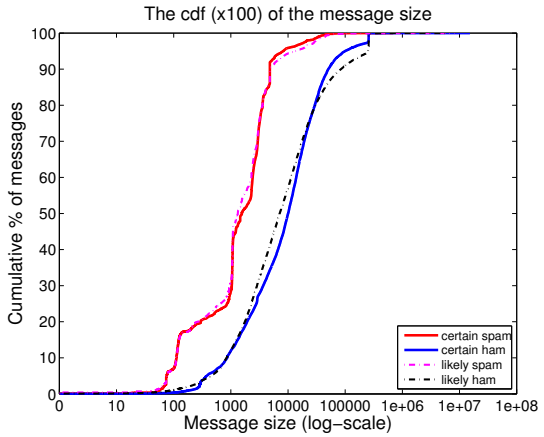


Figure 7: Distribution of message size (in bytes) for the different categories of messages.

half of the legitimate email comes from machines listening on at least one mail service port. Although firewalls might block the probing attempts (which causes the legitimate mail servers show no port listening), the status of the email-related ports still appears highly correlated with the distinction of the senders. When providing this feature as input to a classifier, we represent it as a bitmap (4 bits), where each bit indicates whether the sender IP is listening on a particular port.

3.2 Single-Header and Single-Message Features

In this section, we discuss other features that can be extracted from a single SMTP header or message: the number of recipients in the message, and the length of the message. We distinguish these features from those in the previous section, since extracting these features actually requires opening an SMTP connection, accepting the message, or both. Once a connection is accepted, and the SMTP header and subsequently, the compete message are received. At this point, a spam filter could extract additional non-content features.

3.2.1 Number of recipients: Spam tends to have more recipients

The features discussed so far can be extracted from a single IP packet from any given specific IP address combined with some historical knowledge of messages from other IPs. Another feature available without looking into the content is the number of address in "To" field of the header. This feature can be extracted after receiving the entire SMTP header but before accepting the message body. However, the majority of messages only have one address listed. Over 94% of spam and 96% of legitimate email is sent to a single recipient. Figure 6 shows the distribution of number of addresses in the "To" field for each category of messages for all emails that are sent to more than one recipient. The x-axis is on a log-scale to focus the plot on the smaller values. Based on this plot and looking at the actual values, it appears that if there are very large number of recipients on the "To" field (100 or more), there does not seem to be a significant difference between the different types of senders for this measure. The noticeable differences around 2 to 10 addresses show that, generally, ham has fewer recipients (close to 2) while spam is sent to multiple addresses (close to 10). (We acknowledge that this feature is probably evadable and discuss this in more detail in Section 6.1).

3.2.2 Message size: Legitimate mail has variable message size; spam tends to be small

Once an entire message has been received, the email body size in bytes is also known. Because a given spam sender will mostly send the same or similar content in all the messages, it can be expected that the variance in the size of messages sent by a spammer will be lower than among the messages sent by a legitimate sender. To stay effective, the spam bots also need to keep the message size small so that they can maximize the number of messages they can send out. As such the spam messages can be expected to be biased towards the smaller size. Figure 7 shows the distribution of messages for each category. The spam messages are all clustered in the 1–10KB range, whereas the distribution of message size for legitimate senders is more evenly distributed. Thus, the mes-

sage body size is another property of messages that may help differentiate spammers from legitimate senders.

3.3 Aggregate Features

The behavioral properties discussed so far can all be constructed using a single message (with auxiliary or neighborhood information). If some history from an IP is available, some *aggregate IP-level features* can also be constructed. Given information about multiple messages from a single IP address, the overall *distribution* of the following measures can be captured by using a combination of *mean and variance of*: (1) geodesic distance between the sender and recipient, (2) number of recipients in the "To" field of the SMTP header, and (3) message body length in bytes. By summarizing behavior over multiple messages and over time, these aggregate features may yield a more reliable prediction. On the flip side, computing these features comes at the cost of increased latency as we need to collect a number of messages before we compute these. Sometimes gathering aggregate information even requires cross-domain collaboration. By averaging over multiple messages, these features may also smooth the structure of the feature space, making marginal cases more difficult to classify.

4 Evaluating the Reputation Engine

In this section, we evaluate the performance of *SNARE*'s *RuleFit* classification algorithm using different sets of features: those just from a single packet, those from a single header or message, and aggregate features.

4.1 Setup

For this evaluation, we used fourteen days of data from the traces, from October 22, 2007 to November 4, 2007, part of which are different from the analysis data in Section 3. In other words, the entire data trace is divided into two parts: the first half is used for measurement study, and the latter half is used to evaluate *SNARE*'s performance. The purpose of this setup is both to verify the hypothesis that the feature statistics we discovered would stick to the same distribution over time and to ensure that feature extraction would not interfere with our evaluation of prediction.

Training We first collected the features for each message for a subset of the trace. We then randomly sampled 1 million messages from each day on average, where the volume ratio of spam to ham is the same as the original data (i.e., 5% ham and 95% spam; for now, we consider only messages in the "certain ham" and "certain spam" categories to obtain more accurate ground truth). Only our evaluation is based on this sampled dataset, *not* the feature analysis from Section 3, so the selection of those features should not have been affected by sampling. We then intentionally sampled equal amounts of spam as the ham data (30,000 messages in each categories for each day) to train the classifier because training requires that each class have an equal number of samples. In practice, spam volume is huge, and much spam might be discarded before entering the *SNARE* engine, so sampling on spam for training is reasonable.

Validation We evaluated the classifier using temporal cross-validation, which is done by splitting the dataset into subsets along the time sequence, training on the subset of the data in a time window, testing using the next subset, and moving the time window forward. This process is repeated ten times (testing on October 26, 2007 to November 4, 2007), with each subset accounting for one-day data and the time window set as 3 days (which indicates that long-period history is not required). For each round, we compute the detection rate and false positive rate respectively, where the detection rate (the "true positive" rate) is the ratio of spotted spam to the whole spam corpus, and false positive rate reflects the proportion of misclassified ham to all ham instances. The final evaluation reflects the average computed over all trials.

Summary Due to the high sampling rate that we used for this experiment, we repeated the above experiment for several trials to ensure that the results were consistent across trials. As the results in this section show, detection rates are approximately 70% and false positive rates are approximately 0.4%, even when the classifier is based only on single-packet features. The false positive drops to less 0.2% with the same 70% detection as the classifier incorporates additional features. Although this false positive rate is likely still too high for *SNARE* to subsume all other spam filtering techniques, we believe that the performance may be good enough to be used in conjunction with other methods, perhaps as an early-stage classifier, or as a substitute for conventional IP reputation systems (e.g., SpamHaus).

4.2 Accuracy of Reputation Engine

In this section, we evaluate *SNARE*'s accuracy on three different groups of features. Surprisingly, we find that, even relying on only single-packet features, *SNARE* can automatically distinguish spammers from legitimate senders. Adding additional features based on single-header or single-message, or aggregates of these features based on 24 hours of history, improves the accuracy further.

(a) Single Packet

	Classified as	
	Spam	Ham
Spam	**70%**	30%
Ham	**0.44%**	99.56%

(b) Single Header/Message

	Classified as	
	Spam	Ham
Spam	**70%**	30%
Ham	**0.29%**	99.71%

(c) 24+ Hour History

	Classified as	
	Spam	Ham
Spam	**70%**	30%
Ham	**0.20%**	99.80%

Table 1: *SNARE* performance using *RuleFit* on different sets of features using covariant shift. Detection and false positive rates are shown in bold. (The detection is fixed at 70% for comparison, in accordance with today's DNSBLs [10]).

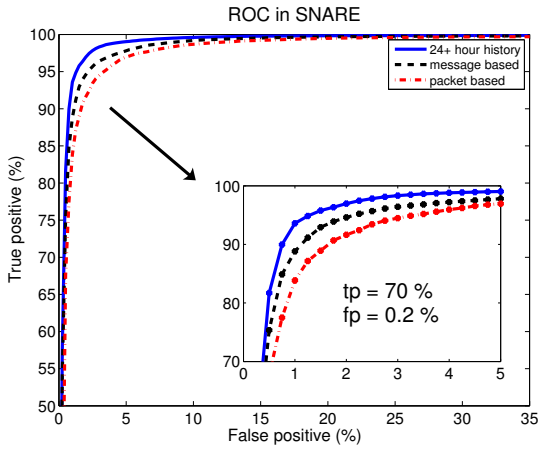


Figure 8: ROC in *SNARE*.

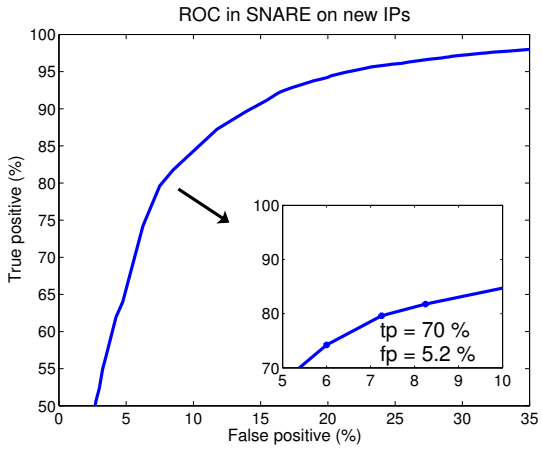


Figure 9: ROC on fresh IPs in *SNARE*.

4.2.1 Single-Packet Features

When a mail server receives a new connection request, the server can provide *SNARE* with the IP addresses of the sender and the recipient and the time-stamp based on the TCP SYN packet alone. Recall from Section 3 even if *SNARE* has never seen this IP address before, it can still combine this information with recent history of behavior of other email servers and construct the following features: (1) geodesic distance between the sender and the recipient, (2) average distance to the 20 nearest neighbors of the sender in the log, (3) probability ratio of spam to ham at the time the connection is requested (4) AS number of the sender's IP, and (5) status of the email-service ports on the sender.

To evaluate the effectiveness of these features, we trained *RuleFit* on these features. The dash-dot curve in Figure 8 demonstrate the ROC curve of *SNARE*'s reputation engine. The fp $= 0.2\%$ and tp $= 70\%$ statistics refer to the curve with 24-hour history (solid line), which will be addresses later. We check the false positive given a fixed true positive, 70%. The confusion matrix is shown in Table 1(a). Just over 0.44% of legitimate email gets labelled as spam. This result is significant because it relies on features constructed from a limited amount of data and just a single IP packet from the candidate IP. Sender reputation system will be deployed in conjunction with a combination of other techniques including content based filtering. As such, as a first line of defense, this system will be very effective in eliminating a lot of undesired senders. In fact, once a sender is identified as a spammer, the mail server does not even need to accept the connection request, saving network bandwidth and computational resources. The features we describe below improve accuracy further.

4.2.2 Single-Header and Single-Message Features

Single-packet features allow *SNARE* to rapidly identify and drop connections from spammers even before looking at the message header. Once a mail server has accepted the connection and examined the entire message, *SNARE* can determine sender reputation with increased confidence by looking at an additional set of features. As described in Section 3.2, these features include the number of recipients and message body length. Table 1(b) shows the prediction accuracy when we combine the single-packet features (i.e., those from the previous section) with these additional features. As the results from Section 3 suggest, adding the *message body length* and

number of recipients to the set of features further improves *SNARE*'s detection rate and false positive rate.

It is worth mentioning that the number of recipients listed on the "To" field is perhaps somewhat evadable: a sender could list the target email addresses on "Cc" and "Bcc" fields. Besides, if the spammers always place a single recipient address in the "To" field, this value will be the same as the large majority of legitimate messages. Because we did not have logs of additional fields in the SMTP header beyond the count of email addresses on the "To" field, we could not evaluate whether considering number of recipients listed under "Cc" and "Bcc" headers is worthwhile.

4.2.3 Aggregate Features

If multiple messages from a sender are available, the following features can be computed: the mean and variance of geodesic distances, message body lengths and number of recipients. We evaluate a classifier that is trained on *aggregate statistics* from the past 24 hours together with the features from previous sections.

Table 1(c) shows the performance of *RuleFit* with these aggregate features, and the ROC curve is plotted as the solid one in Figure 8. Applying the aggregate features decreases the error rate further: 70% of spam is identified correctly, while the false positive rate is merely 0.20%. The content-based filtering is very efficient to identify spam, but can not satisfy the requirement of processing a huge amount of messages for big mail servers. The prediction phase of *RuleFit* is faster, where the query is traversed from the root of the decision tree to a bottom label. Given the low false positive rate, *SNARE* would be a perfect first line of defense, where suspicious messages are dropped or re-routed to a farm for further analysis.

4.3 Other Considerations

Detection of "fresh" spammers We examined data trace, extracted the IP addresses not showing up in the previous training window, and further investigated the detection accuracy for those 'fresh' spammers with all *SNARE*'s features. If fixing the true positive as 70%, the false positive will increase to 5.2%, as shown in Figure 9. Compared with Figure 8, the decision on the new legitimate users becomes worse, but most of the new spammers can still be identified, which validates that *SNARE* is capable of *automatically* classifying "fresh" spammers.

Relative importance of individual features We use the fact that *RuleFit* can evaluate the *relative importance* of the features we have examined in Sections 3. Table 2 ranks all spatio-temporal features (with the most important feature at top). The top three features—*AS num*, *avg length* and *neig density*—play an important role in separating out spammers from good senders. This result is quite promising, since most of these features are lightweight: Better yet, two of these three can be computed having received only a single packet from the sender. As we will discuss in Section 6, they are also relatively resistant to evasion.

rank	*Feature Description*
1	AS number of the sender's IP
2	average of message length in previous 24 hours
3	average distance to the 20 nearest IP neighbors of the sender in the log
4	standard deviation of message length in previous 24 hours
5	status of email-service ports on the sender
6	geodesic distance between the sender and the recipient
7	number of recipient
8	average geodesic distance in previous 24 hours
9	average recipient number in previous 24 hours
10	probability ratio of spam to ham when getting the message
11	standard deviation of recipient number in previous 24 hours
12	length of message body
13	standard deviation of geodesic distance in previous 24 hours

Table 2: Ranking of feature importance in *SNARE*.

Correlation analysis among features We use mutual information to investigate how tightly the features are coupled, and to what extent they might contain redundant information. Given two random variables, mutual information measures how much uncertainty of one variable is reduced after knowing the other (i.e., the information they share). For discrete variables, the mutual information of X and Y is calculated as: $I(X,Y) = \sum_{x,y} p(x,y) \log(\frac{p(x,y)}{p(x)p(y)})$. When logarithm base-two is used, the quantity reflects how many bits can be removed to encode one variable given the other one. Table 3 shows the mutual information between pairs of features for one day of training data (October 23, 2007). We do not show statistics from other days, but features on those days reflect similar quantities for mutual information. The features with continuous values (e.g., geodesic distance between the sender and the recipient) are transformed into discrete variables by dividing the value range into 4,000 bins (which yields good discrete approximation); we calculate mutual information over the discrete probabilities. The indexes of the features in the table are the same as the ranks in Table 2; the packet-based features are marked with black circles. We also calculate the entropy of every feature and show them next to the indices in Table 3.

The interpretation of mutual information is consistent only within a single column or row, since comparison of mutual information without any common variable is meaningless. The table, of course, begs additional analysis but shows some interesting observations. The top-ranked feature, AS number, shares high mutual information (shown in bold) with several other features, especially with feature 6, geodesic distance between sender and recipient. The aggregate features of first-order statis-

	❶ (8.68)	2 (7.29)	❸ (2.42)	4 (6.92)	❺ (1.20)	❻ (10.5)	7 (0.46)	8 (9.29)	9 (2.98)	❿ (4.45)	11 (3.00)	12 (6.20)
2 (7.29)	**4.04**											
❸ (2.42)	**1.64**	1.18										
4 (6.92)	**3.87**	4.79	1.23									
❺ (1.20)	**0.65**	0.40	0.11	0.43								
❻ (10.5)	**5.20**	3.42	0.88	3.20	0.35							
7 (0.46)	**0.11**	0.08	0.02	0.08	0.004	0.15						
8 (9.29)	**5.27**	5.06	1.20	4.79	0.46	5.16	0.13					
9 (2.98)	**1.54**	1.95	0.53	2.03	0.09	1.17	0.10	2.08				
❿ (4.45)	**0.66**	0.46	0.07	0.49	0.02	0.87	0.006	0.85	0.13			
11 (3.00)	**1.87**	1.87	0.75	2.04	0.16	1.55	0.09	2.06	1.87	0.20		
12 (6.20)	**2.34**	2.53	0.49	2.12	0.20	2.34	0.07	2.30	0.52	0.31	0.73	
13 (8.89)	**4.84**	4.78	1.15	4.69	0.41	4.77	0.11	6.47	1.98	0.69	2.04	2.13

Table 3: Mutual information among features in *SNARE*; packet-based features are shown with numbers in dark circles. (The indices are the feature ranking in Table 2.)

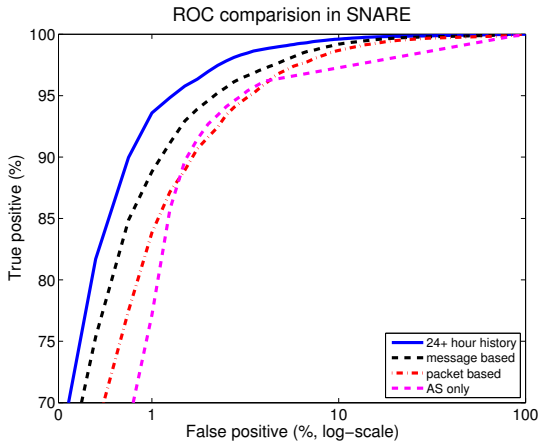


Figure 10: ROC comparison with AS-only case.

tics (e.g., feature 2, 4, 8) also have high values with each other. Because spammers may exhibit one or more of these features across each message, aggregating the features across multiple message over time indicates that, observing a spammer over time will reveal many of these features, though not necessarily on any message or single group of message. For this reason, aggregate features are likely to share high mutual information with other features that are common to spammers.

One possible reason that aggregate features have high mutual information with each other is that aggregating the features across multiple messages over time incorporates history of an IP address that may exhibit many of these characteristics over time.

Performance based on AS number only Since AS number is the most influential feature according to *RuleFit* and shares high mutual information with many other features, we investigated how well this feature alone can distinguish spammers from legitimate senders. We feed the AS feature into the predictive model and plot the ROC as the lower dashed curve in Figure 10. To make a close comparison, the "packet-based", "message-based", and "history-based" ROCs (the same as those in Figure 8) are shown as well, and the false positive is displayed on a log scale. The classifier gets false positive 0.76% under a 70% detection rate. Recall from Table 1 the false positive rate with "packet-based" features is almost a half, 0.44%, and that with "history-based" features will further reduce to 0.20%, which demonstrates that other features help to improve the performance. We also note that using the AS number alone as a distinguishing feature may cause large amounts of legitimate email to be misclassified, and could be evaded if an spammer decides to announce routes with a forged origin AS (which is an easy attack to mount and a somewhat common occurrence) [2, 26, 39].

5 A Spam-Filtering System

This section describes how *SNARE*'s reputation engine could be integrated into an overall spam-filtering system that includes a whitelist and an opportunity to continually retrain the classifier on labeled data (e.g., from spam traps, user inboxes, etc.). Because *SNARE*'s reputation engine still has a non-zero false positive rate, we show how it might be incorporated with mechanisms that could help further improve its accuracy, and also prevent discarding legitimate mail even in the case of some false positives. We propose an overview of the system and evaluate the benefits of these two functions on overall system accuracy.

5.1 System Overview

Figure 11 shows the overall system framework. The system needs not reside on a single server. Large public email providers might run their own instance of *SNARE*, since they have plenty of email data and processing resources. Smaller mail servers might query a remote

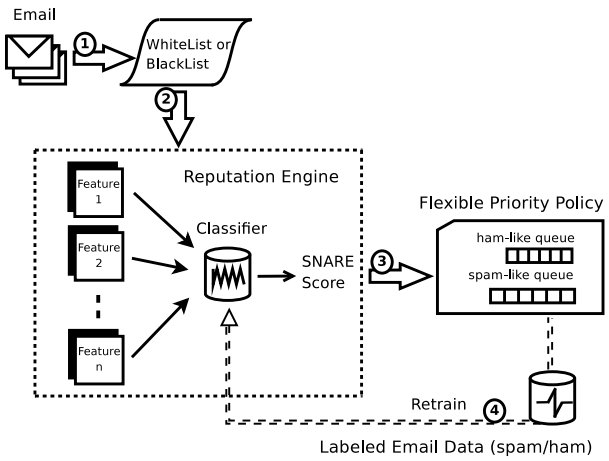


Figure 11: *SNARE* framework.

SNARE server. We envision that *SNARE* might be integrated into the workflow in the following way:

1. **Email arrival.** After getting the first packet, the mail server submits a query to the *SNARE* server (only the source and destination IP). Mail servers can choose to send more information to *SNARE* after getting the SMTP header or the whole message. Sending queries on a single packet or on a message is a tradeoff between detection accuracy and processing time for the email (i.e., sending the request early will make mail server get the response early). The statistics of messages in the received queries will be used to build up the *SNARE* classifier.
2. **Whitelisting.** The queries not listed in the whitelist will be passed to *SNARE*'s reputation engine (presented in Section 2.3) *before* any spam-filtering checks or content-based analysis. The output is a score, where, by default, positive value means likely spam and negative value means likely ham; and the absolute values represent the confidence of the classification. Administrators can set a different score threshold to make tradeoff between the false positive and the detection rate. We evaluate the benefits of whitelisting in Section 5.2.1.
3. **Greylisting and content-based detection.** Once the reputation engine calculates a score, the email will be delivered into different queues. More resource-sensitive and time-consuming detection methods (e.g., content-based detection) can be applied at this point. When the mail server has the capability to receive email, the messages in ham-like queue have higher priority to be processed, whereas the messages in spam-like queue will be offered less resources. This policy allows the server to speed up processing the messages that *SNARE* classifies as spam. The advantage of this hierarchical detecting scheme is that the legitimate email will be delivered to users' inbox sooner. Messages in the spam-like queue could be shunted to more resource-intensive spam filters before they are ultimately dropped.[3]
4. **Retraining** Whether the IP address sends spam or legitimate mail in that connection is not known at the time of the request, but is known after mail is processed by the spam filter. *SNARE* depends on accurately labelled training data. The email will eventually receive more careful checks (shown as "Retrain" in Figure 11). The results from those filters are considered as ground truth and can be used as feedback to dynamically adjust the *SNARE* threshold. For example, when the mail server has spare resource or much email in the spam-like queue is considered as legitimate later, *SNARE* system will be asked to act more generous to score email as likely ham; on the other hand, if the mail server is overwhelmed or the ham-like queue has too many incorrect labels, *SNARE* will be less likely to put email into ham-like queue. Section 5.2.2 evaluates the benefits of retraining for different intervals.

5.2 Evaluation

In this section, we evaluate how the two additional functions (whitelisting and retraining) improve *SNARE*'s overall accuracy.

5.2.1 Benefits of Whitelisting

We believe that a whitelist can help reduce *SNARE*'s overall false positive rate. To evaluate the effects of such a whitelist, we examined the features associated with the false positives, and determine that, 43% of all of *SNARE*'s false positives for a single day originate from just 10 ASes. We examined this characteristic for different days and found that 30% to 40% of false positives from any given day originate from the top 10 ASes. Unfortunately, however, these top 10 ASes do not remain the same from day-to-day, so the whitelist may need to be retrained periodically. It may also be the case that other features besides AS number of the source provide an even better opportunity for whitelisting. We leave the details of refining the whitelist for future work.

Figure 12 shows the average ROC curve when we whitelist the top 50 ASes responsible for most misclassified ham in each day. This whitelisting reduces the best

[3] Although *SNARE*'s false positive rates are quite low, some operators may feel that any non-zero chance that legitimate mail or sender might be misclassified warrants at least a second-pass through a more rigorous filter.

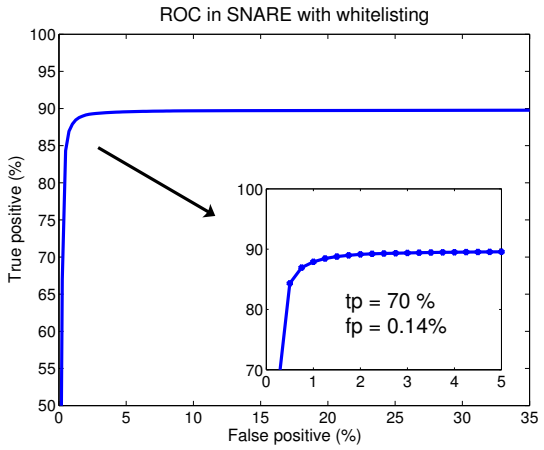


Figure 12: ROC in *SNARE* with whitelisting on ASes.

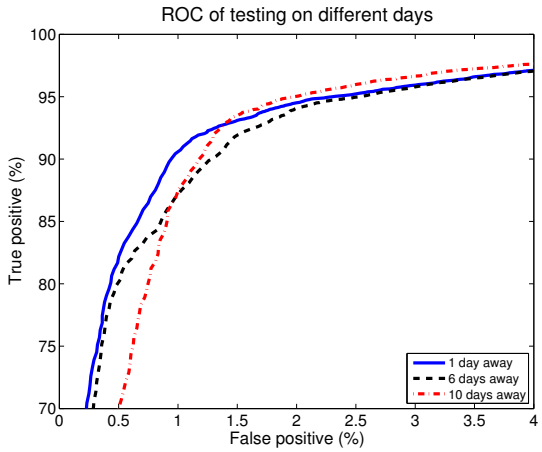


Figure 13: ROC using previous training rules to classify future messages.

possible detection rate considerably (effectively because about 11% of spam originates from those ASes). However, this whitelisting also reduces the false positive rate to about 0.14% for a 70% detection rate. More aggressive whitelisting, or whitelisting of other features, could result in even lower false positives.

5.2.2 Benefits of Retraining

Setup Because email sender behavior is dynamic, training *SNARE* on data from an earlier time period may eventually grow stale. To examine the requirements for periodically retraining the classifier, we train *SNARE* based on the first 3 days' data (through October 23–25, 2007) and test on the following 10 days. As before, we use 1 million randomly sampled spam and ham messages to test the classifier for each day.

Results Figure 13 shows the false positive and true positive on 3 future days, October 26, October 31, and November 4, 2007, respectively. The prediction on future days will become more inaccurate with time passage. For example, on November 4 (ten days after training), the false positive rate has dropped given the same true positive on the ROC curve. This result suggests that, for the spammer behavior in this trace, retraining *SNARE*'s classification algorithms daily should be sufficient to maintain accuracy. (We expect that the need to retrain may vary across different datasets.)

6 Discussion and Limitations

In this section, we address various aspects of *SNARE* that may present practical concerns. We first discuss the extent to which an attacker might be able to evade various features, as well as the extent to which these features might vary across time and datasets. We then discuss scalability concerns that a production deployment of *SNARE* may present, as well as various possible workarounds.

6.1 Evasion-Resistance and Robustness

In this section, we discuss the evasion resistance of the various network-level features that form the inputs to *SNARE*'s classification algorithm. Each of these features is, to some degree, evadable. Nevertheless, *SNARE* raises the bar by making it more difficult for spammers to evade detection without altering the techniques that they use to send spam. Although spammers might adapt to evade some of the features below, we believe that it will be difficult for a spammer to adjust all features to pass through *SNARE*, particularly without somewhat reducing the effectiveness of the spamming botnet. We survey each of the features from Table 2 in turn.

AS number AS numbers are more persistently associated with a sender's identity than the IP address, for two reasons: (1) The spamming mail server might be set up within specific ASes without the network administrator shutting it down. (2) Bots tend to aggregate within ASes, since the machines in the same ASes are likely to have the same vulnerability. It is not easy for spammers to move mail servers or the bot armies to a different AS; therefore, AS numbers are robust to indicate malicious hosts.

Message length In our analysis, we discovered that the size of legitimate email messages tends to be much more variable than that of spam (perhaps because spammers often use templates to sent out large quantities of mail [25]). With knowledge of this feature, a spammer might start to randomize the lengths of their email mes-

sages; this attack would not be difficult to mount, but it might restrict the types of messages that a spammer could send or make it slightly more difficult to coordinate a massive spam campaign with similar messages.

Nearest neighbor distances Nearest neighbor distance is another feature that will be hard to modify. Distances to k nearest neighbors effectively isolate existence of unusually large number of email servers within a small sequence of IP addresses. If the spammers try to alter their neighborhood density, they will not be able to use too many machines within a compromised subnet to send spam to the same set of destinations. Although it is possible for a botnet controller to direct bots on the same subnet to target different sets of destinations, such evasion does require more coordination and, in some cases, may restrict the agility that each spamming bot has in selecting its target destinations.

Status of email service ports Some limitation might fail the active probes, e.g., the outgoing mail servers use own protocol to mitigate messages (such as Google mail) or a firewall blocks the connections from out of the domain. But the bots do not open such ports with high probability, and the attackers need to get root privilege to enable those ports (which requires more sophisticated methods and resources). The basic idea is to find out whether the sender is a legitimate mail server. Although we used active probes in *SNARE*, other methods could facilitate the test, such as domain name checking or mail server authentication.

Sender-receiver geodesic distance The distribution of geodesic distances between the spammers' physical location and their target IP's location is a result of the spammers' requirement to reach as many target mail boxes as possible and in the shortest possible time. Even in a large, geographically distributed botnet, requiring each bot to bias recipient domains to evade this feature may limit the flexibility of how the botnet is used to send spam. Although this feature can also be evaded by tuning the recipient domains for each bot, if bots only sent spam to nearby recipients, the flexibility of the botnet is also somewhat restricted: it would be impossible, for example, to mount a coordinate spam campaign against a particular region from a fully distributed spamming botnet.

Number of recipients We found that spam messages tend to have more recipients than legitimate messages; a spammer could likely evade this feature by reducing the number of recipients on each message, but this might make sending the messages less efficient, and it might alter the sender behavior in other ways that might make a spammer more conspicuous (e.g., forcing the spammer to open up more connections).

Time of day This feature may be less resistant to evasion than others. Having said that, spamming botnets' diurnal pattern results from when the infected machines are switched on. For botnets to modify their diurnal message volumes over the day to match the legitimate message patterns, they will have to lower their spam volume in the evenings, especially between 3:00 p.m. and 9:00 p.m. and also reduce email volumes in the afternoon. This will again reduce the ability of botnets to send large amounts of email.

6.2 Other Limitations

We briefly discuss other current limitations of *SNARE*, including its ability to scale to a large number of recipients and its ability to classify IP addresses that send both spam and legitimate mail.

Scale *SNARE* must ultimately scale to thousands of domains and process hundreds of millions of email addresses per day. Unfortunately, even state-of-the-art machine learning algorithms are not well equipped to process datasets this large; additionally, sending data to a central coordinator for training could potentially consume considerably bandwidth. Although our evaluation suggests that *SNARE*'s classification is relatively robust to sampling of training data, we intend to study further the best ways to sample the training data, or perhaps even perform in-network classification.

Dual-purpose IP addresses Our conversations with large mail providers suggest that one of the biggest emerging threats are "web bots" that send spam from Web-based email accounts [35]. As these types of attacks develop, an increasing fraction of spam may be sent from IP addresses that also send significant amounts of legitimate mail. These cases, where an IP address is neither good nor bad, will need more sophisticated classifiers and features, perhaps involving timeseries-based features.

7 Related Work

We survey previous work on characterizing the network-level properties and behavior of email senders, email sender reputation systems, and other email filtering systems that are not based on content.

Characterization studies Recent characterization studies have provided increasing evidence that spammers have distinct network-level behavioral patterns. Ramachandran *et al.* [34] showed that spammers utilize transient botnets to spam at low rate from any specific IP to any domain. Xie *et al.* [38] discovered that a vast

majority of mail servers running on dynamic IP address were used solely to send spam. In their recently published study [37], they demonstrate a technique to identify bots by using signatures constructed from URLs in spam messages. Unlike *SNARE*, their signature-based botnet identification differs heavily on analyzing message content. Others have also examined correlated behavior of botnets, primarily for characterization as opposed to detection [25, 31]. Pathak *et al.* [29] deployed a relay sinkhole to gather data from multiple spam senders destined for multiple domains. They used this data to demonstrate how spammers utilize compromised relay servers to evade detection; this study looked at spammers from multiple vantage points, but focused mostly on characterizing spammers rather than developing new detection mechanisms. Niu *et al.* analyzed network-level behavior of Web spammers (e.g., URL redirections and "doorway" pages) and proposed using context-based analysis to defend against Web spam [28].

Sender reputation based on network-level behavior SpamTracker [34] is most closely related to *SNARE*; it uses network-level behavioral features from data aggregated across multiple domains to infer sender reputation. While that work initiated the idea of *behavioral blacklisting*, we have discovered many other features that are more lightweight and more evasion-resistant than the single feature used in that paper. Beverly and Sollins built a similar classifier based on transport-level characteristics (e.g., round-trip times, congestion windows) [12], but their classifier is both heavyweight, as it relies on SVM, and it also requires accepting the messages to gather the features. Tang *et al.* explored the detection of spam senders by analyzing the behavior of IP addresses as observed by query patterns [36]. Their work focuses on the breadth and the periodicity of message volumes in relation to sources of queries. Various previous work has also attempted to cluster email senders according to groups of recipients, often with an eye towards spam filtering [21, 24, 27], which is similar in spirit to *SNARE*'s geodesic distance feature; however, these previous techniques typically require analysis of message contents, across a large number of recipients, or both, whereas *SNARE* can operate on more lightweight features. McAfee's TrustedSource [4] and Cisco IronPort [3] deploy spam filtering appliances to hundreds or thousands of domains which then query the central server for sender reputation and also provide meta-data about messages they receive; we are working with McAfee to deploy *SNARE* as part of TrustedSource.

Non-content spam filtering Trinity [14] is a distributed, content-free spam detection system for messages originating from botnets that relies on message volumes. The SpamHINTS project [9] also has the stated goal of building a spam filter using analysis of network traffic patterns instead of the message content. Clayton's earlier work on extrusion detection involves monitoring of server logs at both the local ISP [16] as well as the remote ISP [17] to detect spammers. This work has similar objectives as ours, but the proposed methods focus more on properties related to SMTP sessions from only a single sender.

8 Conclusion

Although there has been much progress in content-based spam filtering, state-of-the-art systems for *sender reputation* (e.g., DNSBLs) are relatively unresponsive, incomplete, and coarse-grained. Towards improving this state of affairs, this paper has presented *SNARE*, a sender reputation system that can accurately and automatically classify email senders based on features that can be determined early in a sender's history—sometimes after seeing only a single IP packet.

Several areas of future work remain. Perhaps the most uncharted territory is that of using temporal features to improve accuracy. All of *SNARE*'s features are essentially discrete variables, but we know from experience that spammers and legitimate senders also exhibit different temporal patterns. In a future version of *SNARE*, we aim to incorporate such temporal features into the classification engine. Another area for improvement is making *SNARE* more evasion-resistant. Although we believe that it will be difficult for a spammer to evade *SNARE*'s features and still remain effective, designing classifiers that are more robust in the face of active attempts to evade and mis-train the classifier may be a promising area for future work.

Acknowledgments

We thank our shepherd, Vern Paxson, for many helpful suggestions, including the suggestions to look at mutual information between features and several other improvements to the analysis and presentation. We also thank Wenke Lee, Anirudh Ramachandran, and Mukarram bin Tariq for helpful comments on the paper. This work was funded by NSF CAREER Award CNS-0643974 and NSF Awards CNS-0716278 and CNS-0721581.

References

[1] FCrDNS Lookup Testing. http://ipadmin.junkemailfilter.com/rdns.php.

[2] Internet Alert Registry. http://iar.cs.unm.edu/.

[3] IronPort. http://www.ironport.com.

[4] McAfee Secure Computing. http://www.securecomputing.com.

[5] SORBS: Spam and Open Relay Blocking System. http://www.au.sorbs.net/.

[6] SpamCop. http://www.spamcop.net/bl.shtml.

[7] SpamHaus IP Blocklist. http://www.spamhaus.org.

[8] GeoIP API. MaxMind, LLC. http://www.maxmind.com/app/api, 2007.

[9] spamHINTS: Happily It's Not The Same. http://www.spamhints.org/, 2007.

[10] DNSBL Resource: Statistics Center. http://stats.dnsbl.com/, 2008.

[11] ALPEROVITCH, D., JUDGE, P., AND KRASSER, S. Taxonomy of email reputation systems. In *Proc. of the First International Workshop on Trust and Reputation Management in Massively Distributed Computing Systems (TRAM)* (2007).

[12] BEVERLY, R., AND SOLLINS, K. Exploiting the transport-level characteristics of spam. In *5th Conference on Email and Anti-Spam (CEAS)* (2008).

[13] BOYKIN, P., AND ROYCHOWDHURY, V. Personal email networks: An effective anti-spam tool. *IEEE Computer 38*, 4 (2005), 61–68.

[14] BRODSKY, A., AND BRODSKY, D. A distributed content independent method for spam detection. In *First Workshop on Hot Topics in Understanding Botnets (HotBots)* (2007).

[15] BURGES, C. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery 2*, 2 (1998), 121–167.

[16] CLAYTON, R. Stopping spam by extrusion detection. In *First Conference of Email and Anti-Spam (CEAS)* (2004).

[17] CLAYTON, R. Stopping outgoing spam by examining incoming server logs. In *Second Conference on Email and Anti-Spam (CEAS)* (2005).

[18] FRIEDMAN, J., AND POPESCU, B. Gradient directed regularization. *Stanford University, Technical Report* (2003).

[19] FRIEDMAN, J., AND POPESCU, B. Predictive learning via rule ensembles. *Annals of Applied Statistics (to appear)* (2008).

[20] GOLBECK, J., AND HENDLER, J. Reputation network analysis for email filtering. In *First Conference on Email and Anti-Spam (CEAS)* (2004).

[21] GOMES, L. H., CASTRO, F. D. O., ALMEIDA, R. B., BETTENCOURT, L. M. A., ALMEIDA, V. A. F., AND ALMEIDA, J. M. Improving spam detection based on structural similarity. In *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)* (2005).

[22] GOODMAN, J., CORMACK, G., AND HECKERMAN, D. Spam and the ongoing battle for the inbox. *Communications of the ACM 50*, 2 (2007), 24–33.

[23] HULTON, E., AND GOODMAN, J. Tutorial on junk email filtering. *Tutorial in the 21st International Conference on Machine Learning (ICML)* (2004).

[24] JOHANSEN, L., ROWELL, M., BUTLER, K., AND MCDANIEL, P. Email communities of interest. In *4th Conference on Email and Anti-Spam (CEAS)* (2007).

[25] KANICH, C., KREIBICH, C., LEVCHENKO, K., ENRIGHT, B., PAXSON, V., VOELKER, G. M., AND SAVAGE, S. Spamalytics: An empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)* (2008).

[26] KARLIN, J., FORREST, S., AND REXFORD, J. Autonomous security for autonomous systems. *Computer Networks 52*, 15 (2008), 2908–2923.

[27] LAM, H., AND YEUNG, D. A learning approach to spam detection based on social networks. In *4th Conference on Email and Anti-Spam (CEAS)* (2007).

[28] NIU, Y., WANG, Y.-M., CHEN, H., MA, M., AND HSU, F. A quantitative study of forum spamming using context-based analysis. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS)* (2007).

[29] PATHAK, A., HU, C., Y., AND MAO, Z., M. Peeking into spammer behavior from a unique vantage point. In *First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)* (2008).

[30] QUINLAN, J. Induction of decision trees. *Machine Learning 1*, 1 (1986), 81–106.

[31] RAJAB, M., ZARFOSS, J., MONROSE, F., AND TERZIS, A. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC)* (2006).

[32] RAMACHANDRAN, A., DAGON, D., AND FEAMSTER, N. Can DNSBLs keep up with bots? In *3rd Conference on Email and Anti-Spam (CEAS)* (2006).

[33] RAMACHANDRAN, A., AND FEAMSTER, N. Understanding the network-level behavior of spammers. In *Proceedings of the ACM SIGCOMM* (2006).

[34] RAMACHANDRAN, A., FEAMSTER, N., AND VEMPALA, S. Filtering spam with behavioral blacklisting. In *ACM Conference on Computer and Communications Security (CCS)* (2007).

[35] Private conversation with Mark Risher, Yahoo Mail., 2008.

[36] TANG, Y. C., KRASSER, S., JUDGE, P., AND ZHANG, Y.-Q. Fast and effective spam IP detection with granular SVM for spam filtering on highly imbalanced spectral mail server behavior data. In *2nd International Conference on Collaborative Computing (CollaborateCom)* (2006).

[37] XIE, Y., YU, F., , ACHAN, K., PANIGRAHY, R., HULTEN, G., AND OSIPKOV, I. Spamming bots: Signatures and characteristics. In *Proceedings of ACM SIGCOMM* (2008).

[38] XIE, Y., YU, F., ACHAN, K., GILUM, E., GOLDSZMIDT, M., AND WOBBER, T. How dynamic are IP addresses. In *Proceedings of ACM SIGCOMM* (2007).

[39] ZHAO, X., PEI, D., WANG, L., MASSEY, D., MANKIN, A., WU, S. F., AND ZHANG, L. An analysis of BGP multiple origin AS (MOAS) conflicts. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement (IMW)* (2001).

Proceedings of ACM SIGCOMM
Seattle, WA. August 2008.

Answering "What-If" Deployment and Configuration Questions with WISE*

Mukarram Bin Tariq[‡†], Amgad Zeitoun[§], Vytautas Valancius[‡], Nick Feamster[‡], Mostafa Ammar[‡]
mtariq@cc.gatech.edu, amgad@google.com, {valas,feamster,ammar}@cc.gatech.edu
[‡] School of Computer Science, Georgia Tech. Atlanta, GA [§] Google Inc. Mountain View, CA

Abstract

Designers of content distribution networks often need to determine how changes to infrastructure deployment and configuration affect service response times when they deploy a new data center, change ISP peering, or change the mapping of clients to servers. Today, the designers use coarse, back-of-the-envelope calculations, or costly field deployments; they need better ways to evaluate the effects of such hypothetical "what-if" questions before the actual deployments. This paper presents *What-If Scenario Evaluator (WISE)*, a tool that predicts the effects of possible configuration and deployment changes in content distribution networks. *WISE* makes three contributions: (1) an algorithm that uses traces from existing deployments to learn causality among factors that affect service response-time distributions; (2) an algorithm that uses the learned causal structure to estimate a dataset that is representative of the hypothetical scenario that a designer may wish to evaluate, and uses these datasets to predict future response-time distributions; (3) a scenario specification language that allows a network designer to easily express hypothetical deployment scenarios without being cognizant of the dependencies between variables that affect service response times. Our evaluation, both in a controlled setting and in a real-world field deployment at a large, global CDN, shows that *WISE* can quickly and accurately predict service response-time distributions for many practical *what-if* scenarios.

Categories and Subject Descriptors: C.2.3 [Computer Communication Networks]: Network Operations, Network Management

General Terms: Algorithms, Design, Management, Performance

Keywords: What-if Scenario Evaluation, Content Distribution Networks, Performance Modeling

1. INTRODUCTION

Content distribution networks (CDNs) for Web-based services comprise hundreds to thousands of distributed servers and data centers [1, 3, 9]. Operators of these networks continually strive to improve the response times for their services. To perform this task, they must be able to predict how service response-time distribution changes in various hypothetical *what-if* scenarios, such as changes to network conditions and deployments of new infrastructure. In many cases, they must also be able to reason about the *detailed* effects of these changes (e.g., what fraction of the users will see at least a 10% improvement in performance because of this change?), as opposed to just coarse-grained point estimates or averages.

Various factors on both short and long timescales affect a CDN's service response time. On short timescales, response time can be affected by routing instability or changes in server load. Occasionally, the network operators may "drain" a data center for maintenance and divert the client requests to an alternative location. In the longer term, service providers may upgrade their existing facilities, move services to different facilities or deploy new data centers to address demands and application requirements, or change peering and customer relationships with neighboring ISPs. These instances require significant planning and investment; some of these decisions are hard to implement and even more difficult to reverse.

Unfortunately, reasoning about the effects of any of these changes is extremely challenging in practice. Content distribution networks are complex systems, and the response time perceived by a user can be affected by a variety of inter-dependent and correlated factors. Such factors are difficult to accurately model or reason about and back-of-the-envelope calculations are not precise.

This paper presents the design, implementation, and evaluation of *What-If Scenario Evaluator* (*WISE*), a tool that estimates the effects of possible changes to network configuration and deployment scenarios on the service response time. *WISE* uses statistical learning techniques to provide a largely automated way of interpreting the *what-if* questions as statistical interventions. *WISE* takes as input packet traces from Web transactions to model factors that affect service response-time prediction. Using this model, *WISE* also transforms the existing datasets to produce a new datasets that are representative of the *what-if* scenarios and are also faithful to the working of the system, and finally uses these to estimate the system response time distribution.

Although function estimation using passive datasets is a common application in the field of machine learning, using these techniques is not straightforward because they can only predict the response-time distribution for a *what-if* scenario accurately if the estimated function receives an input distribution that is representative of the *what-if* scenario. Providing this input distribution presents difficulties at several levels, and is the key problem that *WISE* solves.

WISE tackles the following specific challenges. First, ***WISE* must allow the network designers to easily specify *what-if* scenarios**. A designer might specify a *what-if* scenario to change the

*This work is supported in part by NSF Awards CNS-0643974, CNS-0721581, and CNS-0721559.
†Work performed while the author was visiting Google Inc.

SIGCOMM'08, August 17–22, 2008, Seattle, Washington, USA.

value of some network features relative to their values in an existing or "baseline" deployment. The designer may not know that such a change might also affect other features (or how the features are related). *WISE*'s interface shields the designers from this complexity. *WISE* provides a *scenario specification language* that allows network designers to succinctly specify hypothetical scenarios for arbitrary subsets of existing networks and to specify *what-if* values for different features. *WISE*'s specification language is simple: evaluating a hypothetical deployment of a new proxy server for a subset of users can be specified in only 2 to 3 lines of code.

Second, because the designer can specify a *what-if* scenario without being aware of these dependencies, ***WISE* must automatically produce an accurate dataset** that is both *representative* of the *what-if* scenario the designer specifies and *consistent* with the underlying dependencies. *WISE* uses a causal dependency discovery algorithm to discover the dependencies among variables and a statistical intervention evaluation technique to transform the observed dataset to a representative and consistent dataset. *WISE* then uses a non-parametric regression method to estimate the response time as a piece-wise smooth function for this dataset. We have used *WISE* to predict service response times in both controlled settings on the Emulab testbed and for Google's global CDN for its Web-search service. Our evaluation shows that *WISE*'s predictions of response-time distribution are very accurate, yielding a median error between 8% and 11% for cross-validation with existing deployments and only 9% maximum cumulative distribution difference compared to ground-truth response time distribution for *what-if* scenarios on a real deployment as well as controlled experiments on Emulab.

Finally, ***WISE* must be fast**, so that it can be used for short-term and frequently arising questions. Because the methods relying on statistical inference are often computationally intensive, we have tailored *WISE* for parallel computation and implemented it using the Map-Reduce [16] framework, which allows us to process large datasets comprising hundreds of millions of records quickly and produce accurate predictions for response-time distributions.

The paper proceeds as follows. Section 2 describes the problem scope and motivation. Section 3 makes the case for using statistical learning for the problem of *what-if* scenario evaluation. Section 4 provides an overview of *WISE*, and Section 5 describes *WISE*'s algorithms in detail. We discuss the implementation in Section 6. In Section 7, we evaluate *WISE* for response-time estimation for existing deployments as well as for a *what-if* scenario based on a real operational event. In Section 8, we evaluate *WISE* for *what-if* scenarios for a small-scale network built on the Emulab testbed. In Section 9, we discuss various properties of the *WISE* system and how it relates to other areas in networking. We review related work in Section 10, and conclude in Section 11.

2. PROBLEM CONTEXT AND SCOPE

This section describes common *what-if* questions that the network designers pose when evaluating potential configuration or deployment changes to an existing content distribution network deployment.

Content Distribution Networks: Most CDNs conform to a two-tier architecture. The first tier comprises a set of globally distributed front-end (FE) servers that, depending on the specific implementation, provide caching, content assembly, pipelining, request redirection, and proxy functions. The second tier comprises backend (BE) servers that implement the application logic, and which might also be replicated and globally distributed. The FE and BE servers may belong to a single administrative entity (as is the case with Google [3]) or to different administrative entities, as with commercial content distribution networking service providers, such as Akamai [1]. The network path between the FE and BE servers may be over a public network or a private network, or a LAN when the two are co-located. CDNs typically use DNS redirection or URL-rewriting [13] to direct the users to the appropriate FE and BE servers; this redirection may be based on the user's proximity, geography, availability, and relative server load.

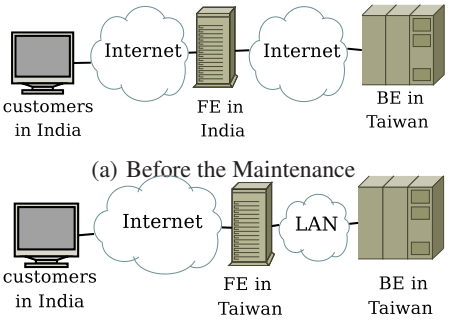


Figure 1: Network configuration for customers in India.

An Example "What-if" Scenario: The network designers may want to ask a variety of *what-if* questions about the CDN configuration. For example, the network designers may want to determine the effects of deploying new FE or BE servers, changing the serving FE or BE servers for a subset of users, changing the size of typical responses, increasing capacity, or changing network connectivity, on the service response time. Following is a real *what-if* scenario from Google's CDN for the Web-search service.

Figure 1 shows an example of a change in network deployment that could affect server response time. Google has an FE data center in India that serves users in India and surrounding regions. This FE data center uses BE servers located elsewhere in the world, including the ones located in Taiwan. On July 16, 2007, the FE data center in India was temporarily "drained" for maintenance reasons, and the traffic was diverted to a FE data center that is co-located with BE in Taiwan, resulting in a change in latency for the users in India. This change in the network configuration can be described as a what-if scenario in terms of change of the assigned FE, or more explicitly as changes in delays between FE and clients that occur due to the new configuration. *WISE* aims to predict the response-time distribution for reconfigurations before they are deployed in practice.

3. A CASE FOR MACHINE LEARNING

In this section, we present two aspects of *what-if* scenario evaluation that make the problem well-suited for machine learning: (1) an underlying model that is difficult to derive from first principles but provides a wealth of data; (2) a need to predict outcomes based on data that may not directly represent the desired *what-if* scenario.

The system is complex, but observable variables are driven by fundamental properties of the system. Unfortunately, in large complex distributed systems, such as CDNs, the parameters that govern the system performance, the relationships between these variables, as well as the functions that govern the response-time distribution of the system, are often complex and are characterized by randomness and variability that are difficult to model as simple readily evaluatable formulas. Fortunately, the underlying fundamental properties and dependencies that determine a CDN's

response time can be observed as correlations and joint probability distributions of the variables that define the system, including the service response time. By observing these joint distributions (e.g., response times observed under various conditions), machine learning algorithms can infer the underlying function that affects the response time. Because most production CDNs collect comprehensive datasets for their services as part of everyday operational and monitoring needs, the requisite datasets are typically readily available.

Obtaining datasets that directly represent the *what-if* scenario is challenging. Once the response-time function is learned, evaluating a *what-if* scenario requires providing this function with input data that is representative of the *what-if* scenario. Unfortunately, data collected from an existing network deployment only represents the current setup, and the system complexities make it difficult for a designer to manually "transform" the data to represent the new scenario. Fortunately, depending on the extent of the dataset that is collected and the nature of *what-if* scenario, machine learning algorithms can reveal the dependencies among the variables and use the dependency structure to intelligently *re-weigh and re-sample* the different parts of the existing dataset to perform this transformation. In particular, if the *what-if* scenario is expressed in terms of the changes to values of the variables that are observed in the dataset and the changed values or similar values of these variables are observed in the dataset even with small densities in the original dataset, then we can transform the original dataset to one that is representative of the *what-if* scenario as well as the underlying principles of the system, while requiring minimal input from the network designer.

4. WISE: HIGH-LEVEL DESIGN

WISE entails four steps: (1) identifying features in the dataset that affect response time; (2) constraining the inputs to "valid" scenarios based on existing dependencies; (3) specifying the *what-if* scenario; (4) estimating the response-time function and distribution. Each of these tasks raises a number of challenges, some of which are general problems with applying statistical learning in practice, and others are specific to *what-if* scenario evaluation. This section provides an overview and necessary background for these steps. Section 5 discuss the mechanisms in more depth; the technical report [24] provides additional details and background.

1. Identifying Relevant Features: The main input to *WISE* is a comprehensive dataset that covers many combinations of variables. Most CDNs have existing network monitoring infrastructure that can typically provide such a dataset. This dataset, however, may contain variables that are not relevant to the response-time function. *WISE* extracts the set of relevant variables from the dataset and discards the rest of the variables. *WISE* can also identify whether there are missing or latent variables that may hamper scenario evaluation (Sections 5.1 and 5.2 provide more details).

The nature of *what-if* scenarios that *WISE* can evaluate is limited by the input dataset—careful choice of variables that the monitoring infrastructure collects from a CDN can therefore enhance the utility of the dataset for evaluating *what-if* scenarios, choosing such variables is outside the scope of *WISE* system.

2. Preparing Dataset to Represent the What-if Scenario: Evaluating a *what-if* scenario requires values for input variables that "make sense." Specifically, an accurate prediction of the response-time distribution for a *what-if* scenario requires a joint distribution of the input variables that is representative of the scenario and is also consistent with the dependencies that are inherent to the system itself. For instance, the distribution of the number of packets that are transmitted in the duration of a service session depends on the distribution of the size of content that the server returns in reply to a request; if the distribution of content size changes, then the distribution for the number of packets that are transmitted must also change in a way that is inherent to the system, e.g., the path-MTU might determine the number of packets. Further, the change might *cascade* to other variables that in turn depend on the number of packets. To enforce such consistency *WISE* learns the dependency structure among the variables and represents these relationships as a *Causal Bayesian Network (CBN)* [20]. We provide a brief background of CBN in this Section and explain the algorithm for learning the CBN in Section 5.2.

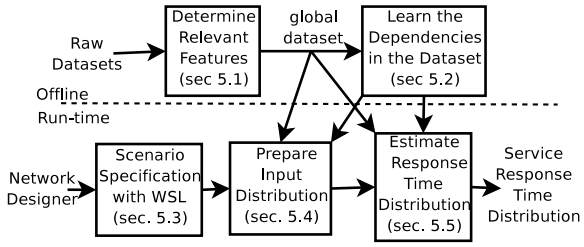
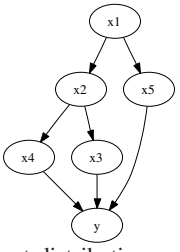


Figure 2: Main steps in the *WISE* approach.

A CBN represents the variables in the dataset as a Directed Acyclic Graph (DAG). The nodes represent the variables and the edges indicate whether there are dependencies among the variables. A variable has a "causal" relationship with another variable, if a change in the value of the first variable *causes* a change in the values of the later. When conditioned on its parent variables, a variable $\mathbf{x_i}$ in a CBN is independent of all other variables in the DAG except its decedents; an optimal DAG for a dataset is one where we find the minimal parents for each node that satisfy the above property.



As an example of how the causal structure may facilitate scenario specification and evaluation, consider a dataset with five input variables $(x_1 \ldots x_5)$, and target variable y. Suppose that we discover a dependency structure among them as shown in the figure to the right. If *WISE* is presented with a *what-if* scenario that requires changes in the value of variable x_2, then the distributions for variables x_1 and x_5 remains unchanged in the input distribution, and *WISE* needs to update only the distribution of the descendants of x_2 to maintain consistency. *WISE* constrains the input distribution by intelligently re-sampling and re-weighing the dataset using the causal structure as a guideline (see Section 5.4).

In general, correlation does not imply causation. Causal interpretation of association or correlation requires that the dataset is independent of the outcome variable (see the Counterfactual Model [20, 25]). A biased dataset can result in false or missing causal assertions; for example, we could falsely infer that a treatment is effective if, by coincidence, the dataset is such that more patients that are treated are healthy that the ones that are not treated. We can make the correct inference if we assign the patients randomly to the treatment because then the dataset would be independent of the outcome. Fortunately, because many computer networking phenomena are fundamentally similar throughout the Internet, we can assume that the datasets are unbiased. Still, frivolous relationships might arise; we address this further in Section 5.2.

3. Facilitating Scenario Specification: *WISE* presents the network designers with an easy-to-use interface in the form of a scenario specification language called *WISE*-Scenario Language

(WSL). The designers can typically specify the baseline setup as well as the hypothetical values for the scenario in 3-4 lines of WSL.

WSL allows the designers to evaluate a scenario for an arbitrary subset of customers. WSL also provides a useful set of built-in operators that facilitate scenario specification as relative changes to the existing values of variables or as new values from scratch. With WSL, the designers are completely shielded from the complexity of dependencies among the variables, because *WISE* automatically updates the dependent variables. We detail WSL and the process of scenario specification and evaluation in Sections 5.3 and 5.4.

4. Estimating Response-Time Distribution: Datasets for typical CDN deployments and *what-if* scenarios span a large multi-dimensional space. While non-parametric function estimation is a standard application in the machine learning literature, the computational requirements for accurately estimating a function spanning such a large space can be astronomical. To address this, *WISE* estimates the function in a piece-wise manner, and also structures the processing so that it is amenable to parallel processing. *WISE* also uses the dependency structure to reduce the number of variables that form the input to the regression function. Sections 5.5 and 5.6 provide more detail.

5. WISE SYSTEM

5.1 Feature Selection

Traditional machine-learning applications use various model selection criteria, e.g., Akaike Information Criterion (AIC), Mallow's C_p Test, or k-fold cross-validation [25], for determining appropriate subset of covariates for a learning problem. *WISE* forgoes the traditional model selection techniques in favor of simple pair-wise independence testing, because at times these techniques can ignore variables that might have interpretive value for the designer.

WISE uses simple pair-wise independence tests on all the variables in the dataset with the response-time variable, and discards all variables that it deems independent of the response-time variable. For each categorical variable (variables that do not have numeric meanings) in the dataset, such as, country of origin of a request, or AS number, *WISE* obtains the conditional distributions of response time for each categorical value, and discards the variable if all the conditional distributions of response time are statistically similar. To test this, we use Two-sample Kolmogorov-Smirnov (KS) goodness of fit test with a significance level of 10%.

For real-valued variables, *WISE* first tests for correlation with the response-time variable, and retains a variable if the correlation coefficient is greater than 10%. Unfortunately, for continuous variables, lack of correlation does not imply independence, so we cannot outright discard a variable if we observe small correlation. A typical example of such a variable in a dataset is the timestamp of the Web transaction, where the correlation may cancel out over a diurnal cycle. For such cases, we divide the range of the variable in question into small buckets and treat each bucket as a category. We then apply the same techniques as we do for the categorical variables to determine whether the variable is independent. There is still a possibility that we may discard a variable that is relevant, but this outcome is less likely if sufficiently small buckets are used. The bucket size depends on the variable in question; for instance, we use one-hour buckets for the time-stamp variable in the datasets.

5.2 Learning the Causal Structure

To learn the causal structure, *WISE* first learns the undirected graph and then uses a set of rules to orient the edges.

Learning the Undirected Graph: Recall that in a Causal Bayesian Network (CBN), a variable, when conditioned on its parents, is independent of all other variables, except its descendants. Further an optimal CBN requires finding the smallest possible set of parents for each node that satisfy this condition. Thus by definition, variables a and b in the CBN have an edge between them, if and only if, there is a subset of *separating variables*, S_{ab}, such that a is independent of b given S_{ab}. This, in the general case, requires searching all the possible $O(2^n)$ combinations of the n variables in the dataset

```
1: WCD (V, W_0, Δ)
   /*Notation
     V: set of all variables
     W_0: set of no-cause variables
     Δ: maximum allowable cardinality for separators
     a ⊥ b: Variable a is independent of variable b */
2:  Make a complete Graph on V
3:  Remove all edges (a, b) if a ⊥ b
4:  W = W_0
5:  for c = 1 to Δ /*prune in the order of increasing cardinality*/
6:      LocalPrune (c)

1: LocalPrune (c)
   /*Try to separate neighbors of frontier variables W*/
2:  ∀w ∈ W
3:    ∀z ∈ N(w) /*neighbors of w*/
4:      if ∃x ⊆ N(z)\w : |x| ≤ c, z ⊥ w|x
5:        then /*found separator node(s)*/
            S_wz = x /*assign the separating nodes*/
6:          Remove the edge (w, z)
7:          Remove edges (w', z), for all the nodes w' ∈ W
              that are also on path from w to nodes in W_0
          /*Update the new frontier variables*/
8:          W = W ∪ x
```

Figure 3: WISE Causal Discovery (WCD) algorithm.

WISE-Causal Discovery Algorithm (WCD) (Figure 3) uses a heuristic to guide the search of separating variables when we have prior knowledge of a subset of variables that are "not caused" by any other variables in the dataset, or that are determined by factors outside our system model (we refer to these variables as the *no-cause* variables). Further, WCD does not perform exhaustive search for separating variables, thus forgoing optimality for lower complexity.

WCD starts with a fully connected undirected graph on the variables and removes the edges among variables that are clearly independent. WCD then progressively finds separating nodes between a restricted set of variables (that we call *frontier* variables), and the rest of the variables in the dataset, in the order of increasing cardinality of allowable separating variables. Initially the frontier variables comprise only the no-cause variables. As WCD discovers separating variables, it adds them to the set of frontier variables.

The algorithm terminates when it has explored separation sets up to the maximum allowed cardinality $\mathbf{\Delta} \leq n$, resulting in a worse case complexity of $O(2^{\Delta})$. This termination condition means that certain variables that are separable are not separated: this does not result in false dependencies but potentially transitive dependencies may be considered direct dependencies. This sub-optimality does not affect the accuracy of the scenario datasets that *WISE* prepares, but it reduces the efficiency because it leaves the graph to be denser and the nodes having larger in-degree.

In the cases where the set of no-cause variables is unknown, *WISE* relies on the PC-algorithm [23], which also performs search

for separating nodes in the order of increasing cardinality among all pair of variables, but not using the frontier variables.

Orienting the Edges: *WISE* orients the edges and attempts to detect latent variables using the following simple rules, well known in the literature; we reproduce the rules here for convenience and refer the reader to [20] for further details.

1. Add outgoing edges from the no-cause variables.
2. If node c has nonadjacent neighbors a and b, and $c \in S_{ab}$, then orient edges $a \rightarrow c \leftarrow b$ (unmarked edges).
3. For all nonadjacent nodes, a, b with a common neighbor c, if there is an edge from a to c, but not from b to c, then add a marked edge $c \stackrel{*}{\rightarrow} b$.
4. If a and b are adjacent and there is directed path of only marked edges from a to b, then add $a \rightarrow b$

In the resulting graph, any unmarked, bi-directed, or undirected edges signify possible latent variables and *ambiguity in causal structure*. In particular, $a \rightarrow b$ means either a really causes b or there is a common latent cause L causing both a and b. Similarly, $a \leftrightarrow b$, signifies a definite common latent cause, and undirected edge between a and b implies either a causes b, b causes a, or a common latent cause in the underlying model.

Addressing False Causal Relationships: False or missing causal relationships can occur if the population in the dataset is not independent of the outcome variables. Unfortunately, because *WISE* relies on passive datasets this is a fundamental limitation that cannot be avoided. However, we expect that because the basic principles of computer networks are similar across the Internet, and the service providers use essentially the same versions of software throughout their networks, the bias in the dataset that would significantly affect the causal interpretation is not common. If such biases do exist, they will likely be among datasets from different geographical deployment regions. To catch such biases, we recommend using a training dataset with *WISE* that is obtained from different geographical locations. We can infer causal structure for each geographical region separately; if the learned structure is different, the differences must be carefully examined in light of the knowledge of systems internal working.

Lastly, while *WISE* depends on the CBN for preparing the scenario dataset, it is not necessary that the CBN is learned automatically from the dataset; the CBN can be supplied, entirely, or in part by a designer who is well-versed with the system.

5.3 Specifying the "What-If" Scenarios

Figure 4 shows the grammar for *WISE*-Specification Language (WSL). A scenario specification with WSL comprises a *use*-statement, followed by optional scenario *update*-statements.

The *use*-statement specifies a condition that describes the subset of present network for which the designer is interested in evaluating the scenario. This statement provides a powerful interface to the designer for choosing the baseline scenario: depending on the features available in the dataset, the designer can specify a subset of network based on location of clients (such as country, network address, or AS number), the location of servers, properties of service sessions, or a combination of these attributes.

The *update*-statements allow the designer to specify *what-if* values for various variables for the service session properties. Each scenario statement begins with either the INTERVENE, or the ASSUME keyword and allows conditional modification of exactly one variable in the dataset.

When the statement begins with the INTERVENE keyword, *WISE* first updates the value of the variable in question. *WISE* then uses the causal dependency structure to make the dataset *consistent* with the underlying dependencies. For this *WISE* uses a process called *Statistical Intervention Effect Evaluation* (Section 5.4).

```
scenario = use_stmt {update_stmt};
use_stmt = "USE" ("*" | condition_stmt)<EOL>;
update_stmt = ("ASSUME"|"INTERVENE") (set_directive |
    setdist_directive) [condition_stmt]<EOL>;
set_directive = "SET" ["RADIAL"* | "FIXED"]
    var set_op value;
setdist_directive = "SETDIST" feature
    dist_name([param])| "FILE" filename);
condition_clause = "WHERE" condition;
condition = simple_cond | compound_cond;
simple_cond = compare_clause | (simple_cond);
compound_cond = (simple_cond ("AND"|"OR")
    (simple_cond|compound_cond));
compare_clause = (var rel_op value) | member_test;
member_test = feature "IN" (value {,value});
set_op = "+=" | "-=" | "*=" | "\=" | "=";
rel_op = "<=" | ">=" | "<>" | "==" | "<" | ">";
var = a variable from the dataset;
```

Figure 4: Grammar for *WISE* Specification Language (WSL).

Advanced designers can override the intelligent update behavior by using the ASSUME keyword in the update statement. In this case *WISE* updates the distribution of the variable specified in the statement but does not attempt to ensure that the distribution of the dependent variables are correspondingly updated. *WISE* allows this functionality for cases where the designers believe that the scenario that they wish to evaluate involves changes to the underlying invariant laws that govern the system. Examples of scenario specification with WSL will follow in Section 7.

5.4 Preparing Representative Distribution for the "What-If" Scenarios

This section describes how *WISE* uses the dataset, the causal structure, and the scenario specification from the designer to prepare a meaningful dataset for the *what-if* scenario.

WISE first filters the global dataset for the entries that match the conditions specified in the *use-statement* of the scenario specification to create the *baseline* dataset. *WISE* then executes the update-statements, one statement at a time, to change the baseline dataset. To ensure consistency among variables after every INTERVENE update statement, *WISE* employs a process called *Statistical Intervention Effect Evaluation*; the process is described below.

Let us denote the action requested on a variable $\mathbf{x_i}$ in the *update*-statement as $\mathbf{set}(\mathbf{x_i})$. We refer to $\mathbf{x_i}$ as the *intervened* variable. Let us also denote the set of variables that are children of $\mathbf{x_i}$ in the CBN for the dataset as $\mathcal{C}(\mathbf{x_i})$. Then the statistical intervention effect evaluation process states that the new distribution of children of $\mathbf{x_i}$ is given as: $\mathbf{Pr}\{\mathcal{C}(\mathbf{x_i})|\mathbf{set}(\mathbf{x_i})\}$. The intuition is that because the parent node in a CBN has a causal effect on its descendent nodes, we expect that a change in the value of the parent variable must cause a change in the value of the children. Further, the new distribution of children variables would be one that we would expect to observe under the *changed* values of the parent variable.

To apply this process, *WISE* conditions the *global* dataset on the new value of the intervened variable, $\mathbf{set}(\mathbf{x_i})$, and the existing values of the all the other parents of the children of the intervened variable, $\mathcal{P}(\mathcal{C}(\mathbf{x_i}))$, in the baseline dataset to obtain an empirical distribution. *WISE* then assigns the children a random value from this distribution. *WISE* thus obtains a subset of the global dataset in which the distribution of $\mathcal{C}(\mathbf{x_i})$ is consistent with the action $\mathbf{set}(\mathbf{x_i})$ as well as the underlying dependencies.

Because the causal effect cascades to all the decedents of $\mathbf{x_i}$, *WISE* repeats this process recursively, considering $\mathcal{C}(\mathbf{x_i})$ as the in-

tervened variables and updating the distributions of $\mathcal{C}(\mathcal{C}(\mathbf{x_i}))$, and so on, until all the descendants of $\mathbf{x_i}$ (except the target variable) are updated. *WISE* cannot update the distribution of a descendant of $\mathbf{x_i}$ until the distribution of all of its ancestors that are descendant of $\mathbf{x_i}$ has been updated. *WISE* thus carefully orders the sequence of the updates by traversing the CBN DAG breadth-first, beginning at node $\mathbf{x_i}$.

WISE sequentially repeats this process for each statement in the scenario specification. The updated dataset produced after each statement serves as the input dataset for the next statement. Once all the statements are executed, the dataset is the representative joint distribution variables for the entire *what-if* scenario.

When the causal structure has ambiguities, *WISE* proceeds as follows. When the edge between two variables is undirected, *WISE* maintains the consistency by always updating the distribution of one if the distribution of the other is updated. For latent variables case, *WISE* assumes an imaginary variable, with directed edges to variables a and b and uses the resulting structure to traverse the graph while preparing the input distribution.

5.5 Estimating Response Time Distribution

Finding the new distribution of response time is also a case of intervention effect evaluation process. We use a non-parametric regression method to estimate the expected response-time distribution, instead of assigning a random value from the constrained empirical distribution as in the previous section, because the designers are interested in the expected values of the response time for each request. In particular, we use a standard Kernel Regression (KR) method, with a radial basis Kernel function (see [24, 26] for details) to estimate the response time for each request in the dataset. To address the computational complexity, *WISE* applies the KR in a piece-wise manner; the details follow in the next section.

5.6 Addressing the Computational Scalability

Because CDNs are complex systems, the response time may depend on a large number of variables, and the dataset might comprise hundreds of millions of requests spanning a large multi-dimensional space. To efficiently evaluate the *what-if* scenarios, *WISE* must address how to efficiently organize and utilize the dataset. In this section, we discuss our approach to these problems.

1. Curse of Dimensionality: As the number of dimensions (variables in the dataset) grow, exponentially more data is needed for similar accuracy of estimation. *WISE* uses the CBN to mitigate this problem. In particular, because when conditioned on its parents, a variable is independent of all variables except its descendants, we can use only the parents of the target variable in the regression function. Because the cardinality of the parent-set would typically be less than the total number of variables in the dataset, the accuracy of the regression function is significantly improved for a given amount of data. Due to this, *WISE* can afford to use fewer training data points with the regression function and still get good accuracy. Also, because the time complexity for the KR method is $\mathcal{O}(kn^3)$, with k variables and n points in the training dataset, *WISE*'s technique results in significant computational speedup.

2. Overfitting: The density of the dataset from a real deployment can be highly irregular; usually there are many points for combinations of variable values that represent the *normal* network operation, while the density of dataset is sparser for combinations that represent the fringe cases. Unfortunately, because the underlying principle of most regression techniques is to find parameters that minimize the errors on the training data, we can end up with parameters that minimize the error for high density regions of the dataset but give poor results in the fringes—this problem is called overfitting. The usual solution to this problem is introducing a smoothness penalty in the objective function of the regression method, but finding the right penalty function requires cross-validation, which is usually at least quadratic in the size of the *global* dataset[1]. In the case of CDNs, even one day of data may contain entries for millions of requests, which makes the quadratic complexity of these algorithms inherently unscalable.

WISE uses piece-wise regression to address this problem. *WISE* divides the dataset into small pieces, that we refer to as *tiles* and performs regression independently for each tile. *WISE* further prunes the *global* dataset to produce a training dataset so that the density of training points is more or less even across all the tiles.

To decompose the dataset, *WISE* uses fixed-size buckets for each dimension in the dataset for most of the variable value space. If the bucket sizes are sufficiently small, having more data points beyond a certain threshold does not contribute appreciably to the response-time prediction accuracy. With this in mind, *WISE* uses two thresholds n_{min} and n_{max} and proceeds with tile boundaries as follows. *WISE* decides on a bucket width b_i along each dimension i, and forms boundaries in the space of the dataset at integer multiples of b_i along each dimension; for categorical variables, *WISE* uses each category as a separate bucket. For each tile, *WISE* obtains a uniform random subset of n_{max} points that belong in the tile boundaries from the global dataset and adds them to the training dataset. If the dataset has fewer than n_{min} data points for the tile, the tile boundaries are folded to merge it with neighboring tile. This process is repeated until the tile has n_{min} number of points. Ultimately, most of the tiles have regular boundaries, but for some tiles, especially those on the fringes, the boundaries can be irregular. Once the preparation of training data is complete, we use cross-validation to derive regression parameters for each tile; the complexity is now only $O(n_{max}^2)$ for each tile.

3. Retrieval of Data: With large datasets, even the mundane tasks, such as retrieving training and test data during the input distribution preparation and response-time estimation phases are challenging. Quick data retrieval and processing is imperative here because both of these stages are *online*, in the sense that they are evaluated when the designer specifies the scenario.

WISE expedites this process by intelligently indexing the training data off-line and the test data as it is created. Tiles are used here as well: Each tile is assigned a *tile-id*, which is simply a string formed by concatenating the tile's boundaries in each dimension. All the data points that lie in the tile boundaries are assigned the tile-id as a key that is used for indexing. For the data preparation stage, *WISE* performs the tile-id assignment and indexing along the dimensions comprising the parents of most commonly used variables, and for the regression phase, the tiling and indexing is performed for the dimensions comprising the parents of the target variable. Because the tile-ids use fixed length bins for most of the space, mapping of a point to its tile can be performed in constant time for most of the data-points using simple arithmetic operations.

4. Parallelization and Batching: We have carefully designed the various stages in *WISE* to support parallelization and batching of jobs that use similar or same data. In the training data preparation stage, each entry in the dataset can be independently assigned its *tile-id* based key because *WISE* uses regular sized tiles. Similarly, the regression parameters for each tile can be learned independently

[1]Techniques such as in [10] can reduce the complexity for such N-body problems but are still quite complex than the approximations that *WISE* uses.

and in parallel. In the input data preparation stage, *WISE* batches the test and training data that belong in a tile and fetch the data from the training data for all of these together. Finally, because *WISE* uses piece-wise regression to evaluate the effects of intervention, it can batch the test and training data for each tile; further because the piece-wise computation are independent, they can take place in parallel.

6. IMPLEMENTATION

We have implemented *WISE* with the Map-Reduce framework [16] using the Sawzall logs processing language [22] and Python Map-Reduce libraries. We chose this framework to best exploit the parallelization and batching opportunities offered by the *WISE* design[2]. We have also implemented a fully-functional prototype for *WISE* using a Python front-end and a MySQL backend that can be used for small scale datasets. We provide a brief overview of the Map-Reduce based implementation here.

Most steps in *WISE* are implemented using a combination of one or more of the four Map-Reduce patterns shown in Figure 5. *WISE* uses *filter* pattern to obtain conditional subsets of dataset for various stages. *WISE* uses the *Tile-id Assignment* pattern for preparing the training data. We set the n_{min} and n_{max} thresholds to 20 and 50, respectively to achieve 2-5% confidence intervals. In the input data preparation phase, the *use*-statement is implemented using the *filter* pattern. The *update*-statements use *update* pattern for applying the new values to the variable in the statement. If the *update*-statement uses the INTERVENE keyword then *WISE* uses the *Training & Test Data Collation* pattern to bring together the relevant test and training data and update the distribution of the test data in a batched manner. Each *update*- statement is immediately followed by the *Tile-id Assignment* pattern because the changes in the value of the data may necessitate re-assignment of the tile-id. Finally, *WISE* uses the *Training & Test Data Collation* pattern for piece-wise regression. Our Map-Reduce based implementation can evaluate typical scenarios in about 5 minutes on a cluster of 50 PCs while using nearly 500 GB of training data.

7. EVALUATING WISE FOR A REAL CDN

In this section, we describe our experience applying *WISE* to a large dataset obtained from Google's global CDN for Web-search service. We start by briefly describing the CDN and the service architecture. We also describe the dataset from this CDN and the causal structure discovered using WCD. We also evaluate *WISE*'s ability to predict response-time distribution for the *what-if* scenarios.

7.1 Web-Search Service Architecture

Figure 6(a) shows Google's Web-search service architecture. The service comprises a system of globally distributed HTTP reverse proxies, referred to as Front End (FE) and a system of globally distributed clusters that house the Web servers and other core services (the Back End, or BE). A DNS based request redirection system redirects the user's queries to one of the FEs in the CDN. The FE process forwards the queries to the BE servers, which generate dynamic content based on the query. The FE caches static portions of typical reply, and starts transmitting that part to the requesting user as it waits for reply from the BE. Once the BE replies, the dynamic content is also transmitted to the user. The FE servers may or may not be co-located in the same data center with the BE

[2]Hadoop [11] provides an open-source Map-Reduce library. Modern data-warehousing appliances, such the ones by Netezza [18], can also exploit the parallelization in *WISE* design.

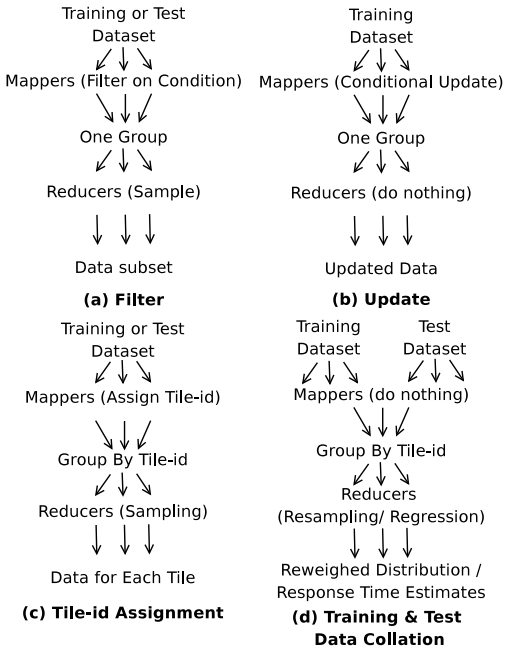


Figure 5: Map-Reduce patterns used in WISE implementation.

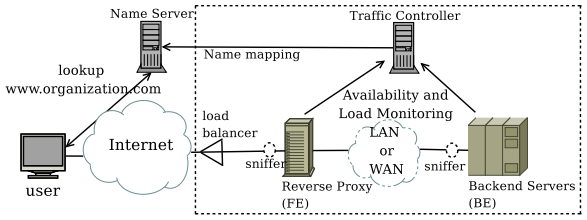
servers. If they are co-located, they can be considered to be on the same local area network and the round-trip latency between them is only a few milliseconds. Otherwise, the connectivity between the FE and the BE is typically on a well-provisioned connection on the public Internet. In this case the latency between the FE and BE can be several hundred milliseconds.

The *server response time* for a request is the time between the instance when the user issues the HTTP request and the instance when the last byte of the response is received by the users. We estimate this value as the sum of the round-trip time estimate obtained from the TCP three-way handshake, and the time between the instance when the request is received at the FE and when the last byte of the response is sent by the FE to user. The key contributors to server response time are: (i) the transfer latency of the request from the user to the FE (ii) the transfer latency of request to the BE and the transfer latency of sending the response from the BE to the FE; (iii) processing time at the BE, (iv) TCP transfer latency of the response from the FE to the client; and (v) any latency induced by loss and retransmission of TCP segments.

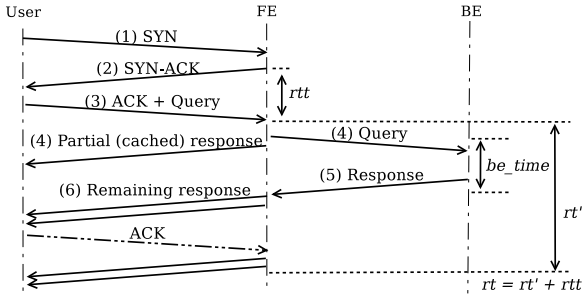
Figure 6(b) shows the process by which a user's Web search query is serviced. This message exchange has three features that affect service response time in subtle ways, making it hard to make accurate "back-of-the-envelop calculations" in the general case:

1. Asynchronous transfer of content to the user. Once the TCP handshake is complete, user's browser sends an HTTP request containing the query to the FE. While the FE waits on a reply from the BE, it sends some static content to the user; this content—essentially a "head start" on the transfer—is typically brief and constitutes only a couple of IP packets. Once the FE receives the response from the BE, it sends the response to the client and completes the request. A client may use the same TCP connection for subsequent HTTP requests.

2. Spliced TCP connections. FE processes maintain several TCP connections with the BE servers and reuse these connections for forwarding user requests to the BE. FE also supports HTTP pipelin-



(a) Google's Web-search Service Architecture



(b) Message Exchange

Figure 6: Google's Web-search service architecture and message exchange for a request on a *fresh* TCP connection.

ing, allowing the user to have multiple pending HTTP requests on the same TCP connection.

3. Spurious retransmissions and timeouts. Because most Web requests are short TCP transfers, the duration of the connection is not sufficient to estimate a good value for the TCP retransmit timer and many Web servers use default values for retransmits, or estimate the timeout value from the initial TCP handshake round-trip time. This causes spurious retransmits for users with slow access links and high serialization delays for MTU sized packets.

7.2 Data

We use data from an existing network monitoring infrastructure in Google's network. Each FE cluster has network-level sniffers, located between the FE and the load-balancers, that capture traffic and export streams in *tcpdump* format. A similar monitoring infrastructure captures traffic in the BE. Although the FE and BE servers use NTP for time synchronization, it is difficult to collate the traces from the two locations using only the timestamps. Instead, we use the hash of each client's IP, port and part of query along with the timestamp to collate the request between the FE and the BE. *WISE* then applies the relevance tests (ref. Sec. 5.1) on the features in the dataset collected in this manner. Table 1 describes the variables that *WISE* found to be relevant to the service response-time variable.

7.3 Causal Structure in the Dataset

To obtain the causal structure, we use a small sampled data subset collected on June 19, 2007, from several data center locations. This dataset has roughly 25 million requests, from clients in 12,877 unique ASes.

We seed the WCD algorithm with the *region* and *ts* variables as the *no-cause* variables. Figure 7 shows the causal structure that WCD produces. Most of the causal relationships in Figure 7 are straightforward and make intuitive sense in the context of networking, but a few relationships are quite surprising. WCD detects a relationship between the $region$ and sB attribute (the size of the result page); we found that this relationship exists due to the differences in the sizes of search response pages in different languages

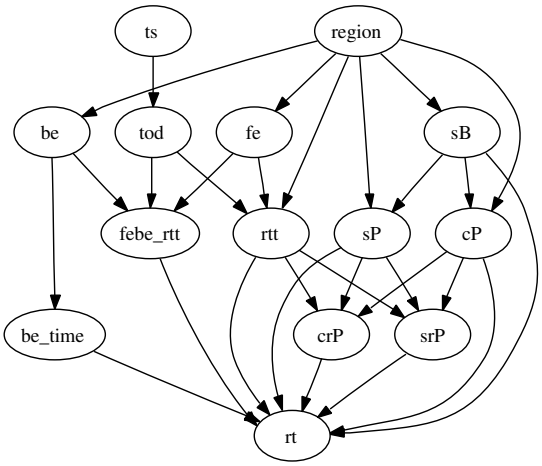


Figure 7: Inferred causal structure in the dataset. $A \rightarrow B$ **means A *causes* B.**

and regions. Another unexpected relationship is between $region$, cP and sP attributes; we found that this relationship exists due to different MTU sizes in different parts of the world. Our dataset, unfortunately, did not have load, utilization, or data center capacity variables that could have allowed us to model the be_time variable. All we observed was that the be_time distribution varied somewhat among the data centers. Overall, we find that WCD algorithm not only discovers relationships that are faithful to how networks operate but also discovers relationships that might escape trained network engineers.

Crucially, note that many variables are not direct children of the $region$, ts, fe or be variables. This means that when conditioned on the respective parents, these variables are independent of the region, time, choice of FE and BE, and we can use training data from past, different regions, and different FE and BE data centers to estimate the distributions for these features! Further, while most of the variables in the dataset are correlated, the in-degree for each variable is smaller than the total number of variables. This reduces the number of dimensions that *WISE* must consider for estimating the value of the variables during scenario evaluation, allowing *WISE* to produce accurate estimates, more quickly and with less data.

7.4 Response-Time Estimation Accuracy

Our primary metric for evaluation is *prediction accuracy*. There are two sources of error in response-time prediction: (i) error in response-time estimation function (Section 5.5) and (ii) inaccurate input, or error in estimating a valid input distribution that is representative of the scenario (Section 5.4). To isolate these errors, we first evaluate the estimation accuracy alone and later consider the overall accuracy for a complete scenario in Section 7.5.

To evaluate accuracy of the piece-wise regression method in isolation we can try to evaluate a scenario: "What-if I make no changes to the network?" This scenario is easy to specify with WSL by not including any optional scenario update statements. For example, a scenario specification with the following line:

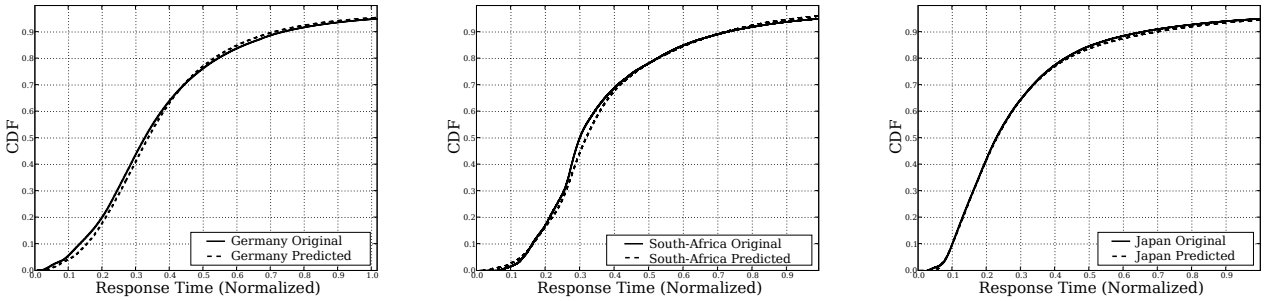
```
USE WHERE country==deu
```

would produce an input distribution for the response-time estimation function that is representative of users in Germany without any error and any inaccuracies that arise would be due to regression method. To demonstrate the prediction accuracy, we present results for three such scenarios:

(a) `USE WHERE country==deu`
(b) `USE WHERE country==zaf`

Feature	Description
ts, tod	A time-stamp of instance of arrival of the request at the FE. We also extract the hourly time-of-day (tod) from the timestamp.
sB	Number of bytes sent to the user from the server; this does not include any data that might be retransmitted or TCP/IP header bytes.
sP	Number of packets sent by the server to the client excluding retranmissions.
cP	Number of packets sent by the client that are received at the server, excluding retransmissions.
srP	Number of packets retransmitted by the server to the client, either due to loss, reordering, or timeouts at the server.
crP	Number of packets retransmitted by the client that are received at the server.
region	We map the IP address of the client to a region identifier at the country and state granularity. We also determine the /24(s24) and /16(s16) network addresses and the originating AS number. We collectively refer to these attributes as region.
fe, be	Alphanumeric identifiers for the FE data center at which the request was received and the BE data center that served the request.
rtt	Round-trip time between the user and FE estimated from the initial TCP three-way handshake.
febe_rtt	The network level round-trip time between the front end and the backend clusters.
be_time	Time between the instance that BE receives the request forwarded by the FE and when the BE sends the response to the FE.
rt	The response time for the request as seen by the FE (see Section 7.1). Response time is also our target variable.

Table 1: Features in the dataset from Google's CDN.



(a) Germany (b) South Africa (c) FE in Japan

Figure 8: Prediction accuracy: comparison of normalized response-time distributions for the scenarios in Section 7.4.

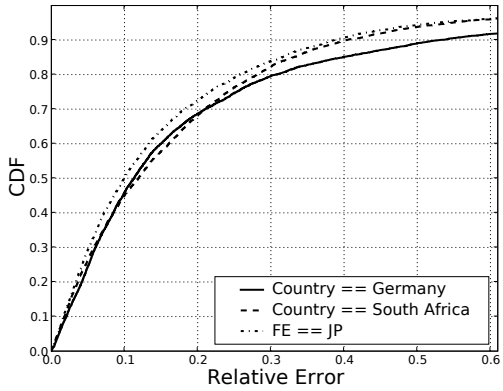


Figure 9: Relative prediction error for the scenarios in Figure 8.

(c) `USE WHERE fe==jp`

The first two scenarios specify estimating the response-time distribution for the users in Germany, and South Africa, respectively (deu and zaf are the country codes for these countries in our dataset), and the third scenario specifies estimating the response-time distribution for users that are served from FE in Japan (jp is the code for the FE data center in Japan); this data center primarily serves users in South and East Asia. We used the dataset from the week of June 17-23, 2007 as our training dataset and predicted the response time field for dataset for the week of June 24-30, 2007.

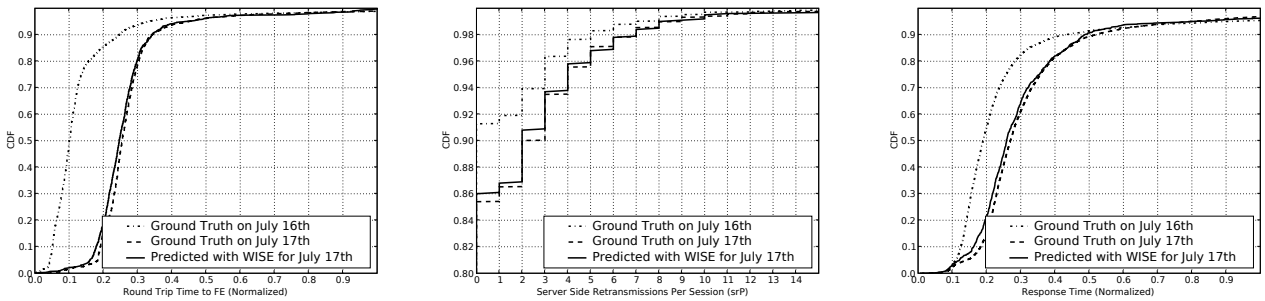
Figures 8(a), (b), and (c), show the results for the three scenarios, respectively. The ground-truth distribution for response time is based on the response-time values observed by the monitoring infrastructure for June 24-30 period, and is shown in solid line. The response time for the same period that *WISE* predicts is shown in a dotted line. Further, in Figure 9 we present the relative prediction error for these experiments. The error is defined as $|rt - \widehat{rt}|/rt$, where rt is the ground-truth value and $\widehat{rt}$ is the value that *WISE* predicts. The median error lies between 8-11%.

Dataset	***WISE***	AKM	CSA
deu	**18%**	55%	45%
zaf	**12%**	35%	120%
jp	**15%**	38%	50%

Table 2: Comparison of median relative error for response-time estimation for scenarios in Section 7.4.

Comparison with TCP Transfer Latency Estimation: It is reasonable to ask how well we could do using a simpler parametric model. To answer this question, we relate the problem of response-time estimation to that of estimating TCP-transfer latency, because Google uses TCP to transfer the data to clients. Many parametric models for TCP transfer latency are known; we have implemented two recent models by Arlitt [2] and Cardwell [5] for comparison. We refer to these methods as AKM and CSA, respectively. We modified the two methods to account for the additional latency that occurs due to the round-trip from the FE to the BE, and the time that the backend servers take (be_time). AKM uses a compensation multiplier called '*CompWeight*' to minimize the error for each trace. We computed the value of this weight that yielded minimum error. For the CSA scheme, we estimated the loss rate as the ratio of number of retransmitted packets and the total number of packets sent by the server in each of the three scenarios. We used the values of other model parameters that are correct to the best of our knowledge for each trace.

Table 2 presents the median error for requests that had at least one retransmitted packet in the response ($srP > 1$). The three datasets had 3.0%, 57.0%, and 9.1%, sessions with at least one retransmit, and the loss rates (calculated as described above) are



(a) Client-FE RTT (b) Server Side Retransmits (c) Response Time Distribution

Figure 10: Predicted distribution for response time and intermediary variables for the India data center drain scenario.

0.9%, 1.2%, 22.5%, and 3.2%, respectively. While the AKM method does not account for retransmits and losses, the compensation factor improves its accuracy. The CSA method models TCP more faithfully, but it is not accurate for short TCP transfers. The estimation error with *WISE* is at least 2.5-2.9 times better than the other scheme across different traces without using any trace specific compensation.

7.5 Evaluating a Live What-if Scenario

We evaluate how *WISE* predicts the response-time distribution for the affected set of customers during the scenario that we presented in Section 2 as a motivating example. In particular, we will focus on the effect of this event on customers of AS 9498, which is a large consumer ISP in India.

To appreciate the complexity of this scenario, consider what happens on the ground during this reconfiguration. First, because the FE in Taiwan is co-located with the BE, $febe\_rtt$ reduces to a typical intra-data center round-trip latency of 3ms. Also we observed that the average latency to the FE for the customers of AS 9498 increased by about 135ms as they were served from FE in Taiwan (`tw`) instead of the FE in India (`im`).

If the training dataset already contains the rtt estimates for customers in AS 9498 to the fe in Taiwan then we can write the scenario in two lines as following:

```
USE WHERE as_num==9498 AND fe==im AND be==tw
INTERVENE SET fe=tw
```

WISE uses the CBN to automatically update the scenario distribution. Because, fe variable is changed, *WISE* updates the distribution of children of fe variable, which in this case include $febe\_rtt$ and rtt variables. This in-turn causes a change in children of rtt variable, and similarly, the change cascades down to the rt variable in the DAG. In the case when such rtt is not included in the training dataset, the value can be explicitly provided as following:

```
USE WHERE as_num==9498 AND fe==im AND be==tw
INTERVENE SET febe_rtt=3
INTERVENE SET rtt+=135
```

We evaluated the scenario using the former specification. Figure 10 shows ground truth response-time distribution as well as distributions for intermediary variables (Figure 7) for users in AS 9498 between hours of 12 a.m. and 8 p.m. on July 16^{th} and the same hours on July 17^{th}, as well as the distribution estimated with *WISE* for July 17^{th} for these variables. We observe only slight underestimation of the distributions with *WISE*—this underestimation primarily arises due to insufficient training data to evaluate the variable distribution for the peripheries of the input distribution; *WISE* was not able to predict the response time for roughly 2% of the requests in the input distribution. Overall, maximum cumulative distribution differences[3] for the distributions of the three variables were between 7-9%.

8. CONTROLLED EXPERIMENTS

Because evaluating *WISE* on a live production network is limited by the nature of available datasets and variety of events with which we can corroborate, we have created a small-scale Web service environment using the Emulab testbed [7]. The environment comprises a host running the Apache Web server as the backend or BE, a host running the Squid Web proxy server as the frontend or FE, and a host running a multi-threaded wget Web client that issues request at an exponentially distributed inter-arrival time of 50 ms. The setup also includes delay nodes that use dummynet to control latency.

To emulate realistic conditions, we use a one-day trace from several data centers in Google's CDN that serve users in the USA. We configured the resource size distribution on the BE, as well as to emulate the wide area round-trip time on the delay node based on this trace.

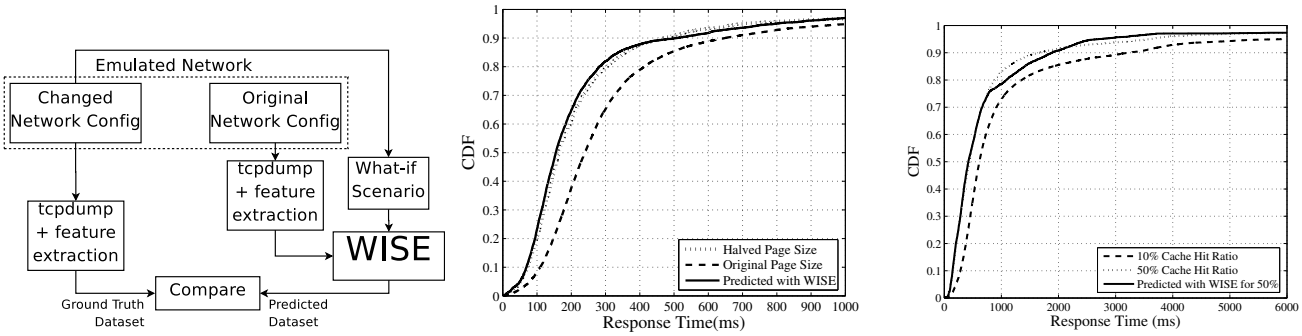
For each experiment we collect *tcpdump* data and process it to extract a feature set similar to one described in Table 1. We do not specifically emulate losses or retransmits because these occurred for fewer than 1% of requests in the USA trace. We have conducted two *what-if* scenario experiments in this environment; these are described below.

Experiment 1: Changing the Resource Size: For this experiment, we used just the backend server and the client machine, and used the delay node to emulate wide area network delays. We initially collected data using the resource size distribution based on the real trace for about two hours, and used this dataset as the training dataset. For the *what-if* scenario, we replaced all the resources on the server with resources that are half the size, and collected the test dataset for another two hours. We evaluated the test case with *WISE* using the following specification:

```
USE *
INTERVENE SET FIXED sB/=2
```

Figure 11(b) presents the response-time distribution for the original page size (dashed), the observed response-time distribution with halved page sizes (dotted), and the response-time distribution for the *what-if* scenario predicted with *WISE* using the original page size based dataset as input (solid). The maximum CDF distance in this case is only 4.7%, which occurs around the 40^{th} percentile.

[3]We could not use the relative error metric here because the requests in the input distribution prepared with *WISE* for *what-if* scenario cannot be pair-wise matched with ones in the ground-truth distribution; maximum distribution difference is a common metric used in statistical tests, such as Kolmogorov-Smirnov Goodness-of-Fit Test.



(a) Controlled Experiment Setup (b) Changing the Page Size (c) Changing the Cache Hit Ratio

Figure 11: Controlled *what-if* scenarios on Emulab testbed: experiment setup and results.

Experiment 2: Changing the Cache Hit Ratio: For this experiment, we introduced a host running a Squid proxy server to the network and configured the proxy to cache 10% of the resources uniformly at random. There is a delay node between the client and the proxy as well as the proxy and the backend server, each emulates trace driven latency as in the previous experiment. For the *what-if* scenario, we configured the Squid proxy to cache 50% of the resources uniformly at random. To evaluate this scenario, we include binary variable b_cached for each entry in the dataset that indicates whether the request was served by the caching proxy server or not. We use about 3 hours of trace with 10% caching as the training dataset, and use *WISE* to predict the response-time distribution for the case with 50% caching by using the following specification:

```
USE *
INTERVENE SETDIST b_cached FILE 50pcdist.txt
```

The `SETDIST` directive tells *WISE* to update the b_cached variable by randomly drawing from the empirical distribution specified in the file, which in this case contains 50% 1s and 50% 0s. Consequently, we intervene 50% of the requests to have a cached response.

Figure 11(c) shows the response-time distribution for the 10% cache-hit ratio (dashed), the response-time distribution with 50% cache-hit ratio (dotted), and the response-time distribution for the 50% caching predicted with *WISE* using the original 10% cache-hit ratio based dataset as input (solid). *WISE* predicts the response time quite well for up to the 80^{th} percentile, but there is some deviation for the higher percentiles. This occurred because the training dataset did not contain sufficient data for some of the very large resources or large network delays. The maximum CDF distance in this case is 4.9%, which occurs around 79^{th} percentile.

9. DISCUSSION

In this section, we discuss the limitations of and extensions to *WISE*. First we discuss what can and what cannot be predicted with *WISE*. We also describe issues related to parametric and non-parametric techniques. Lastly we discuss how the framework can be extended to other realms in networking.

What Can and Cannot Be Predicted: The class of *what-if* scenarios that can be evaluated with *WISE* depends entirely on the dataset that is available; in particular, *WISE* has two requirements:

First, *WISE* requires expressing the *what-if* scenario in terms of (1) variables in the dataset and (2) manipulation of those variables. At times, it is possible for dataset to capture the effect of the variable without capturing the variable itself. In such cases, *WISE* cannot evaluate any scenarios that require manipulation of that hidden variable. For example, the dataset from Google, presented earlier, does not include the TCP timeout variable even though this variable has an effect on response time. Consequently, *WISE* cannot evaluate a scenario that manipulates the TCP timeout.

Second, *WISE* also requires that the dataset contains values of variables that are similar to the values that represent the what-if scenario. If the global dataset does not have sufficient points in the space where the manipulated values of the variables lie, the prediction accuracy is affected, and *WISE* raises warnings during scenario evaluation.

WISE also makes stability assumptions, i.e., the causal dependencies remain unchanged under any values of intervention, and the underlying behavior of the system that determines the response times does not change. We believe that this assumption is reasonable as long as the fundamental protocols and methods that are used in the network do not change.

Parametric vs. Non-Parametric: *WISE* uses the assumption of functional dependency among variables to update the values for the variables during the statistical intervention evaluation process. In the present implementation, *WISE* only relies on non-parametric techniques for estimating this function but nothing in the *WISE* framework prevents using parametric functions. If the dependencies among some or all of the variables are parametric or deterministic, then we can improve *WISE*'s utility. Such a situation can in some cases allow *extrapolation* to predict variable values outside of what has been observed in the training dataset.

What-if Scenarios in other Realms of Networking: We believe that our work on evaluating what-if scenarios can be extended to incorporate other realms, such as routing, policy decisions, and security configurations by augmenting the reasoning systems with a decision evaluation system, such as *WISE*.

Our ultimate goal is to evaluate what-if scenarios for high-level goals, such as, "What if I deploy a new server at location X?", or better yet, "How should I configure my network to achieve certain goal?"; we believe that *WISE* is an important step in this direction.

10. RELATED WORK

We are unaware of any technique that uses *WISE*'s approach of answering *what-if* deployment questions, but *WISE* is similar to previous work on TCP throughput prediction and the application of Bayesian inference to networking problems.

A key component in response time for Web requests is TCP transfer latency. There has been significant work on TCP throughput and latency prediction using TCP modeling [2,5,19]. Due to inherent complexity of TCP these models make simplifying assumptions to keep the analysis tractable; these assumptions may produce inaccurate results. Recently, there has been effort to embrace

the complexity and using past behavior to predict TCP throughput. He et al. [12] evaluate predictability using short-term history, and Mirza et al. [17] use machine-learning techniques to estimate TCP throughput — these techniques tend to be more accurate. We also use machine-learning and statistical inference in our work, but techniques of [17] are not directly applicable because they rely on estimating path properties immediately before making a prediction. Further, they do not provide a framework for evaluating *what-if* scenarios. The parametric techniques, as we show in Section 7.4, unfortunately are not very accurate for predicting response-time.

A recent body of work has explored use of Bayesian inference for fault and root-cause diagnosis. SCORE [15] uses spatial correlation and shared risk group techniques to find the best possible explanation for observed faults in the network. Shrink [14] extends this model to a probabilistic setting, because the dependencies among the nodes may not be deterministic due to incomplete information or noisy measurements. Sherlock [4] additionally finds causes for poor performance and also models fail-over and load-balancing dependencies. Rish et al. [21] combine dependency graphs with active probing for fault-diagnosis. None of these work, however, address evaluating *what-if* scenarios for networks.

11. CONCLUSION

Network designers must routinely answer questions about how specific deployment scenarios affect the response time of a service. Without a rigorous method for evaluating such scenarios, the network designers must rely on ad hoc methods or resort to costly field deployments to test their ideas. This paper has presented *WISE*, a tool for specifying and accurately evaluating *what-if* deployment scenarios for content distribution networks. To our knowledge, *WISE* is the first tool to automatically derive causal relationships from Web traces and apply statistical intervention to predict networked service performance. Our evaluation demonstrates that *WISE* is both fast and accurate: it can predict response time distributions in "what if" scenarios to within a 11% error margin. *WISE* is also easy to use: its scenario specification language makes it easy to specify complex configurations in just a few lines of code.

In the future, we plan to use similar techniques to explore how causal inference can help network designers better understand the dependencies that transcend beyond just performance related issues in their networks. *WISE* represents an interesting point in the design space because it leverages almost no domain knowledge to derive causal dependencies; perhaps *what-if*scenario evaluators in other domains that rely almost exclusively on domain knowledge (e.g., [8]) could also leverage statistical techniques to improve accuracy and efficiency.

Acknowledgments

We would like to thank Andre Broido and Ben Helsley at Google, and anonymous reviewers for the valuable feedback that helped improve several aspects of our work. We would also like to thank Jeff Mogul for sharing source code for the methods in [2].

12. REFERENCES

[1] Akamai Technologies. www.akamai.com

[2] M. Arlitt, B. Krishnamurthy, J. Mogul. Predicting short-transfer latency from TCP arcana: A trace-based validation. IMC'2005.

[3] L.A. Barroso, J. Dean, U. Holzle. Web Search for a Planet: The Google Cluster Architecture. IEEE Micro. Vol. 23, No. 2. pp 22–28

[4] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. Maltz, M. Zhang. Towards Highly Reliable Enterprise Network Services via Inference of Multi-level Dependencies. ACM SIGCOMM 2007.

[5] N. Cardwell, S. Savage, T. Anderson. Modeling TCP Latency. IEEE Infocomm 2000.

[6] G. Cooper. A Simple Constraint-Based Algorithm for Efficiently Mining Observational Databases for Causal Relationships. Data Mining and Knowledge Discovery 1, 203-224. 1997.

[7] Emulab Network Testbed. http://www.emulab.net

[8] N. Feamster and J. Rexford. Network-Wide Prediction of BGP Routes. IEEE/ACM Transactions on Networking. Vol. 15. pp. 253–266

[9] M. Freedman, E. Freudenthal, D. Mazieres. Democratizing Content Publication with Coral. USENIX NSDI 2004.

[10] A. Gray, A. Moore, 'N -Body' Problems in Statistical Learning. Advances in Neural Information Processing Systems 13. 2000.

[11] Lucene Hadoop. http://lucene.apache.org/hadoop/

[12] Q. He, C. Dovrolis, M. Ammar. On the Predictability of Large Transfer TCP Throughput. ACM SIGCOMM 2006.

[13] A. Barbir, et al. Known Content Network Request Routing Mechanisms. IETF RFC 3568. July 2003.

[14] S. Kandula, D. Katabi, J. Vasseur. Shrink: A Tool for Failure Diagnosis in IP Networks. MineNet Workshop SIGCOMM 2005.

[15] R. Kompella, J. Yates, A. Greenberg, A. Snoeren. IP Fault Localization Via Risk Modeling. USENIX NSDI 2005.

[16] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. USENIX OSDI 2004.

[17] M. Mirza, J. Sommers, P. Barford, X. Zhu. A Machine Learning Approach to TCP Throughput Prediction. ACM SIGMETRICS 2007.

[18] Netezza http://www.netezza.com/

[19] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. IEEE/ACM Transactions on Networking. Vol 8. pp. 135-145

[20] J. Pearl. Causality: Models, Reasoning, and Inference. Cambridge University Press. 2003.

[21] I. Rish, M. Brodie, S. Ma. Efficient Fault Diagnosis Using Probing. AAAI Spring Symposium on DMDP. 2002.

[22] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the Data: Parallel Analysis with Sawzall. Scientific Programming Journal. Vol. 13. pp. 227–298.

[23] P. Sprites, C. Glymour. An Algorithm for fast recovery of sparse causal graphs. Social Science Computer Review 9. USENIX Symposium on Internet Technologies and Systems. 1997.

[24] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, M. Ammar. Answering "What-if" Deployment and Configuration Questions with WISE. Georgia Tech Technical Report GT-CS-08-02. February 2008.

[25] L. Wasserman. All of Statistics: A Concise Course in Statistical Inference. Springer Texts in Statistics. 2003.

[26] J. Wolberg. Data Analysis Using the Method of Least Squares. Springer. Feb 2006.

To appear in ACM SIGCOMM
New Delhi, India. August 2010.

SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware

Muhammad Bilal Anwer, Murtaza Motiwala, Mukarram bin Tariq, Nick Feamster
School of Computer Science, Georgia Tech

ABSTRACT

We present SwitchBlade, a platform for rapidly deploying custom protocols on programmable hardware. SwitchBlade uses a pipeline-based design that allows individual hardware modules to be enabled or disabled on the fly, integrates software exception handling, and provides support for forwarding based on custom header fields. SwitchBlade's ease of programmability and wire-speed performance enables rapid prototyping of custom data-plane functions that can be directly deployed in a production network. SwitchBlade integrates common packet-processing functions as hardware modules, enabling different protocols to use these functions without having to resynthesize hardware. SwitchBlade's customizable forwarding engine supports both longest-prefix matching in the packet header and exact matching on a hash value. SwitchBlade's software exceptions can be invoked based on either packet or flow-based rules and updated quickly at runtime, thus making it easy to integrate more flexible forwarding function into the pipeline. SwitchBlade also allows multiple custom data planes to operate in parallel on the same physical hardware, while providing complete isolation for protocols running in parallel. We implemented SwitchBlade using NetFPGA board, but SwitchBlade can be implemented with any FPGA. To demonstrate SwitchBlade's flexibility, we use SwitchBlade to implement and evaluate a variety of custom network protocols: we present instances of IPv4, IPv6, Path Splicing, and an OpenFlow switch, all running in parallel while forwarding packets at line rate.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design C.2.5 [Computer-Communication Networks]: Local and Wide-Area Networks C.2.6 [Computer-Communication Networks]: Internetworking

General Terms: Algorithms, Design, Experimentation, Performance

Keywords: Network Virtualization, NetFPGA

SIGCOMM 2010, August 30-September 3, 2010, New Delhi, India.

1. INTRODUCTION

Countless next-generation networking protocols at various layers of the protocol stack require data-plane modifications. The past few years alone have seen proposals at multiple layers of the protocol stack for improving routing in data centers, improving availability, providing greater security, and so forth [3, 13, 17, 24]. These protocols must ultimately operate at acceptable speeds in production networks—perhaps even alongside one another—which raises the need for a platform that can support fast hardware implementations of these protocols running in parallel. This platform must provide mechanisms to deploy these new network protocols, header formats, and functions quickly, yet still forward traffic as quickly as possible. Unfortunately, the conventional hardware implementation and deployment path on custom ASICs incurs a long development cycle, and custom protocols may also consume precious space on the ASIC. Software-defined networking paradigms (e.g., Click [7, 16]) offer some hope for rapid prototyping and deployment, but a purely software-based approach cannot satisfy the strict performance requirements of most modern networks. The networking community needs a development and deployment platform that offers high performance, flexibility, and the possibility of rapid prototyping and deployment.

Although other platforms have recognized the need for fast, programmable routers, they stop somewhat short of providing a programmable platform for rapid prototyping on the hardware itself. Platforms that are based on network processors can achieve fast forwarding performance [22], but network processor-based implementations are difficult to port across different processor architectures, and customization can be difficult if the function that a protocol requires is not native to the network processor's instruction set. All other functions should be implemented in software. PLUG [8] is an excellent framework for implementing modular lookup modules, but the model focuses on manufacturing high-speed chips, which is costly and can have a long development cycle. RouteBricks [12] provides a high-performance router, but it is implemented entirely in software, which may introduce scalability issues; additionally, prototypes developed on RouteBricks cannot be easily ported to hardware.

This paper presents SwitchBlade, a programmable hardware platform that strikes a balance between the programmability of software and the performance of hardware, and enables rapid prototyping and deployment of new protocols. SwitchBlade enables rapid deployment of new protocols on hardware by providing modular building blocks to afford customizability and programmability that is sufficient for implementing a variety of data-plane functions. SwitchBlade's ease of programmability and wire-speed performance enables rapid prototyping of custom data-plane functions that can be directly deployed in a production network. SwitchBlade

relies on field-programmable gate arrays (FPGAs). Designing and implementing SwitchBlade poses several challenges:

- *Design and implementation of a customizable hardware pipeline.* To minimize the need for resynthesizing hardware, which can be prohibitive if multiple parties are sharing it, SwitchBlade's packet-processing pipeline includes hardware modules that implement common data-plane functions. New protocols can select a subset of these modules on the fly, without resynthesizing hardware.
- *Seamless support for software exceptions.* If custom processing elements cannot be implemented in hardware (*e.g.*, due to limited resources on the hardware, such as area on the chip), SwitchBlade must be able to invoke software routines for processing. SwitchBlade's hardware pipeline can directly invoke software exceptions on either packet or flow-based rules. The results of software processing (e.g., forwarding decisions), can be cached in hardware, making exception handling more efficient.
- *Resource isolation for simultaneous data-plane pipelines.* Multiple protocols may run in parallel on same hardware; we call each data plane a Virtual Data Plane (VDP). SwitchBlade provides each VDP with separate forwarding tables and dedicated resources. Software exceptions are the VDP that generated the exception, which makes it easier to build virtual control planes on top of SwitchBlade.
- *Hardware processing of custom, non-IP headers.* SwitchBlade provides modules to obtain appropriate fields from packet headers as input to forwarding decisions. SwitchBlade can forward packets using longest-prefix match on 32-bit header fields, an exact match on fixed length header field, or a bitmap added by custom packet preprocessing modules.

The design of SwitchBlade presents additional challenges, such as (1) dividing function between hardware and software given limited hardware resources; (2) abstracting physical ports and input/output queues; (3) achieving rate control on per-VDP basis instead of per-port basis; and (4) providing a clean interface to software.

We have implemented SwitchBlade using the NetFPGA board [2], but SwitchBlade can be implemented with any FPGA. To demonstrate SwitchBlade's flexibility, we use SwitchBlade to implement and evaluate several custom network protocols. We present instances of IPv4, IPv6, Path Splicing, and an OpenFlow switch, all of which can run in parallel and forward packets at line rate; each of these implementations required only modest additional development effort. SwitchBlade also provides seamless integration with software handlers implemented using Click [16], and with router slices running in OpenVZ containers [20]. Our evaluation shows that SwitchBlade can forward traffic for custom data planes—including non-IP protocols—at hardware forwarding rates. SwitchBlade can also forward traffic for multiple distinct custom data planes in parallel, providing resource isolation for each. An implementation of SwitchBlade on the NetFPGA platform for four parallel data planes fits easily on today's NetFPGA platform; hardware trends will improve this capacity in the future. SwitchBlade can support additional VDPs with less than a linear increase in resource use, so the design will scale as FPGA capacity continues to increase.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 explains our design goals and the key resulting design decisions. Section 4 explains the SwitchBlade design, and Section 5 describes the implementation of SwitchBlade, as well as our implementations of three custom data planes on SwitchBlade. Section 6 presents performance results. Section 7 briefly describes how we have implemented a virtual router on top of SwitchBlade using OpenVZ. We discuss various extensions in Section 8 and conclude in Section 9.

2. RELATED WORK

We survey related work on programmable data planes in both software and hardware.

The Click [16] modular router allows easy, rapid development of custom protocols and packet forwarding operations in software; kernel-based packet forwarding can operate at high speeds but cannot keep up with hardware for small packet sizes. An off-the-shelf NetFPGA-based router can forward traffic at 4 Gbps; this forwarding speed can scale by increasing the number of NetFPGA cards, and development trends suggest that much higher rates will be possible in the near future. RouteBricks [12] uses commodity processors to achieve software-based packet processing at high speed. The design requires significant PCIe interconnection bandwidth to allow packet processing at CPUs instead of on the network cards themselves. As more network interface cards are added, and as traffic rates increase, however, some packet processing may need to be performed on the network cards themselves to keep pace with increasing line speeds and to avoid creating bottlenecks on the interconnect.

Supercharged PlanetLab (SPP) [22] is a network processor (NP)-based technology. SPP uses Intel IXP network processors [14] for data-plane packet processing. NP-based implementations are specifically bound to the respective vendor-provided platform, which can inherently limit the flexibility of data-plane implementations.

Another solution to achieve wire-speed performance is developing custom high-speed networking chips. PLUG [8] provides a programming model for manufacturing chips to perform high-speed and flexible packet lookup, but it does not provide an off-the-shelf solution. Additionally, chip manufacturing is expensive: fabrication plants are not common, and cost-effective manufacturing at third-party facilities requires critical mass of demand. Thus, this development path may only make sense for large enterprises and for protocols that have already gained broad acceptance. Chip manufacturing also has a high turnaround time and post-manufacturing verification processes which can impede development of new protocols that need small development cycle and rapid deployment.

SwitchBlade is an FPGA-based platform and can be implemented on any FPGA. Its design and implementation draws inspiration from our earlier work on designing an FPGA-based data plane for virtual routers [4]. FPGA-based designs are not tied to any single vendor, and it scales as new, faster and bigger FPGAs become available. FPGAs also provide a faster development and deployment cycle compared to chip manufacturing.

Casado *et al.* argue for simple but high-speed hardware with clean interfaces with software that facilitate independent development of protocols and network hardware [9]. They argue that complex routing decisions can be made in software and cached in hardware for high-speed processing; in some sense, SwitchBlade's caching of forwarding decisions that are handled by software exception handlers embodies this philosophy. OpenFlow [19] enables the rapid development of a variety of protocols, but the division of functions between hardware and software in SwitchBlade is quite different. Both OpenFlow and SwitchBlade provide software exceptions and caching of software decisions in hardware, but SwitchBlade also provides selectable hardware preprocessing modules that effectively moves more flexible processing to hard-

ware. SwitchBlade also easily accommodates new hardware modules, while OpenFlow does not.

SwitchBlade provides wire-speed support for parallel customized data planes, isolation between them, and their interfacing with virtualization software, which would make SwitchBlade a suitable data plane for a virtual router. OpenFlow does not directly support multiple custom data planes operating in parallel. FlowVisor [1] provides some level of virtualization but sits between the OpenFlow switch and controller, essentially requiring virtualization to occur in software.

3. GOALS AND DESIGN DECISIONS

The primary goal of SwitchBlade is to enable *rapid development and deployment of new protocols working at wire-speed*. The three subgoals, in order of priority, are: (1) Enable rapid development and deployment of new protocols; (2) Provide customizability and programmability while maintaining wire-speed performance; and (3) Allow multiple data planes to operate in parallel, and facilitate sharing of hardware resources across those multiple data planes. In this section, we describe these design goals, their rationale, and highlight specific design choices that we made in SwitchBlade to achieve these goals.

Goal #1. Rapid development and deployment on fast hardware. Many next-generation networking protocols require data-plane modifications. Implementing these modifications entirely in software results in a slow data path that offers poor forwarding performance. As a result, these protocols cannot be evaluated at the data rates of production networks, nor can they be easily transferred to production network devices and systems.

Our goal is to provide a platform for designers to quickly deploy, test, and improve their designs with wire-speed performance. This goal influences our decision to implement SwitchBlade using FPGAs, which are programmable, provide acceptable speeds, and are not tied to specific vendors. An FPGA-based solution can allow network protocol designs to take advantage of hardware trends, as larger and faster FPGAs become available. SwitchBlade relies on programmable hardware, but incorporates software exception handling for special cases; a purely software-based solution cannot provide acceptable forwarding performance. From the hardware perspective, custom ASICs incur a long development cycle, so they do not satisfy the goal of rapid deployment. Network processors offer speed, but they do not permit hardware-level customization.

Goal #2. Customizability and programmability. New protocols often require specific customizations to the data plane. Thus, SwitchBlade must provide a platform that affords enough customization to facilitate the implementation and deployment of new protocols.

Providing customizability along with fast turnaround time for hardware-based implementations is challenging: a bare-bones FPGA is customizable, but programming from scratch has a high turnaround time. To reconcile this conflict, SwitchBlade recognizes that even custom protocols share common data-plane extensions. For example, many routing protocols might use longest prefix or exact match for forwarding, and checksum verification and update, although different protocols may use these extensions on different fields on in the packets. SwitchBlade provides a rich set of common extensions as modules and allows protocols to dynamically select any subset of modules that they need. SwitchBlade's modules are programmable and can operate on arbitrary offsets within packet headers.

Feature	Design Goals	Pipeline Stages
Virtual Data Plane (§ 4.2)	Parallel custom data planes	VDP selection
Customizable hardware modules (§ 4.3)	Rapid programming, customizability	Preprocessing, Forwarding
Flexible matching in forwarding (§ 4.4)	Customizability	Forwarding
Programmable software exceptions (§ 4.5)	Rapid programming, customizability	Forwarding

Table 1: SwitchBlade design features.

For extensions that are not included in SwitchBlade, protocols can either add new modules in hardware or implement exception handlers in software. SwitchBlade provides hardware caching for forwarding decisions made by these exception handlers to reduce performance overhead.

Goal #3. Parallel custom data planes on a common hardware platform. The increasing need for data-plane customization for emerging network protocols makes it necessary to design a platform that can support the operation of several custom data planes that operate simultaneously and in parallel on the same hardware platform. SwitchBlade's design identifies functions that are common across data-plane protocols and provides those implementations shared access to the hardware logic that provides those common functions.

SwitchBlade allows customized data planes to run in parallel. Each data plane is called a Virtual Data Plane (VDP). SwitchBlade provides separate forwarding tables and virtualized interfaces to each VDP. SwitchBlade provides isolation among VDP using per-VDP rate control. VDPs may share modules, but to preserve hardware resources, shared modules are not replicated on the FPGA. SwitchBlade ensures that the data planes do not interface even though they share hardware modules.

Existing platforms satisfy some or all of these goals, but they do not address all the goals at once or with the prioritization we have outlined above. For example, SwitchBlade trades off higher customizability in hardware for easier and faster deployability by providing a well-defined but modular customizable pipeline. Similarly, while SwitchBlade provides parallel data planes, it still gives each data plane direct access to the hardware, and allows each VDP access to a common set of hardware modules. This level of sharing still allows protocol designers enough isolation to implement a variety of protocols and systems; for example, in Section 7, we will see that designers can run virtual control planes and virtual environments (e.g., OpenVZ [20], Trellis [6]) on top of SwitchBlade.

4. DESIGN

SwitchBlade has several unique design features that enable rapid development of customized routing protocols with wire-speed performance. SwitchBlade has a pipelined architecture (§4.1) with various processing stages. SwitchBlade implements Virtual Data Planes (VDP) (§4.2) so that multiple data plane implementations can be supported on the same platform with performance isolation between the different forwarding protocols. SwitchBlade provides customizable hardware modules (§4.3) that can be enabled or disabled to customize packet processing at runtime. SwitchBlade implements a flexible matching forwarding engine (§4.4) that provides a longest prefix match and an exact hash-based lookup on

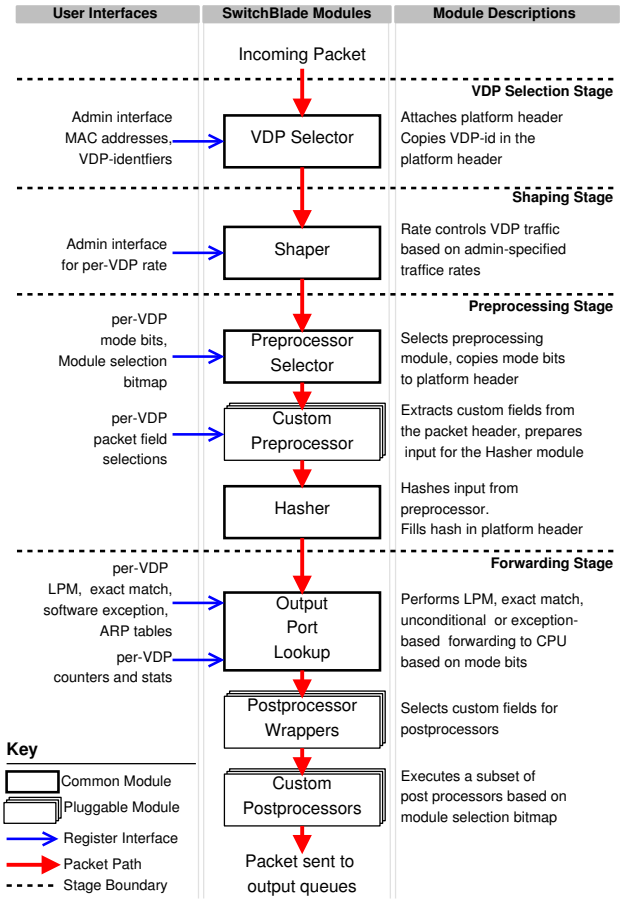


Figure 1: SwitchBlade Packet Processing Pipeline.

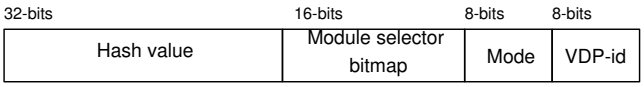


Figure 2: Platform header format. This 64 bit header is applied to every incoming packet and removed before the packet is forwarded.

various fields in the packet header. There are also programmable software exceptions (§4.5) that can be configured from software to direct individual packets or flows to the CPU for additional processing.

4.1 SwitchBlade Pipeline

Figure 1 shows the SwitchBlade pipeline. There are four main stages in the pipeline. Each stage consists of one or more hardware modules. We use a pipelined architecture because it is the most straightforward choice in hardware-based architectures. Additionally, SwitchBlade is based on reference router from the NetFPGA group at Stanford [2]; this reference router has a pipelined architecture as well.

VDP Selection Stage. An incoming packet to SwitchBlade is associated with one of the VDPs. The *VDP Selector* module classifies the packet based on its MAC address and uses a stored table that maps MAC addresses to *VDP identifiers*. A register interface populates the table with the VDP identifiers and is described later.

Field	Value	Description/Action
Mode	0	Default, Perform LPM on IPv4 destination address
	1	Perform exact matching on hash value
	2	Send packet to software for custom processing
	3	Lookup hash in software exceptions table
Module Selector Bitmap	1	Source MAC not updated
	2	Don't decrement TTL
	4	Don't Calculate Checksum
	8	Dest. MAC not updated
	16	Update IPv6 Hop Limit
	32	Use Custom Module 1
	64	Use Custom Module 2
	128	Use Custom Module 3

Table 2: Platform Header: The Mode field selects the forwarding mechanism employed by the Output Port Lookup module. The Module Selector Bitmap selects the appropriate postprocessing modules.

This stage also attaches a 64-bit *platform header* on the incoming packet, as shown in Figure 2. The registers corresponding to each VDP are used to fill the various fields in the platform header. SwitchBlade is a pipelined architecture, so we use a specific header format that to make the architecture extensible. The first byte of this header is used to select the VDP for every incoming packet. Table 2 describes the functionality of the different fields in the platform header.

Shaping Stage. After a packet is designated to a particular VDP, the packet is sent to the shaper module. The shaper module rate limits traffic on per VDP basis. There is a register interface for the module that specifies the traffic rate limits for each VDP.

Preprocessing Stage. This stage includes all the VDP-specific preprocessing hardware modules. Each VDP can customize which preprocessor module in this stage to use for preprocessing the packet via a register interface . In addition to selecting the preprocessor, a VDP can select the various bit fields from the preprocessor using a register interface. A register interface provides information about the mode bits and the preprocessing module configurations. In addition to the custom preprocessing of the packet, this stage also has the hasher module, which can compute a hash of an arbitrary set of bits in the packet header and insert the value of the hash in the packet's platform header.

Forwarding Stage. This final stage in the pipeline handles the operations related to the actual packet forwarding. The *Output Port Lookup* module determines the destination of the packet, which could be one of: (1) longest-prefix match on the packet's destination address field to determine the output port; (2) exact matching on the hash value in the packet's platform header to determine the output port; or (3) exception-based forwarding to the CPU for further processing. This stage uses the mode bits specified in the preprocessing stage. The *Postprocessor Wrappers* and the *Custom Postprocessors* perform operations such as decrementing the packet's time-to-live field. After this stage, SwitchBlade queues the packet in the appropriate output queue for forwarding. SwitchBlade selects the postprocessing module or modules based on the *module selection bits* in the packet's platform header.

4.2 Custom Virtual Data Plane (VDP)

SwitchBlade enables multiple customized data planes top operate simultaneously in parallel on the same hardware. We refer to each data plane as Virtual Data Plane (VDP). SwitchBlade provides a separate packet processing pipeline, as well as separate lookup ta-

bles and register interfaces for each VDP. Each VDP may provide custom modules or share modules with other VDPs. With SwitchBlade, shared modules are not replicated on the hardware, saving valuable resources. Software exceptions include VDP identifiers, making it easy to use separate software handlers for each VDP.

Traffic Shaping. The performance of a VDP should not be affected by the presence of other VDPs. The *shaper* module enables SwitchBlade to limit bandwidth utilization of different VDPs. When several VDPs are sharing the platform, they can send traffic through any of the four ports of the VDP to be sent out from any of the four router ports. Since a VDP can start sending more traffic than what is its bandwidth limit thus affecting the performance of other VDPs. In our implementation, the shaper module comes after the *Preprocessing* stage not before it as shown in Figure 1. This implementation choice, although convenient, does not affect our results because the FPGA data plane can process packets faster than any of the inputs. Hence, the traffic shaping does not really matter. We expect, however, that in the future FPGAs there might be much more than the current four network interfaces for a single NetFPGA which would make traffic shaping of individual VDPs necessary. In the existing implementation, packets arriving at a rate greater than the allocated limit for a VDP are dropped immediately. We made this decision to save memory resources on the FPGA and to prevent any VDP from abusing resources.

Register interface. SwitchBlade provides a register interface for a VDP to control the selection of preprocessing modules, to customize packet processing modules (*e.g.*, which fields to use for calculating hash), and to set rate limits in the shaper module. Some of the values in the registers are accessible by each VDP, while others are only available for the SwitchBlade administrator. SwitchBlade divides the register interfaces into these two security modes: the *admin* mode and the *VDP* mode. The admin mode allows setting of global policies such as traffic shaping, while the VDP mode is for per-VDP module customization.

SwitchBlade modules also provide statistics, which are recorded in the registers and are accessible via the admin interface. The statistics are specific to each module; for example, the VDP selector module can provide statistics on packets accepted or dropped. The admin mode provides access to all registers on the SwitchBlade platform, whereas the VDPmode is only to registers related to a single VDP.

4.3 Customizable Hardware Modules

Rapidly deploying new routing protocols may require custom packet processing. Implementing each routing protocol from scratch can significantly increase development time. There is a significant implementation cycle for implementing hardware modules; this cycle includes design, coding, regression tests, and finally synthesis of the module on hardware. Fortunately, many basic operations are common among different forwarding mechanisms, such as extracting the destination address for lookup, checksum calculation, and TTL decrement. This commonality presents an opportunity for a design that can reuse and even allow sharing the implementations of basic operations which can significantly shorten development cycles and also save precious resources on the FPGA.

SwitchBlade achieves this reuse by providing modules that support a few basic packet processing operations that are common across many common forwarding mechanism. Because SwitchBlade provides these modules as part of its base implementation, data plane protocols that can be composed from only the base modules can be implemented without resynthesizing the hardware and can be programmed purely using a register interface. As an example, to implement a new routing protocol such as Path Splicing [17], which requires manipulation of splicing bits (a custom field in the packet header), a VDP can provide a new module that is included at synthesis time. This module can append preprocessing headers that are later used by SwitchBlade's forwarding engine. A protocol such as OpenFlow [19] may depend only on modules that are already synthesized on the SwitchBlade platform, so it can choose the subset of modules that it needs.

SwitchBlade's reusable modules enable new protocol developers to focus more on the protocol implementation. The developer needs to focus only on bit extraction for custom forwarding. Each pluggable module must still follow the overall timing constraints, but for development and verification purposes, the protocol developer's job is reduced to the module's implementation. Adding new modules or algorithms that offer new functionality of course requires conventional hardware development and must still strictly follow the platform's overall timing constraints.

A challenge with reusing modules is that different VDPs may need the same postprocessing module (*e.g.*, decrementing TTL), but the postprocessing module may need to operate on different locations in the packet header for different protocols. In a naïve implementation, SwitchBlade would have to implement two separate modules, each looking up the corresponding bits in the packet header. This approach doubles the implementation effort and also wastes resources on the FPGA. To address this challenge, SwitchBlade allows a developer to include *wrapper modules* that can customize the behavior of existing modules, within same data word and for same length of data to be operated upon.

As shown in Figure 1 custom modules can be used in the preprocessing and forwarding stages. In the preprocessing stage, the customized modules can be selected by a VDP by specifying the appropriate selection using the register interface. Figure 3 shows an example: the incoming packet from the previous shaping stage which goes to a demultiplexer which selects the appropriate module or modules for the packet based on the input from the register interface specific to the particular VDP that the packet belongs to. After being processed by one of the protocol modules (*e.g.*, IPv6, OpenFlow), the packet arrives at the hasher module. The hasher module takes 256 bits as input and generates a 32-bit hash of the input. The hasher module need not be restricted to 256 bits of input data, but a larger input data bus would mean using more resources. Therefore, we decided to implement a 256-bit wide hash data bus to accommodate our design on the NetFPGA.

Each VDP can also use custom modules in the forwarding stage, by selecting the appropriate postprocessor wrappers and custom postprocessor modules as shown in Figure 1. SwitchBlade selects these modules based on the *module selection bitmap* in the platform header of the packet. Figure 4(b) shows an example of the custom wrapper and postprocessor module selection operation.

4.4 Flexible Matching for Forwarding

New routing protocols often require customized routing tables, or forwarding decisions on customized fields in the packet. For example, Path Splicing requires multiple IP-based forwarding tables, and the router chooses one of them based on splicing bits in the packet header. SEATTLE [15] and Portland [18] use MAC address-based forwarding. Some of the forwarding mechanisms are still simple enough to be implemented in hardware and can benefit from fast-path forwarding; others might be more complicated and it might be easier to just have the forwarding decision be made in software. Ideally, all forwarding should take place in hardware, but there is a tradeoff in terms of forwarding performance and hardware implementation complexity.

Preprocessor Selection		
Code	Processor	Description
1	Custom Extractor	Allows selection of variable 64-bit fields in packet on 64-bit boundaries in first 32 bytes
2	OpenFlow	OpenFlow packet processor that allows variable field selection.
3	Path Splicing	Extracts Destination IP Address and uses bits in packet to select the Path/Forwarding Table.
4	IPv6	Extracts IPv6 destination address.

Table 3: Processor Selection Codes.

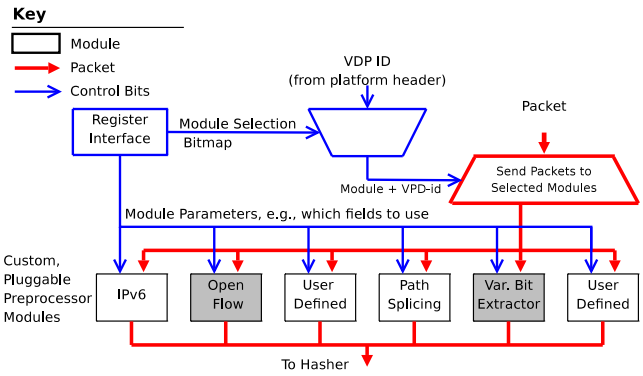


Figure 3: Virtualized, Pluggable Module for Programmable Processors.

SwitchBlade uses a hybrid hardware-software approach to strike a balance between forwarding performance and implementation complexity. Specifically, SwitchBlade's forwarding mechanism implementation, provided by the *Output Port Lookup* module as shown in Figure 1, provides the following four different methods for making forwarding decision on the packet: (1) conventional longest prefix matching (LPM) on any 32-bit address field in the packet header within the first 40-bytes; (2) exact matching on hash value stored in the packet's platform header; (3) unconditionally sending the packet to the CPU for making the forwarding computation; and (4) sending only packets which match certain user defined exceptions, called software exceptions 4.5, to the CPU. The details of how the output port lookup module performs these tasks is illustrated in Figure 4(a). Modes (1) and (2) enable fast-path packet forwarding because the packet never leaves the FPGA. We observe that many common routing protocols can be implemented with these two forwarding mechanisms alone. Figure 4 is not the actual implementation but shows the functional aspect of SwitchBlade's implementation.

By default, SwitchBlade performs a longest-prefix match, assuming an IPv4 destination address is present in the packet header. To enable use of customized lookup, a VDP can set the appropriate mode bit in the *platform header* of the incoming packet. One of the four different forwarding mechanisms can be invoked for the packet by the mode bits as described in Table 2. The output port lookup module performs LPM and exact matching on the hash value from the forwarding table stored in the TCAM. The same TCAM is used for LPM and for exact matching for hashing therefore the mask from the user decides the nature of match being done.

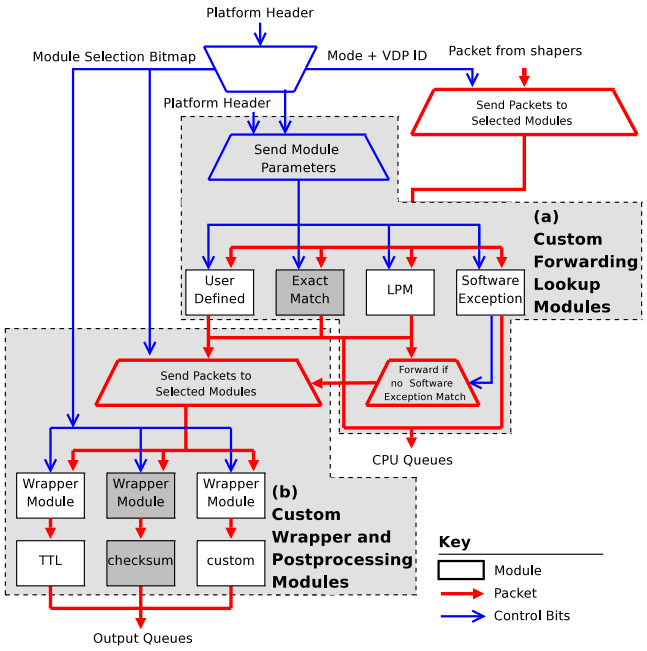


Figure 4: Output Port Lookup and Postprocessing Modules.

Once the output port lookup module determines the output port for the packet it adds the output port number to the packet's platform header. The packet is then sent to the postprocessing modules for further processing. In Section 4.5, we describe the details of software work and how the packet is handled when it is sent to the CPU.

4.5 Flexible Software Exceptions

Although performing all processing of the packets in hardware is the only way to achieve line rate performance, it may be expensive to introduce complex forwarding implementations in the hardware. Also, if certain processing will only be performed on a few packets and the processing requirements of those packets are different from the majority of other packets, development can be faster and less expensive if those few packets are processed by the CPU instead (*e.g.*, ICMP packets in routers are typically processed in the CPU).

SwitchBlade introduces *software exceptions* to programmatically direct certain packets to the CPU for additional processing. This concept is similar to the OpenFlow concept of rules that can identify packets that match a particular traffic flow that should be passed to the controller. However, combining software exceptions with the LPM table provides greater flexibility, since a VDP can add exceptions to existing forwarding rules. Similarly, if a user starts receiving more traffic than expected from a particular software exception, that user can simply remove the software exception entry and add the forwarding rule in forwarding tables.

There is a separate exceptions table, which can be filled via a register interface on a per-VDP basis and is accessible to the output port lookup module, as shown in Figure 4(a). When the mode bits field in the platform header is set to 3 (Table 2), the output port lookup module performs an exact match of the hash value in the packet's platform header with the entries in the exceptions table for the VDP. If there is a match, then the packet is redirected to the CPU where it can be processed using software-based handlers, and if there is none then the packet is sent back to the output port

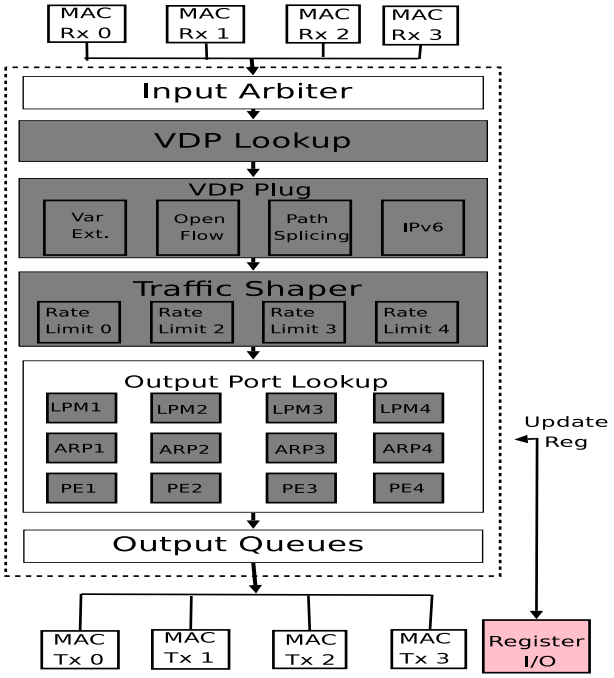


Figure 5: SwitchBlade Pipeline for NetFPGA implementation.

lookup module to perform an LPM on the destination address. We describe the process after the packet is sent to the CPU later.

SwitchBlade's software exceptions feature allows decision caching [9]: software may install its decisions as LPM or exact match rules in the forwarding tables so that future packets are forwarded rapidly in hardware without causing software exceptions.

SwitchBlade allows custom processing of some packets in software. There are two forwarding modes that permit this function: unconditional forwarding of all packets or forwarding of packets based on software exceptions to the CPU. Once a packet has been designated to be sent to the CPU, it is placed in a CPU queue corresponding to its VDP, as shown in Figure 4(a). The current SwitchBlade implementation forwards the packet to the CPU, with the platform header attached to the packet. We describe one possible implementation of a software component on top of SwitchBlade's VDP—a virtual router—in Section 7.

5. NETFPGA IMPLEMENTATION

In this section, we describe our NetFPGA-based implementation of SwitchBlade, as well as custom data planes that we have implemented using SwitchBlade. For each of these data planes, we present details of the custom modules, and how these modules are integrated into the SwitchBlade pipeline.

5.1 SwitchBlade Platform

SwitchBlade implements all the modules shown in Figure 5 on the NetFPGA [2] platform. The current implementation uses four packet preprocessor modules, as shown in Table 3. SwitchBlade uses SRAM for packet storage and BRAM and SRL16e storage for forwarding information for all the VDPs and uses the PCI interface to send or receive packets from the host machine operating system. The NetFPGA project provides reference implementations for various capabilities, such as the ability to push the Linux routing table to the hardware. Our framework extends this implementation to add other features, such as the support of virtual data planes, customizable hardware modules, and programmable software exceptions. Figure 5 shows the implementation of the NetFPGA router-based pipeline for SwitchBlade. Because our implementation is based on the NetFPGA reference implementation, adding multicast packet forwarding depends on the capabilities of NetFPGA reference router [2] implementation. Because the base implementation can support multicast forwarding, SwitchBlade can also support it.

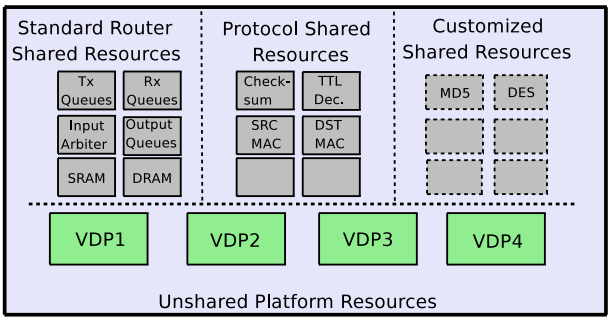


Figure 6: Resource sharing in SwitchBlade.

VDP Selection Stage. The SwitchBlade implementation adds three new stages to the NetFPGA reference router [2] pipeline as shown in gray in Figure 5. The VDP selection stage essentially performs destination MAC lookup for each incoming packet and if the destination MAC address matches then the packet is accepted and the VDP-id is attached to the packet's platform header (Table 2). VDP selection is implemented using a CAM (Content Addressable Memory), where each MAC address is associated with a VDP-ID. This table is called the *Virtual Data Plane table*. An admin register interface allows the SwitchBlade administrator to allow or disallow users from using a VDP by adding or removing their destination MAC entries from the table.

Preprocessing Stage. A developer can add customizable packet preprocessor modules to the VDP. There are two main benefits for these customizable preprocessor modules. First, this modularity streamlines the deployment of new forwarding schemes. Second, the hardware cost of supporting new protocols does not increase linearly with the addition of new protocol preprocessors. To enable custom packet forwarding, the preprocessing stage also provides a hashing module that takes 256-bits as input and produces a 32-bit output (Table 2). The hashing scheme does not provide a longest-prefix match; it only offers support for an exact match on the hash value. In our existing implementation each preprocessor module is fixed with one specific VDP.

Shaping Stage. We implement bandwidth isolation for each VDP using a simple network traffic rate limiter. Each VDP has a configurable rate limiter that increases or decreases the VDP's allocated bandwidth. We used a rate limiter from the NetFPGA's reference implementation for this purpose. The register interface to update the rate limits is accessible only with admin privileges.

Software Exceptions. To enable programmable software exceptions, SwitchBlade uses a 32-entry CAM within each VDP that can be configured from software using the register interface. SwitchBlade has a register interface that can be used to add a 32-bit hash representing a flow or packet. Each VDP has a set of registers to update the software exceptions table to redirect packets from hardware to software.

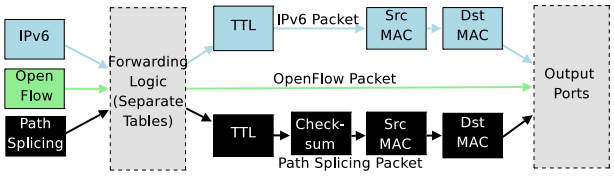


Figure 7: Life of OpenFlow, IPv6, and Path Splicing packets.

Sharing and custom packet processing. The modules that function on the virtual router instance are shared between different virtual router instances that reside on the same FPGA device. Only those modules that the virtual router user selects can operate on the packet; others do not touch the packet. This path-selection mechanism is unique. Depending on an individual virtual router user's requirements, the user can simply select the path of the packet and the modules that the virtual router user requires.

5.2 Custom Data Planes using SwitchBlade

Implementing any new functionality in SwitchBlade requires hardware programming in Verilog, but if the module is added as a pluggable preprocessor, then the developer needs to be concerned with the pluggable preprocessor implementation only, as long as decoding can occur within specific clock cycles. Once a new module is added and its interface is linked with the register interface, a user can write a high-level program to use a combination of the newly added and previously added modules. Although the number of modules in a pipeline may appear limited because of smaller header size, this number can be increased by making the pipeline wider or by adding another header for every packet.

To allow developers to write their own protocols or use existing ones, SwitchBlade offers header files in C++, Perl, and Python; these files refer to register address space for that user's register interface only. A developer simply needs to include one of these header files. Once the register file is included, the developer can write a user-space program by reading and writing to the register interface. The developer can then use the register interface to enable or disable modules in the SwitchBlade pipeline. The developer can also use this interface to add hooks for software exceptions. Figure 7 shows SwitchBlade's custom packet path. We have implemented three different routing protocols and forwarding mechanisms: OpenFlow [19], Path Splicing [17], and IPv6 [11] on SwitchBlade.

OpenFlow. We implemented the exact match lookup mechanism of OpenFlow in hardware using SwitchBlade without VLAN support. The OpenFlow preprocessor module, as shown in Figure 3, parses a packet and extracts the ten tuples of a packet defined in OpenFlow specifications. The OpenFlow preprocessor module extracts the bits from the packet header and returns a 240-bit wide OpenFlow flow entry. These 240-bits travel on a 256-bit wire to the hasher module. The hasher module returns a 32-bit hash value that is added to the SwitchBlade platform header (Figure 2). After the addition of hash value this module adds a module selector bitmap to the packet's platform header. The pipeline then sets mode field in the packet's platform header to 1, which makes the output port lookup module perform an exact match on the hash value of the packet. The output port lookup module looks up the hash value in the exact match table and forwards the packet to the output port if the lookup was a hit. If the table does not contain the corresponding entry, the platform forward the packet to the CPU for processing with software-based handlers.

Because OpenFlow offers switch functionality and does not require any extra postprocessing (*e.g.*, TTL decrement or checksum calculation), a user can prevent the forwarding stage from performing any extra postprocessing functions on the packet. Nothing happens in the forwarding stage apart from the lookup, and SwitchBlade queues the packet in the appropriate output queue. A developer can update source and destination MACs as well, using the register interface.

Path Splicing. Path Splicing enables users to select different paths to a destination based on the *splicing bits* in the packet header. The splicing bits are included as a bitmap in the packet's header and serve as an index for one of the possible paths to the destination. To implement Path Splicing in hardware, we implemented a processing module in the preprocessing stage. For each incoming packet, the preprocessor module extracts the splicing bits and the destination IP address. It concatenates the IP destination address and the splicing bits to generate a new address that represents a separate path. Since Path Splicing allows variation in path selection, this bit field can vary in length. The hasher module takes this bit field, creates a 32-bit hash value, and attaches it to the packet's platform header.

When the packet reaches the exact match lookup table, its 32-bit hash value is extracted from SwitchBlade header and is looked up in the exact match table. If a match exists, the card forwards the packet on the appropriate output port. Because the module is concatenating the bits and then hashing them and there is an exact match down the pipeline, two packets with the same destination address but different paths will have different hashes, so they will be matched against different forwarding table entries and routed along two different paths. Since Path Splicing uses IPv4 for packet processing, all the postprocessing modules on the default path (*e.g.*, TTL decrement) operate on the packet and update the packet's required fields. SwitchBlade can also support equal-cost multipath (ECMP). For this protocol, the user must implement a new preprocessor module that can select two different paths based on the packet header fields and can store their hashes in the lookup table sending packets to two separate paths based on the hash match in lookup.

IPv6. The IPv6 implementation on SwitchBlade also uses the customizable preprocessor modules to extract the 128-bit destination address from an incoming IPv6 packet. The preprocessor module extracts the 128-bits and sends them to the hasher module to generate a 32-bit hash from the address.

Our implementation restricts longest prefix match to 32-bit address fields, so it is not currently possible to perform longest prefix match for IPv6. The output port lookup stage performs an exact match on the hash value of the IPv6 packet and sends it for postprocessing. When the packet reaches the postprocessing stage, it only needs to have its TTL decremented because there is no checksum in IPv6. But it also requires to have its source and destination MACs updated before forwarding. The module selector bitmap shown in Figure 5 enables only the postprocessing module responsible for TTL decrement and not the ones doing checksum recalculation. Because the TTL offset for IPv6 is at a different byte offset than the default IPv4 TTL field, SwitchBlade uses a wrapper module that extracts only the bits of the packet's header that are required by the TTL decrement module; it then updates the packet's header with the decremented TTL.

Resource	NetFPGA Utilization	% Utilization
Slices	21 K out of 23 K	90%
4-input LUTs	37 K out of 47 K	79%
Flip Flops	20 K out of 47 K	42%
External IOBs	353 out of 692	51%
Eq. Gate Count	13 M	N/A

Table 4: Resource utilization for the base SwitchBlade platform.

6. EVALUATION

In this section, we evaluate our implementation of SwitchBlade using NetFPGA [2] as a prototype development platform. Our evaluation focuses on three main aspects of SwitchBlade: (1) resource utilization for the SwitchBlade platform; (2) forwarding performance and isolation for parallel data planes; and (3) data-plane update rates.

6.1 Resource Utilization

To provide insight about the resource usage when different data planes are implemented on SwitchBlade, we used Xilinx ISE [23] 9.2 to synthesize SwitchBlade. We found that a single physical IPv4 router implementation developed by the NetFPGA group at Stanford University uses a total of $23K$ four-input LUTs, which consume about 49% of the total available four-input LUTs, on the NetFPGA. The implementation also requires 123 BRAM units, which is 53% of the total available BRAM.

We refer to our existing implementation with one OpenFlow, one IPv6, one variable bit extractor, and one Path Splicing preprocessor with an IPv4 router and capable of supporting four VDPs as the SwitchBlade"base configuration". This implementation uses $37K$ four-input LUTs, which account for approximately 79% of four-input LUTs. Approximately 4.5% of LUTs are used for shift registers. Table 4 shows the resource utilization for the base SwitchBlade implementation; SwitchBlade uses more resources than the base IPv4 router, as shown in table 5, but the increase in resource utilization is less than linear in the number of VDPs that SwitchBlade can support.

Sharing modules enables resource savings for different protocol implementations. Table 5 shows the resource usage for implementations of an IPv4 router, an OpenFlow switch, and path splicing. These implementations achieve 4 Gbps; OpenFlow and Path Splicing implementations provide more resources than SwitchBlade. But there is not much difference in resource usage for these implementations when compared with the possible configurations which SwitchBlade can support.

Virtual Data Planes can support multiple forwarding planes in parallel. Placing four Path Splicing implementations in parallel on a larger FPGA to run four Path Splicing data planes will require four times the resources of existing Path Splicing implementation. Because no modules are shared between the four forwarding planes, the number of resources will not increase linearly and will remain constant in the best case.

From a gate count perspective, Path Splicing with larger forwarding tables and more memory will require approximately four times the resources as in Table 5; SwitchBlade with smaller forwarding tables and less memory will require almost same amount of resources. This resource usage gap begins to increase as we increase the number of Virtual Data Planes on the FPGA. Recent trends in FPGA development such as Virtex 6 suggest higher speeds and larger area; these trends will allow more VDPs to be placed on a single FPGA, which will facilitate more resource sharing.

NetFPGA Implementation	Slices	4-input LUTs	Flip Flops	BRAM	Equivalent Gate Count
Path Splicing	17 K	19 K	17 K	172	12 M
OpenFlow	21 K	35 K	22 K	169	12 M
IPv4	16 K	23 K	15 K	123	8 M

Table 5: Resource usage for different data planes.

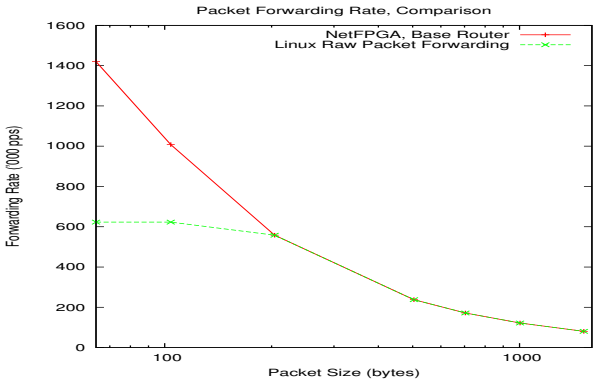


Figure 8: Comparison of forwarding rates.

6.2 Forwarding Performance and Isolation

We used the NetFPGA-based packet generator [10] for traffic generation to generate high speed traffic to evaluate the forwarding performance of SwitchBlade and the isolation provided between the VDPs. Some of the results we present in this section are derived from experiments in previous work [4].

Raw forwarding rates. Previous work has measured the maximum sustainable packet forwarding rate for different configurations of software-based virtual routers [5]. We also measure packet forwarding rates and show that hardware-accelerated forwarding can increase packet forwarding rates. We compare forwarding rates of Linux and NetFPGA-based router implementation from NetFPGA group [2], as shown in Figure 8. The maximum forwarding rate shown, about 1.4 million packets per second, is the maximum traffic rate which we were able to generate through the NetFPGA-based packet generator.

The Linux kernel drops packets at high loads, but our configuration could not send packets at a high enough rate to see packet drops in hardware. If we impose the condition that no packets should be dropped at the router, then the packet forwarding rates for the Linux router drops significantly, but the forwarding rates for the hardware-based router remain constant. Figure 8 shows packet forwarding rates when this "no packet drop" condition is *not* imposed (*i.e.*, we measure the maximum sustainable forwarding rates). For large packet sizes, SwitchBlade could achieve the same forwarding rate using in-kernel forwarding as we were using a single port of NetFPGA router. Once the packet size drops below 200 bytes; the software-based router cannot keep pace with the forwarding requirements.

Forwarding performance for Virtual Data Planes. Figure 9 shows the data-plane forwarding performance of SwitchBlade running four data planes in parallel versus the NetFPGA reference router [2], for various packet sizes. We have disabled the rate limiters in SwitchBlade for these experiments. The figure shows that running SwitchBlade incurs no additional performance penalty when compared to the performance of running the refer-

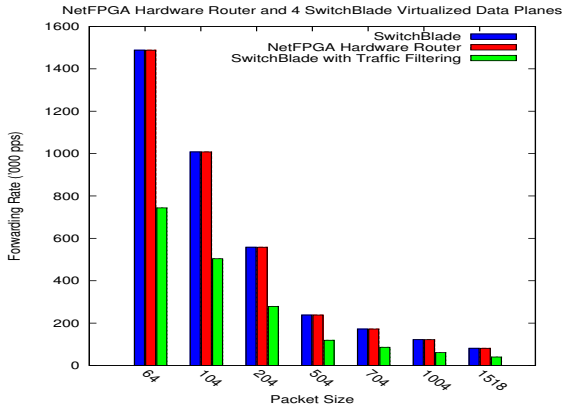


Figure 9: Data plane performance: NetFPGA reference router vs. SwitchBlade.

Physical Router ('000s of packets)			
Packet Size(bytes)	Pkts Sent	Pkts Fwd @ Core	Pkts recv @Sink
64	40 K	40 K	20 K
104	40 K	40 K	20 K
204	40 K	40 K	20 K
504	40 K	40 K	20 K
704	40 K	40 K	20 K
1004	39.8 K	39.8 K	19.9 K
1518	4 K	4 K	1.9 K

Table 6: Physical Router, Packet Drop Behavior.

ence router [2]. By default, traffic belonging to any VDP can arrive on any of the physical Ethernet interfaces since all of the ports are in promiscuous mode. To measure SwitchBlade's to filter traffic that is not destined for any VDP, we flooded SwitchBlade with a mix of traffic where half of the packets had destination MAC addresses of SwitchBlade virtual interfaces and half of the packets had destination MAC addresses that didn't belong to any vdp. As a result, half of the packets were dropped and rest were forwarded, which resulted in a forwarding rate that was half of the incoming traffic rate.

Isolation for Virtual Data Planes. To measure CPU isolation, we used four parallel data planes to forward traffic when a user-space process used 100% of the CPU. We then sent traffic where each user had an assigned traffic quota in packets per second. When no user surpassed the assigned quotas, the router forwarded traffic according to the assigned rates, with no packet loss. To measure traffic isolation, we set up a topology where two 1 Gbps ports of routers were flooded at 1 Gbps and a sink node were connected to a third 1 Gbps port. We used four VDPs to forward traffic to the same output port. Tables 6 and 7 show that, at this packet forwarding rate, only half of the packets make it through, on first come first serve basis, as shown in fourth column. These tables show that both the reference router implementation and SwitchBlade have the same performance in the worst-case scenario when an output port is flooded. The second and third columns show the number of packets sent to the router and the number of packets forwarded by the router. Our design does not prevent against contention that may arise when many users send traffic to one output port.

Forwarding performance for non-IP packets. We also tested whether SwitchBlade incurred any forwarding penalty for forwarding custom, non-IP packets; SwitchBlade was also able to forward these packets at the same rate as regular IP packets. Figure 10

Four Data Planes ('000s of packets)			
Packet Size(bytes)	Pkts Sent	Pkts Fwd @ Core	Pkts recv @Sink
64	40 K	40 K	20 K
104	40 K	40 K	20 K
204	40 K	40 K	20 K
504	40 K	40 K	20 K
704	40 K	40 K	20 K
1004	39.8 K	39.8 K	19.9 K
1518	9.6 K	9.6 K	4.8 K

Table 7: Four Parallel Data Planes, Packet Drop Behavior.

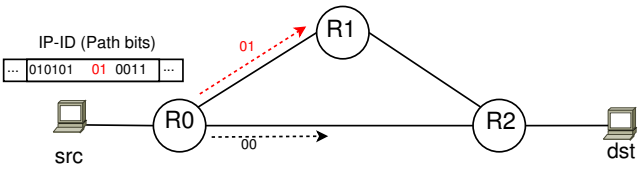


Figure 10: Test topology for testing SwitchBlade implementation of Path Splicing.

shows the testbed we used to test the Path Splicing implementation. We again used the NetFPGA-based hardware packet generator [10] to send and receive traffic. Figure 11 shows the packet forwarding rates of this NetFPGA-based implementation, as observed at the sink node. No packet loss occurred on any of the nodes shown in Figure 10. We sent two flows with same destination IP address but using different splicing bits to direct them to different routers. Packets from one flow were sent to $R2$ via $R1$, while others went directly to $R2$. In another iteration, we introduced four different flows in the network, such that all four forwarding tables at router $R0$ and $R2$ were looked up with equal probability; in this case, SwitchBlade also forwarded the packets at full rate. Both these experiments show that SwitchBlade can implement schemes like Path Splicing and forward traffic at hardware speeds for non-IP packets.

In another experiment, we used the Variable Bit Extraction module to extract first 64 bits from the header for hashing. We used a simple source and sink topology with SwitchBlade between them and measured the number of packets forwarded. Figure 12 shows the comparison of forwarding rates when forwarding was being done using SwitchBlade based on the first 64-bits of an Ethernet frame and when it was done using NetFPGA base router.

6.3 Data-Plane Update Rates

Each VDP in a router on SwitchBlade needs to have its own forwarding table. Because the VDPs share a single physical device, simultaneous table updates from different VDPs might create a bottleneck. To evaluate the performance of SwitchBlade for forwarding table update speeds, we assumed the worst-case scenario, where all VDPs flush their tables and rewrite them again at the same time. We assumed that the table size for each VDP is 400,000 entries. We updated all four tables simultaneously, but there was no performance decrease while updating the forwarding table from software. Four processes were writing the table entries in the forwarding table.

Table 8 shows updating 1.6 million entries simultaneously took 89.77 seconds on average, with a standard deviation of less than one second. As the number of VDPs increases, the average update rate remains constant, but as the number of VDPs increases, the PCI interconnect speed becomes a bottleneck between the VDP processes updating the table and the SwitchBlade FPGA.

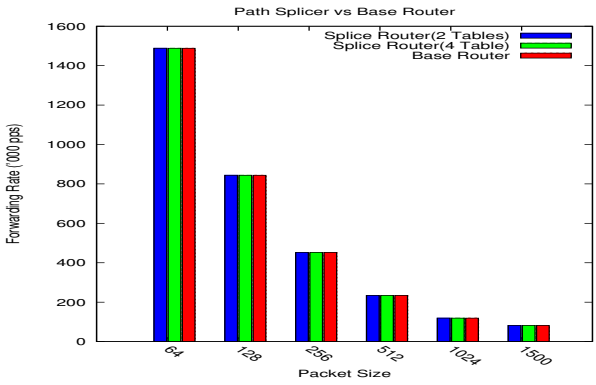


Figure 11: Path Splicing router performance with varying load compared with base router.

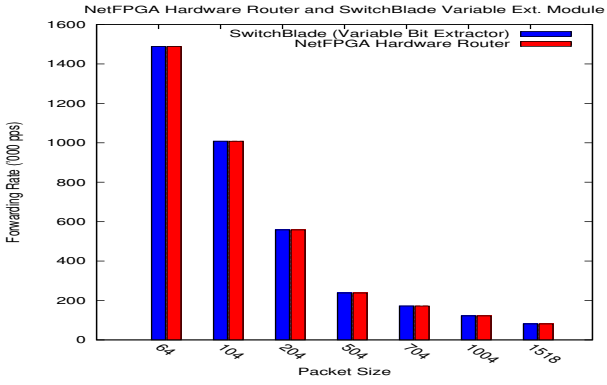


Figure 12: Variable Bit Length Extraction router performance compared with base router.

7. A VIRTUAL ROUTER ON SWITCHBLADE

We now describe our integration of SwitchBlade with a virtual router environment that runs in an OpenVZ container [20]. We use OpenVZ [20] as the virtual environment for hosting virtual routers for two reasons related to isolation. First, OpenVZ provides some level of namespace isolation between each of the respective virtual environments. Second, OpenVZ provides a CPU scheduler that prevents any of the control-plane processes from using more than its share of CPU or memory resources. We run the Quagga routing software [21] in OpenVZ, as shown in Figure 13.

Each virtual environment has a corresponding VDP that acts as its data plane. SwitchBlade exposes a register interface to send commands from the virtual environment to its respective VDP. Similarly, each VDP can pass data packets into its respective virtual environment using the software exception handling mechanisms described in Section 4.5.

We run four virtual environments on the same physical machine and use the SwitchBlade's isolation capabilities to share the hardware resources. Each virtual router receives a dedicated amount of processing and is isolated from the other routers' virtual data planes. Each virtual router also has the appearance of a dedicated data path.

8. DISCUSSION

PCIe interconnect speeds and the tension between hardware and software. Recent architectures for software-based routers such as RouteBricks, use the PCI express (PCIe) interface between the CPU, which acts as the I/O hub, and the network interface cards that forward traffic. PCIe offers more bandwidth than a standard PCI interface; for example, PCIe version 2, with 16 lanes link, has a total aggregate bandwidth of 8 GBps per direction. Although this high PCIe bandwith would seem to offer great promise for building programmable routers that rely on the CPU for packet processing, the speeds of programmable interface cards are also increasing, and it is unclear as yet whether the trends will play out in favor of CPU-based packet processing. For example, one Virtex-6 HXT FPGA from Xilinx or Stratix V FPGA from Altera can process packets at 100 Gbps. Thus, installing NICs with only one such FPGA can make the PCIe interconnect bandwidth a bottleneck, and also puts an inordinate amount of strain on the CPU. SwitchBlade thus favors making FPGAs more flexible and programmable, allowing more customizability to take place directly on the hardware itself.

VDPs	Total Ent.	Entries/Table	Time(sec)	Single Ent. (μs)
1	400 K	400 K	86.582	216
2	800 K	400 K	86.932	112
3	1,200 K	400 K	88.523	74
4	1,600 K	400 K	89.770	56

Table 8: Forwarding Table Update Performance.

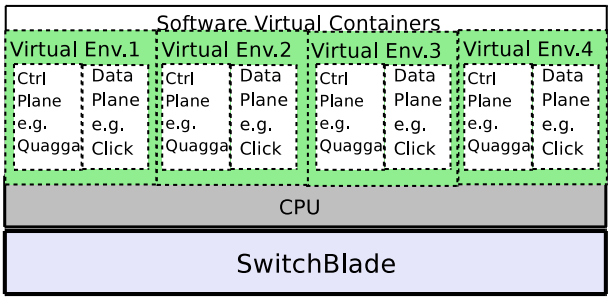


Figure 13: Virtual router design with OpenVZ virtual environments interfacing to SwitchBlade data plane.

Modifying packets in hardware. SwitchBlade's hardware implementation focuses on providing customization for protocols that make only limited modifications to packets. The design can accommodate writing packets using preprocessor modules, but we have not yet implemented this function. Providing arbitrary writing capability in hardware will require either using preprocessor stage for packet writing and a new pipeline stage after postprocessing, or adding two new stages to the pipeline (both before and after lookup).

Packet rewriting can be performed in two ways: (1) modifying existing packet content without changing total data unit size, or (2) adding or removing some data to each packet with the output packet size different from the input packet size. Although it is easy to add the first function to the preprocessor stage, adding or removing bytes into packet content will require significant effort.

Scaling SwitchBlade. The current SwitchBlade implementation provides the capability for four virtualized data planes on a single NetFPGA, but this design is general enough to scale as the capabilities of hardware improve. We see two possible avenues for increasing the number of virtualized data planes in hardware. One option is to add several servers, each having one FPGA card and have one or more servers running the control plane that controls the hardware forwarding-table entries. Other scaling options include adding more FPGA cards to a single physical machine or

taking advantage of hardware trends, which promise the ability to process data in hardware at increasingly higher rates.

9. CONCLUSION

We have presented the design, implementation, and evaluation of SwitchBlade, a platform for deploying custom protocols on programmable hardware. SwitchBlade uses a pipeline-based hardware design; using this pipeline, developers can swap common hardware processing modules in and out of the packet-processing flow on the fly, without having to resynthesize hardware. SwitchBlade also offers programmable software exception handling to allow developers to integrate custom functions into the packet processing pipeline that cannot be handled in hardware. SwitchBlade's customizable forwarding engine also permits the platform to make packet forwarding decisions on various fields in the packet header, enabling custom, non-IP based forwarding at hardware speeds. Finally, SwitchBlade can host multiple data planes in hardware in parallel, sharing common hardware processing modules while providing performance isolation between the respective data planes. These features make SwitchBlade a suitable platform for hosting virtual routers or for simply deploying multiple data planes for protocols or services that offer complementary functions in a production environment. We implemented SwitchBlade using the NetFPGA platform, but SwitchBlade can be implemented with any FPGA.

Acknowledgments

This work was funded by NSF CAREER Award CNS-0643974 and NSF Award CNS-0626950. We thank Mohammad Omer for his help in solving various technical difficulties during project. We also thank our shepherd, Amin Vahdat, for feedback and comments that helped improve the final draft of this paper.

REFERENCES

[1] FlowVisor. `http://www.openflowswitch.org/wk/index.php/FlowVisor`.

[2] NetFPGA. `http://www.netfpga.org`.

[3] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *Proc. ACM SIGCOMM*, Seattle, WA, Aug. 2008.

[4] M. B. Anwer and N. Feamster. Building a Fast, Virtualized Data Plane with Programmable Hardware. In *Proc. ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, Barcelona, Spain, Aug. 2009.

[5] S. Bhatia, M. Motiwala, W. Muhlbauer, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. Hosting Virtual Networks on Commodity Hardware. Technical Report GT-CS-07-10, Georgia Institute of Technology, Atlanta, GA, Oct. 2007.

[6] S. Bhatia, M. Motiwala, W. Mühlbauer, V. Valancius, A. Bavier, N. Feamster, J. Rexford, and L. Peterson. Hosting virtual networks on commodity hardware. Technical Report GT-CS-07-10, College of Computing, Georgia Tech, Oct. 2007.

[7] G. Calarco, C. Raffaelli, G. Schembra, and G. Tusa. Comparative analysis of smp click scheduling techniques. In *QoS-IP*, pages 379–389, 2005.

[8] L. D. Carli, Y. Pan, A. Kumar, C. Estan, and K. Sankaralingam. Flexible lookup modules for rapid deployment of new protocols in high-speed routers. In *Proc. ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[9] M. Casado, T. Koponen, D. Moon, and S. Shenker. Rethinking packet forwarding hardware. In *Proc. Seventh ACM SIGCOMM HotNets Workshop*, Nov. 2008.

[10] G. A. Covington, G. Gibb, J. Lockwood, and N. McKeown. A Packet Generator on the NetFPGA platform. In *FCCM '09: IEEE Symposium on Field-Programmable Custom Computing Machines*, 2009.

[11] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. Internet Engineering Task Force, Dec. 1998. RFC 2460.

[12] M. Dobrescu, , N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting parallelism to scale software routers. In *Proc. 22nd ACM Symposium on Operating Systems Principles (SOSP)*, Big Sky, MT, Oct. 2009.

[13] B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. In *Proc. ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[14] Intel IXP 2xxx Network Processors. `http://www.intel.com/design/network/products/npfamily/ixp2xxx.htm`.

[15] C. Kim, M. Caesar, and J. Rexford. Floodless in SEATTLE: A scalable ethernet architecture for large enterprises. In *Proc. ACM SIGCOMM*, Seattle, WA, Aug. 2008.

[16] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, Aug. 2000.

[17] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path Splicing. In *Proc. ACM SIGCOMM*, Seattle, WA, Aug. 2008.

[18] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: A scalable fault-tolerant layer2 data center network fabric. In *Proc. ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[19] OpenFlow Switch Consortium. `http://www.openflowswitch.org/`, 2008.

[20] OpenVZ: Server Virtualization Open Source Project. `http://www.openvz.org`.

[21] Quagga software routing suite. `http://www.quagga.net/`.

[22] J. Turner, P. Crowley, J. DeHart, A. Freestone, B. Heller, F. Kuhns, S. Kumar, J. Lockwood, J. Lu, M. Wilson, et al. Supercharging PlanetLab: A High Performance, Multi-application, Overlay Network Platform. In *Proc. ACM SIGCOMM*, Kyoto, Japan, Aug. 2007.

[23] Xilinx. Xilinx ise design suite. `http://www.xilinx.com/tools/designtools.htm`.

[24] X. Yang, D. Wetherall, and T. Anderson. Source selectable path diversity via routing deflections. In *Proc. ACM SIGCOMM*, Pisa, Italy, Aug. 2006.