

Chadi Barakat
Ian Pratt (Eds.)

Passive and Active Network Measurement

5th International Workshop, PAM 2004
Antibes Juan-les-Pins, France, April 2004
Proceedings



Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board:

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Oscar Nierstrasz

University of Berne, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

Dortmund University, Germany

Demetri Terzopoulos

New York University, NY, USA

Doug Tygar

University of California at Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Springer

Berlin

Heidelberg

New York

Hong Kong

London

Milan

Paris

Tokyo

Chadi Barakat Ian Pratt (Eds.)

Passive and Active Network Measurement

5th International Workshop, PAM 2004
Antibes Juan-les-Pins, France, April 19-20, 2004
Proceedings

Springer

eBook ISBN: 3-540-24668-1
Print ISBN: 3-540-21492-5

©2005 Springer Science + Business Media, Inc.

Print ©2004 Springer-Verlag
Berlin Heidelberg

All rights reserved

No part of this eBook may be reproduced or transmitted in any form or by any means, electronic, mechanical, recording, or otherwise, without written consent from the Publisher

Created in the United States of America

Visit Springer's eBookstore at: <http://ebooks.springerlink.com>
and the Springer Global Website Online at: <http://www.springeronline.com>

Preface

PAM 2004 was the 5th International Workshop on Passive and Active Measurement, held in Juan-les-Pins on the French Riviera, co-organized by the University of Cambridge and INRIA-Sophia Antipolis, with financial support from Intel and Cisco Systems.

This year we received a record number of submissions (184), reflecting the growth of the field and the critical role it plays in maintaining the network infrastructure on which we all rely. From the two-page abstracts submitted, the programme committee selected 29 papers whose authors were invited to submit full papers to appear in these proceedings. Particular emphasis was placed on selecting work that we felt was fresh and exciting, so as to encourage a dynamic and interactive workshop that provided a first public presentation of research that will go on to appear in other, more formal conferences and journals. The programme committee was greatly impressed with the strength and depth of submissions received, which bodes well for the future of the subject area.

This workshop took place during April 19–20, 2004 in Juan-les-Pins. Located between the Alps and the Mediterranean, and close to Nice, Cannes and Monaco, Juan-les-Pins is one of the most beautiful sites on the French Riviera. Juan-les-Pins is also close to Sophia Antipolis, the French telecom valley.

The workshop could not have succeeded without the support of many people whom we would like thank. First, we thank the members of the programme committee for donating a considerable amount of time to review the unexpectedly large number of submissions, while working to a very tight deadline.

Second, we would like to thank the University of Cambridge and INRIA Sophia Antipolis for their support. In particular, special thanks go to Tim Granger for running the conference Web site, and writing the tools used to process the paper submissions and collate reviews. Dany Sergeant from INRIA kindly took care of the registration and financial issues. Gianluca Iannaccone chaired the student travel support panel, and Andrew Moore collated the published proceedings. Joerg Micheel of Endace ran the e-mail MTA for the pam2004.org domain.

We are very grateful to Cisco Systems and Intel for their generous financial support that enabled us to assist 15 graduate students in attending the workshop.

Finally, our thanks go to the authors of the submitted papers, to the speakers, and to all the participants for making PAM 2004 a success!

We hope you enjoyed the PAM 2004 workshop, and had a pleasant stay on the French Riviera.

April 2004

Chadi Barakat (General Chair)
Ian Pratt (PC Chair)

This page intentionally left blank

Organization

PAM 2004 was organized by the University of Cambridge Computer Laboratory and INRIA Sophia Antipolis.

Program Committee

Chadi Barakat (INRIA, France)
Andre Broido (CAIDA, USA)
Nevil Brownlee (University of Auckland, New Zealand)
Randy Bush (IIJ, Japan)
Avi Freedman (Akamai, USA)
Ian Graham (University of Waikato, New Zealand)
Geoff Huston (Telstra, Australia)
Gianluca Iannaccone (Intel Research, UK)
Peter Key (Microsoft Research, UK)
Yasuichi Kitamura (APAN, Japan)
Joerg Micheel (NLANR/Endace, USA)
Sue Moon (KAIST, Korea)
Andrew Moore (University of Cambridge, UK)
Ian Pratt (University of Cambridge, UK)
Ronn Ritke (NLANR/MNA, USA)
Nicolas Simar (DANTE, UK)
Alex Tudor (Agilent, USA)
Daryl Veitch (University of Melbourne, Australia)

Additional Reviewers

M. Crovella	T. Griffin	J. Li
J. Crowcroft	A. Al Hamra	V. Ramos
C. Filsfils	V. Kanevsky	L. Rizzo

Sponsoring Institutions

Intel Research, Cambridge, UK
Cisco Systems, San Jose, California, USA

This page intentionally left blank

Table of Contents

Session 1: P2P and Overlay

Dissecting BitTorrent: Five Months in a Torrent's Lifetime	1
<i>M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, L. Garcés-Erice</i>	
A Measurement-Based Traffic Profile of the eDonkey Filesharing Service .	12
<i>Kurt Tutschku</i>	
Bootstrapping in Gnutella: A Measurement Study.....	22
<i>Pradnya Karbhari, Mostafa Ammar, Amogh Dhamdhere, Himanshu Raj, George F. Riley, Ellen Zegura</i>	
Testing the Scalability of Overlay Routing Infrastructures	33
<i>Sushant Rewaskar, Jasleen Kaur</i>	

Session 2: Network Optimization

Toward a Measurement-Based Geographic Location Service	43
<i>Artur Ziviani, Serge Fdida, José F. de Rezende, Otto Carlos M.B. Duarte</i>	
An Incremental Super-linear Preferential Internet Topology Model	53
<i>Sagy Bar, Mira Gonen, Avishai Wool</i>	
Geometric Exploration of the Landmark Selection Problem.....	63
<i>Liying Tang, Mark Crovella</i>	
The Interdomain Connectivity of PlanetLab Nodes	73
<i>Suman Banerjee, Timothy G. Griffin, Marcelo Pias</i>	

Session 3: Traffic Analysis

Searching for Self-similarity in GPRS	83
<i>Roger Kalden, Sami Ibrahim</i>	
The Effect of Flow Capacities on the Burstiness of Aggregated Traffic ...	93
<i>Hao Jiang, Constantinos Dovrolis</i>	
Micro-time-scale Network Measurements and Harmonic Effects	103
<i>Mark Carson, Darrin Santay</i>	
Their Share: Diversity and Disparity in IP Traffic	113
<i>Andre Broido, Young Hyun, Ruomei Gao, kc claffy</i>	

Session 4: Protocol and System Measurement

Origins of Microcongestion in an Access Router	126
<i>Konstantina Papagiannaki, Darryl Veitch, Nicolas Hohn</i>	
Performance Measurement and Analysis of H.323 Traffic	137
<i>Prasad Calyam, Mukundan Sridharan, Weiping Mandrawa, Paul Schopis</i>	
Measurements and Laboratory Simulations of the Upper DNS Hierarchy	147
<i>Duane Wessels, Marina Fomenkov, Nevil Brownlee, kc claffy</i>	

Session 5: Tools

A Robust Classifier for Passive TCP/IP Fingerprinting	158
<i>Robert Beverly</i>	
NETI@home: A Distributed Approach to Collecting End-to-End Network Performance Measurements	168
<i>Charles Robert Simpson, Jr., George F. Riley</i>	
Comparative Analysis of Active Bandwidth Estimation Tools	175
<i>Federico Montesino-Pouzols</i>	
Active Measurement for Multiple Link Failures Diagnosis in IP Networks	185
<i>Hung X. Nguyen, Patrick Thiran</i>	

Session 6: Miscellaneous

Structured Errors in Optical Gigabit Ethernet Packets	195
<i>Laura James, Andrew Moore, Madeleine Glick</i>	
Flow Clustering Using Machine Learning Techniques	205
<i>Anthony McGregor, Mark Hall, Perry Lorier, James Brunskill</i>	
Using Data Stream Management Systems for Traffic Analysis – A Case Study –	215
<i>Thomas Plagemann, Vera Goebel, Andrea Bergamini, Giacomo Tolu, Guillaume Urvoy-Keller, Ernst W. Biersack</i>	
Inferring Queue Sizes in Access Networks by Active Measurement	227
<i>Mark Claypool, Robert Kinicki, Mingzhe Li, James Nichols, Huahui Wu</i>	

Session 7: Network Measurement

Reordering of IP Packets in Internet	237
<i>Xiaoming Zhou, Piet Van Mieghem</i>	

Effects of Interrupt Coalescence on Network Measurements	247
<i>Ravi Prasad, Manish Jain, Constantinos Dovrolis</i>	
Measurement Approaches to Evaluate Performance Optimizations for Wide-Area Wireless Networks	257
<i>Rajiv Chakravorty, Julian Chesterfield, Pablo Rodriguez, Suman Banerjee</i>	
Session 8: BGP and Routing	
Measuring BGP Pass-Through Times	267
<i>Anja Feldmann, Hongwei Kong, Olaf Maennel, Alexander Tudor</i>	
Impact of BGP Dynamics on Router CPU Utilization	278
<i>Sharad Agarwal, Chen-Nee Chuah, Supratik Bhattacharyya, Christophe Diot</i>	
Correlating Internet Performance Changes and Route Changes to Assist in Trouble-Shooting from an End-User Perspective	289
<i>Connie Logg, Jiri Navratil, Les Cottrell</i>	
Author Index	299

This page intentionally left blank

Dissecting BitTorrent: Five Months in a Torrent’s Lifetime

M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and
L. Garcés-Erice

Institut Eurecom, 2229, route des Crêtes, 06904 Sophia-Antipolis, France
`{izal,urvoy,erbi,felber,alhamra,garces}@eurecom.fr`

Abstract. Popular content such as software updates is requested by a large number of users. Traditionally, to satisfy a large number of requests, larger server farms or mirroring are used, both of which are expensive. An inexpensive alternative are peer-to-peer based replication systems, where users who retrieve the file, act simultaneously as clients and servers. In this paper, we study BitTorrent, a new and already very popular peer-to-peer application that allows distribution of very large contents to a large set of hosts. Our analysis of BitTorrent is based on measurements collected on a five months long period that involved thousands of peers. We assess the performance of the algorithms used in BitTorrent through several metrics. Our conclusions indicate that BitTorrent is a realistic and inexpensive alternative to the classical server-based content distribution.

1 Introduction

BitTorrent [4] is a file distribution system based on the peer-to-peer (P2P) paradigm. BitTorrent has quickly emerged as a viable and popular alternative to file mirroring for the distribution of large content, as testified by the numerous Web sites that host active “torrents” (e.g., <http://f.scarywater.net/>).

We have conducted a comprehensive analysis of BitTorrent to assess its performance. To that end, we have used two sources of information. First, we have obtained the “tracker” log of arguably the most popular torrent (BitTorrent session) so far—the 1.77GB Linux Redhat 9 distribution—for its first 5 months of activity. The log contains statistics for more than 180,000 clients, and most interestingly, it clearly exhibits an initial flash-crowd period with more than 50,000 clients initiating a download in the first five days. This first source of information allows us to estimate the global efficiency of BitTorrent, the macroscopic behavior of clients, and the scalability of a P2P application under flash-crowd conditions. Our second source of information consists of data collected with a modified client that participated to the same torrent downloading Redhat 9. This second log allows us to study the direct interactions between the clients. The remaining of the paper is organized as follows. In Section 2, we present the main features of BitTorrent. In Section 3, we review the related work. In Section 4, we present the results obtained from the tacker log and in Section 5 the conclusions obtained from our client log. We conclude with future directions in Section 6.

2 BitTorrent

BitTorrent is a P2P application that capitalizes the *resources* (access bandwidth and disk storage) of peer nodes to efficiently distribute large contents. There is a separate torrent for each file that is distributed. Unlike other well-known P2P applications, such as Gnutella or Kazaa, which strive to quickly *locate* hosts that hold a given file, the sole objective of BitTorrent is to quickly *replicate* a single large file to a set of clients. The challenge is thus to maximize the speed of replication.

A torrent consists of a central component, called *tracker* and all the currently active peers. BitTorrent distinguishes between two kinds of peers depending on their download status: clients that have already a complete copy of the file and continue to serve other peers are called *seeds*; clients that are still downloading the file are called *leechers*. The tracker is the only centralized component of the system. The tracker is not involved in the actual distribution of the file; instead, it keeps meta-information about the peers that are currently active and acts as a rendez-vous point for all the clients of the torrent.

A user joins an existing torrent by downloading a *torrent* file (usually from a Web server), which contains the IP address of the tracker. To initiate a new torrent, one thus needs at least a Web server that allows to discover the tracker and an initial seed with a complete copy of the file. To update the tracker's global view of the system, active clients periodically (every 30 minutes) report their state to the tracker or when joining or leaving the torrent. Upon joining the torrent, a new client receives from the tracker a list of active peers to connect to. Typically, the tracker provides 50 peers chosen at random among active peers while the client seeks to maintain connections to 20–40 peers. If ever a client fails to maintain at least 20 connections, it recontacts the tracker to obtain additional peers. The set of peers to which a client is connected is called its *peer set*.

The clients involved in a torrent cooperate to replicate the file among each other using *swarming* techniques: the file is broken into equal size chunks (typically 256kB each) and the clients in a peer set exchange chunks with one another. The swarming technique allows the implementation of parallel download [7] where different chunks are simultaneously downloaded from different clients. Each time a client obtains a new chunk, it informs all the peers it is connected with. Interactions between clients are primarily guided by two principles. First, a peer preferentially sends data to peers that reciprocally sent data to him. This “tit-for-tat” strategy is used to encourage cooperation and ban “free-riding” [1]. Second, a peer limits the number of peers being served simultaneously to 4 peers and continuously looks for the 4 best downloaders (in terms of the rate achieved) if it is a seed or the 4 best uploaders if it is a leecher.

BitTorrent implements these two principles, using a “choke/unchoke” policy. “Choking” is a temporary refusal to upload to a peer. However, the connection is not closed and the other party might still upload data. A leecher services the 4 best uploaders and chokes the other peers. Every 10 seconds, a peer re-evaluates the upload rates for all the peers that transfer data to him. There might be more than 4 peers uploading to him since first, choking is not necessarily reciprocal

and second, peers are not synchronized.¹ He then chokes the peer, among the current top 4, with the smallest upload rate if another peer offered a better upload rate. Also, every 3 rounds, that is every 30 seconds, a peer performs an *optimistic unchoke*, and unchoke a peer regardless of the upload rate offered. This allows to discover peers that might offer a better service (upload rate). Seeds essentially apply the same strategy, but based solely on download rates. Thus, seeds always serve the peers to which the download rate is highest.

Another important feature of BitTorrent is the chunk selection algorithm. The main objective is to consistently maximize the entropy of each chunk in the torrent. The heuristic used to achieve this goal is that a peer always seeks to upload the chunk the least duplicated among the chunks it needs in its peer set (keep in mind that peers only have a local view of the torrent). This policy is called the *rarest first policy*. There exists an exception to the rarest first policy when a peer joins a torrent and has no chunks. Since this peer needs to quickly obtain a first chunk (through optimistic unchoke), it should not ask for the rarest chunk because few peers hold this chunk. Instead, a newcomer uses a random first policy for the first chunk and then turns to the rarest first policy for the next ones.

3 Previous Work

Approaches to replicate contents to a large set of clients can be classified as client side and server side approaches. The first client-side approach was to cache contents already downloaded by clients of a given network. A symmetric approach on the server side is to transparently redirect clients to a set of mirror sites run by a content provider (e.g. Akamai).

The peer-to-peer paradigm has been applied to obtain client side and server side solutions. On the server side, one finds proposals like [5] where overlay nodes dynamically organize themselves so as to form a tree with maximum throughput. FastReplica [2] is designed to efficiently replicate large contents to a set of well-known and stable overlay nodes.

On the client side, one finds many proposals to build application-layer multic平 services [8,6,3]. A solution similar to BitTorrent is Slurpie [9]. Slurpie aims at enforcing cooperation among clients to alleviate the load of a Web server that is the primary source of the content. The algorithms of Slurpie are much more complex than the ones of BitTorrent and require to estimate the number of peers in the Slurpie network. The expected improvements over BitTorrent is less load on the topology server (equivalent to the BitTorrent tracker) and on the primary source (original seed) through a back-off algorithm. Results are promising since Slurpie is able to outperform BitTorrent in a controlled environment. Still, the actual performance of Slurpie in case of flash crowds and for a large number of clients is unknown.

¹ For instance, a peer that was offering a very good upload rate has decided to choke this connection just 1 second before the reevaluation on the other side and thus he might still be in the top 4 list for the next 10 seconds and received service.

4 Tracker Log Analysis

The tracker log covers a period of 5 months from April to August 2003. The corresponding torrent has as content the 1.77 GB Linux Redhat 9 distribution. 180,000 clients participated to this torrent with a peak of 51,000 clients during the first five days (see Figures 1(a) and 1(b)). These first five days clearly exhibits a flash-crowd. As clients periodically report to the tracker their current state, along with the amount of bytes they have uploaded and downloaded, the tracker log allows us to observe the global evolution of the file replication process among peers.

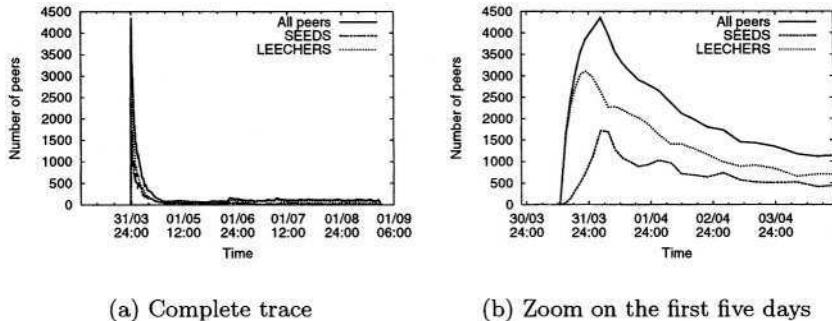


Fig. 1. Number of active peers over time

4.1 Global Performance

Analyzing the tracker log, our first finding is that BitTorrent clients are altruistic in the sense that they actively send data to other clients, both as leechers and as seeds. Altruism is enforced during the download phase by the tit-for-tat policy, as a selfish client will be served with a very low priority. Once they become seed, the peers remain connected for another six and a half hours on average. This “social” behavior can be explained by two factors: first, the client must be explicitly terminated after completion of the download, which might well happen while the user is not at his computer, e.g., overnight; second, as the content being replicated is perfectly legal, the user has no particular incentive to quickly disconnect from the torrent. In fact, the presence of seeds is a key feature, since it greatly enhances the upload capacity of this torrent and the ability to scale to large client populations. Over the 5 months period covered by the log file, we observed that the seeds have contributed more than twice the amount of data sent by leechers (see Figure 2). We also observed that the proportion of seeds is consistently higher than 20%, with a peak at 40% during the first 5 days

(see Figure 3). This last figure clearly illustrates that BitTorrent can sustain a high flash-crowd since it quickly creates new seeds. To put it differently, in situations where new peers arrive at a high rate, the resources of the system are not divided evenly between clients, which would result, like in a processor sharing queue under overload, in no peers completing the download. On the contrary, older peers have a higher priority since they hold more chunks than younger peers, which gives them more chance to complete the download and become seeds for newcomers. Obviously, this strategy benefits from the cooperation of users that let their clients stay as seeds for long periods of time.

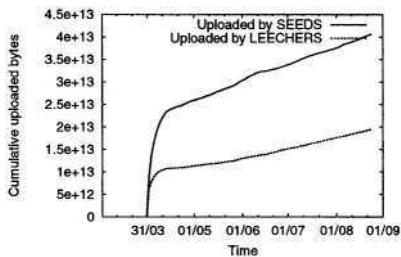


Fig. 2. Volumes uploaded by seeds and leechers

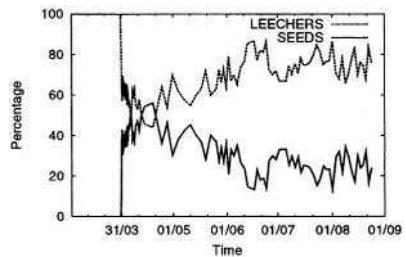


Fig. 3. Proportions of seeds and leechers

4.2 Individual Session Performance

A fundamental performance metric for BitTorrent is the average download rate of leechers. Over the 5 months period covered by the tracker log, we observed an average download rate consistently above 500kb/s. This impressive figure indicates that most of the BitTorrent clients have good connectivity (at least ADSL), which is not surprising given the total size of the file. Moreover, BitTorrent exhibits good scalability: during the initial flash-crowd, the average download rate was close to 600kb/s and the aggregate download rate of all simultaneously active clients was more than 800Mb/s.

BitTorrent is thus able to capitalize the bandwidth of end system hosts to achieve a very high aggregate throughput. Still, the final objective of BitTorrent is to replicate a given content over all the peers in a torrent. We thus need to know how many peers eventually completed the download and the fraction of bytes that the peers that did not complete the transfer downloaded. Since BitTorrent allows users to suspend and resume the download at any time, a peer might complete the download after one or several sessions. In the tracker log, a session id identifies the session (hash of the IP address of the host and its current time), along with the IP address of the host as seen by the tracker. Thus, identifying single session downloads is easy based on the session id. To reconstruct multi-sessions downloads, we assumed that the IP address of the host does not change

from one session to another. NATs often used port numbers to disambiguate connexions and not IP addresses, so our assumption can hold even in this case. Of course, if two or more hosts behind a firewall simultaneously participate to the same torrent, it might be difficult to disambiguate them. In addition, since a peer provides the amount of bytes it has downloaded in each of its reports (sent every 30 minutes), we mated two sessions with the same IP if the amount of bytes of the last report for the first session is close to the amount of bytes of the first report of the second session. These two values are not necessarily the same if for instance, the user disconnected the BitTorrent client improperly which results in the latter not sending its disconnection report with its current amount of bytes downloaded. A set of sessions form a multi-sessions download if the download of the file is completed during the last session (on average, we found that a multi-session download consists of 3.9 sessions). We also observed some sessions that started with already 100% of the file downloaded. This is the case if the user is kind enough to rejoin the torrent after the completion of the download to act as a seed. We thus categorize the sessions into four categories : single session download, multi-session download, seed sessions and session that never complete (incomplete downloads). We provide statistics for these sessions in Table 1.

Table 1. Sessions types and their characteristics

Type	Number of sessions	Total down. (TB)	Total up. (TB)	Down. /session (MB)	Up. /session (MB)	Down. rate (kb/s)	Up. rate (kb/s)	Duration (hours)
Single session downloads	20584	34.7	28.5	1765.7	1450.2	1331.1	325. 2	8.1
Multi-session downloads	14285	6.2	4.3	455.2	319.2	390.6	145.0	19.2
Incomplete downloads	138423	11.2	9.1	84.5	69.1	114.7	50.5	-
Seed sessions	8555	-	12.7	-	1556.6	-	223.6	-
Total	181847	52.1	54.6	-	-	-	-	-

Overall, Table 1 indicates that only 19% of the sessions are part of a transfer that eventually completed. However, the 81% of sessions that did not complete the transfer represent only 21.5% of the total amount of downloaded bytes. Moreover, the incomplete sessions uploaded almost as much bytes as they downloaded. It is difficult to assess the reason behind the abortion of a transfer. It might be because the user eventually decides he is not interested in the file or because of a poor download performance. To investigate this issue, we look at the statistics (durations and volumes) for sessions that start with 0% of the file during the first five days. We classify these sessions into two categories denoted as “completed” and “non-completed”. The completed category corresponds to the single session downloads defined previously while the non-completed category corresponds to transfers that never complete or the first session of transfers that will eventually complete. We present on Figure 4 the complementary cumulative distribution functions for the download volumes and session durations

for the completed and non-completed sessions. We can observe from this figure that 90% of the non-completed sessions remain less than 10,000 seconds in the torrent (and 60% less than 1000 seconds) and retrieve less than 10% of the file. Most of the non-completed sessions are thus short and little data is downloaded.

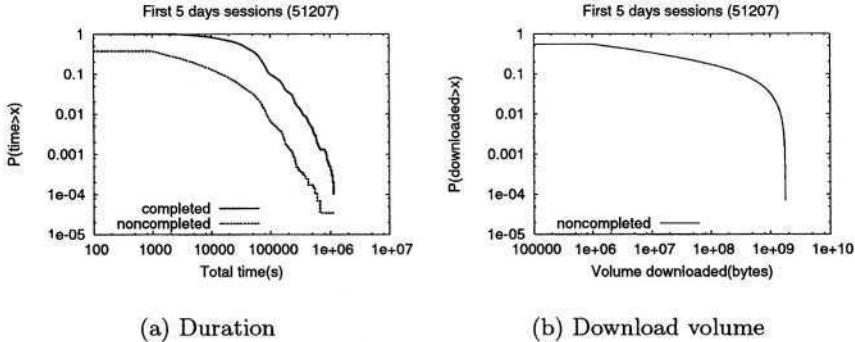


Fig. 4. Single sessions statistics for sessions seen during the first 5 days

Let us now further focus on the performance achieved by single session downloads. The average download rate of these 20,584 sessions is close to 1.3Mb/s over the whole trace, which is larger than the download rate averaged over all sessions (500kb/s). The average download time for single session download is about 29,000 seconds overall. Such values reveal a high variability in the download rates achieved by these sessions since if they had all achieved a download rate of 1.3Mb/s, the average download time would be $\frac{1.77\text{GB}}{1.3\text{Mb/s}} \sim 10,000$ seconds. This is confirmed by the distribution of these download throughputs (see Figure 5(a)) that clearly exhibits a peak around 400kb/s (a typical value for ADSL lines), a value significantly smaller than the mean.

4.3 Geographical Analysis

The tracker log provides the IP addresses of the clients. Based on these addresses, we have used NetGeo (<http://www.caida.org/tools/utilities/netgeo/>) to obtain an estimation of the origin country of the peers that participated to the torrent. The estimation might be imprecise as NetGeo is not maintained any more. Overall, we were not able to obtain information for around 10% of the hosts. In Table 2 we indicate the top five countries for the first 5 days, the first 4 weeks and the complete trace. We can observe that this set of countries, as well as the ranking is consistently the same for all three time scales. Most of the clients are US clients while Europe is represented only through the Netherlands. Still, for Netherlands, 50% of the hosts originate from ripe.net (an information

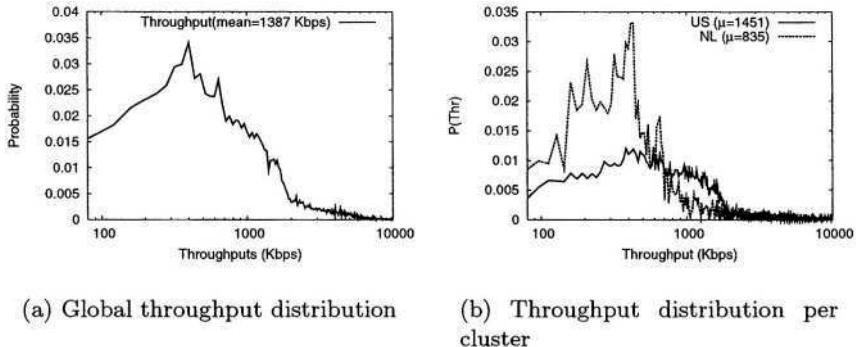


Fig. 5. Throughputs for single session downloads

also provided by NetGeo), which acts as the Regional Internet Registry for the whole of Europe, Middle East, Africa and part of Asia also. Thus we can guess that most of these peers are spread all over Europe.

Table 2. Geographical origins of peers

Countries	First week	First four weeks	Complete trace
United States	44.6%	44.3%	32%
Netherlands (Europe)	14.9%	15.3%	23.9%
Australia	12.7%	12.6%	17.8%
Canada	5.7%	5.8%	4.9%
% of total hosts	77.9%	78%	78.6%

We next investigate the relative performance of the four clusters identified above (US, NL, AU and CN). To do so, we consider again the 20,584 peers that complete the download in a single session. 17,000 peers out of the 20,584 initial peers, are in the four clusters (11498 in US cluster, 2114 in NL, 1995 in AU and 1491 in CM). We plot in Figure 5(b) the distribution of download throughputs achieved by the peers in the NL and US clusters (the AU and CN clusters are highly similar to the US cluster and not depicted for sake of clarity). Figure 5(b) reveals that the download throughput of the hosts in the NL cluster is significantly smaller than the throughput of the hosts in the US cluster (where more mass is on the right side of the curve). This can indicate that clients in the US have, in general, better access links than in Europe.

5 Client Log Analysis

To better observe the individual behavior of a BitTorrent peer, we have run an instrumented client behind a 10Mb/s campus network access link. Our client joined the torrent approximatively in the middle of the 5 months period (i.e., far after the initial flash crowd). We experienced a transfer time of approximately 4,500 seconds (i.e., much lower than the average download times previously mentioned) and our client remained connected as a seed for 13 hours. Our client logged detailed information about the upload or download of each individual chunk. In Figure 6, we represent the number of peers with whom our client is trading. At time 0, our client knows around 40 peers whose addresses have been provided by the tracker. We next continuously discover new peers (peers that contacted us) up to the end of the download (at time 4,500 seconds) where we observe a sudden decrease of the number of peers, most probably because the seeds we were connected to have closed their connection to us as soon as we have completed the download. After the download, we stay connected as seed to between 80 and 95 leechers (4 of them being served while the others are choked).

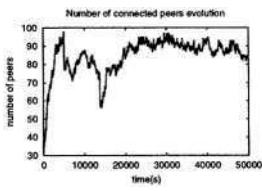


Fig. 6. Number of peers during and after download

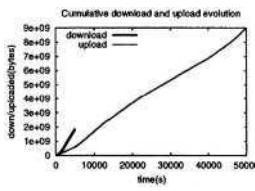


Fig. 7. Complete torrent cumulative download and upload evolution

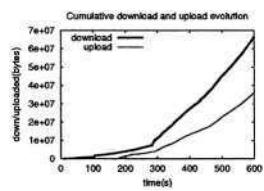


Fig. 8. First 10 minutes of download

Figures 7 and 8 show the amount of bytes downloaded and uploaded with respect to time for the complete trace and the first 10 minutes of the trace respectively. From these figures, we can draw several conclusions:

- There is a warm-up period (around 100 seconds) to obtain some first chunks. But as soon as the client has obtained a few chunks, it is able to start uploading to others peers. This means that the rarest first policy works well, otherwise our client would maybe not find anyone interested in its first chunks.
- The download and upload rates are (positively) correlated, which indicates that the tit-for-tat policy works. It also indicates that we always find peers interested in our chunks and peers from which we can download chunks. Otherwise, we would observe some periods where the curves are flat, indicating that our client is stalled. Also, the way peer sets are built with “old” and “new” peers mixed together must have a positive impact on the efficiency of BitTorrent. Indeed, a torrent can be viewed as a collection of interconnected sets of peers. A peer that joins a torrent will obviously be the youngest peer in its initial set, but it may later be contacted by younger peers. It may also

be contacted by older peers since peers try to keep contact with a minimum number of peers (generally 20) throughout their lifetimes in the torrent. This temporal diversity is a key component of the efficiency of BitTorrent, since it guarantees with high probability that a given peer will find other peers (younger or older) that hold some missing chunks of the file.

- It takes twice as much the download period to upload the same quantity of bytes (1.77GB) to the system. This illustrates the importance of peers staying as seeds once they have completed the download. This means also that we downloaded at a faster rate than we uploaded. This is due to the fact that we have a high speed link and since seeds always seek for the fastest downloaders, we should be consistently favored by the seeds that serve us.

We further investigated the type of clients we have been trading with. Overall, we found that approximately 40% of the file was provided by seeds and 60% by leechers. We also observed that more than 85% of the total file was sent by only 20 peers, including the 8 seeds that provided 40% of the file. An interesting remark is that these 20 top uploaders were not in our initial peer set provided by the tracker, but contacted us later. We can thus conjecture that to obtain the best possible performance with BitTorrent, clients should not be behind firewalls or NATs that prevent inbound connections.

We also want to assess the efficiency of the tit-for-tat policy. A good tit-for-tat policy should enforce clients to exchange chunks with one another, but with enough flexibility so as not to artificially block transfers when data does not flow at the same rate in both directions. BitTorrent avoids this type of problem by chocking/unchoking connections every 10 seconds, a long period at the time scale of a single TCP connection. We have studied the correlations between upload and download throughput, as well as between upload and download traffic volumes. Results show that, while traffic *volumes* are correlated (positive correlation close to 0.5), the upload and download *throughputs* are not correlated (close to 0), which means that BitTorrent is flexible. We also computed the correlations between download volumes and download throughputs on one side and upload volumes and upload throughputs on the other side. Both are correlated with value 0.9 and 0.5, respectively. The high correlation observed between downloaded throughputs and volumes is probably due to the fact that the top three downloaders are seeds that provide 29% of the file. The upload throughput and volume are also correlated because, once our client becomes a seed, it continuously seeks for the best downloaders, which is not the case during the download phase where a peer primarily seeks for peers that have some chunks he is interested in.

6 Conclusion

Large content replication has become a key issue in the Internet; e.g. big companies are interested in a replication service to update simultaneously a large set of hosts (e.g., virus patches or software updates). BitTorrent is a very popular peer-to-peer application targeted at large file replication. We have extensively analyzed a large torrent (with up to a few thousands simultaneously active clients) over a large period of time. The performance achieved, in terms of the

throughput per client during download and the ability to sustain high flash-crowd, demonstrate that BitTorrent is highly effective.

There still remain open questions in the area of large content replication such as: (i) what is the optimal (at least theoretically) replication policy, in terms of response times, for a given user profile (arrival rates, willingness to stay as seeds), (ii) how to build a robust replication service, i.e. to ensure that all machines eventually complete the downloads (a must for corporate usage), (iii) how to efficiently protect these applications against denial of service and malicious peers.

Acknowledgment. We are extremely grateful to Eike Frost for kindly providing us with the tracker log of the torrent analyzed in this paper.

References

1. E. Adar and B. A. Huberman, “Free Riding on Gnutella”, *First Monday*, 5(10), October 2000.
2. L. Cherkasova and J. Lee, “FastReplica: Efficient Large File Distribution within Content Delivery Networks (USITS 2003)”, In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.
3. Y.-H. Chu, S. G. Rao, and H. Zhang, “A case for end system multicast”, In *ACM SIGMETRICS 2000*, pp. 1–12, Santa Clara, CA, USA, June 2000.
4. B. Cohen, “Incentives to Build Robustness in BitTorrent”, <http://bitconjurer.org/BitTorrent/bittorrentecon.pdf>, May 2003.
5. J. Jannotti, D. K. Gifford, and K. L. Johnson, “Overcast: Reliable Multicasting with an Overlay Network”, In *Proc. 4-th Symp. on Operating Systems Design and Implementation*, Usenix, October 2000.
6. S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, “Application-Level Multicast Using Content-Addressable Networks”, In *NGC 2001*, pp. 14–29, November 2001.
7. P. Rodriguez and E. W. Biersack, “Dynamic Parallel-Access to Replicated Content in the Internet”, *IEEE/ACM Transactions on Networking*, 10(4):455–464, August 2002.
8. A. Rowstron et al., “SCRIBE: The design of a large scale event notification infrastructure”, In *Proc. NGC 2001*, November 2001.
9. R. Sherwood, R. Braud, and B. Bhattacharjee, “Slurpie: A Cooperative Bulk Data Transfer Protocol”, In *Proceedings of IEEE INFOCOM*, March 2004.

A Measurement-Based Traffic Profile of the eDonkey Filesharing Service

Kurt Tutschku

Institute of Computer Science, University of Würzburg,
Am Hubland, D-97074 Würzburg, Germany.

tutschku@informatik.uni-wuerzburg.de

Abstract. Peer-to-peer file sharing applications have evolved to one of the major traffic sources in the Internet. In particular, the eDonkey file sharing system and its derivatives are causing high amounts of traffic volume in today's networks. The eDonkey system is typically used for exchanging very large files like audio/video CDs or even DVD images. In this report we provide a measurement based traffic profile of the eDonkey service. Furthermore, we discuss how this type of service increases the "mice and elephants" phenomenon in the Internet traffic characteristics.

1 Introduction

Peer-to-peer (P2P) file sharing applications have evolved to the major traffic sources in the Internet. In particular, the eDonkey2000 P2P file sharing system [1] and its derivatives [2,3] are causing high amounts of traffic volume [4]. The eDonkey¹ system is typically used for exchanging very large files like CDs or even complete DVDs images. The service is highly robust and obtains considerable short download times. P2P file sharing traffic is considered to be hazardous for networks. This view is mainly due to the high traffic volume but also caused by the transfer of very large files. The latter feature might increase the "mice and elephants" phenomenon in Internet traffic [5,6]. The phenomenon describes that the traffic consists of mainly short transfers (referred to as "mice") and long transfers (referred to "elephants"). Elephant streams are considered harmful for the network since they clog the system whereas mice may reduce the throughput if issued with high frequency [7].

The aim of this paper is to provide a traffic profile for the eDonkey service. The focus of the study is on the distinction of non-download traffic and download traffic. In addition, we discuss the "mice and elephants" characteristic in eDonkey and the origin and destination of eDonkey flows. The paper is organized as following. Section 2 outlines the eDonkey architecture and protocol. Section 3 describes at briefly the measurement setup and focuses on the measurements. Section 4 discusses related work on P2P behavior and traffic models. Section 5 summarizes the measurement results and provides a brief outlook.

¹ In this paper we subsume eDonkey2000 and all its derivatives by the single term eDonkey.

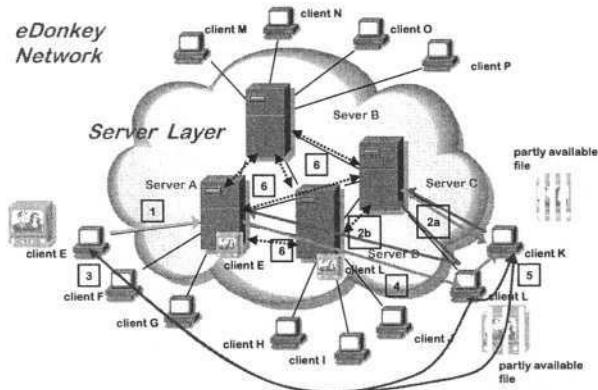


Fig. 1. eDonkey Communication

2 The eDonkey P2P File Sharing Service

The main features of eDonkey P2P file sharing application are: *a)* it doesn't rely on a single central server, *b)* a file can be downloaded from several different peers at once, and *c)* a file can be shared by a peer before it is completely obtained. The eDonkey protocol appears is not officially documented. A details have been obtained recently through reverse engineering [2,3,8].

Architecture and Operation: The eDonkey file sharing service belongs to the class of *hybrid P2P* architectures. Its architecture comprises two applications which form the eDonkey network: the eDonkey client² and the eDonkey server, cf. Figure 1. The eDonkey client is used to share and download files. The eDonkey server operates as an index server for file locations and distributes addresses of other servers to clients³. In the eDonkey network no files are transmitted through the server. Every eDonkey user is eligible to setup a server.

Searching and Sharing of Files: When a client connects to the eDonkey service, it logs on to one of the servers (using a TCP connection) and registers all files it is willing to sharing, cf. **[1]** in Figure 1. Each server keeps a list of all files shared by the clients connected to it. When a client searches a file, cf. **[2a]** in Figure 1, it sends the query to its main server. The server returns a list of matching files and their locations. The client may resubmit the query to another server, cf. **[2b]**, if none or an insufficient number of matches have been returned. The major communication between client and server is typically implemented by TCP connections on port '4661'. Additional communication between clients and servers, e.g. further queries and their results, are transmitted via UDP on port '4665'.

² The terms "client" and "peer" are exchangeable in the context of eDonkey.

³ In addition, eDonkey clients may also distribute server addresses among each other.

Downloading of Files: When an eDonkey client decides to download a file, it first gathers a list of all potential file providers and then asks the providing peers for an upload slot, see [3] in Figure 1. Upon reception of a download request, the providing client places the request in its *upload queue*. A download request is served as soon as it obtains an upload slot. eDonkey clients may restrict their total upload bandwidth to a given limit. An upload slot comes available when a minimum fair share of the upload limit is possible. When an upload slot is available, the providing client initiates a TCP connection to the requesting client, negotiates which chunk of the file is exchanged, and transmits the data.

The eDonkey protocols splits the file into separate pieces, denoted as *chunks*. A chunk has typically a size of 10MBytes. The consuming client can reassemble the file using the chunks or parts of chunks. A client can share a file as soon as it has received a complete chunk, see [4] in Figure 1. A major feature of eDonkey is that the consuming client may operate in the *multiple source download* mode, cf. [5] in Figure 1. In this mode, the downloading client issues in parallel two or more requests to different providing clients and retrieves data in parallel from the providers.

Since an eDonkey client may leave the eDonkey service at any time, the requesting client has to renew its download request periodically otherwise the requests are dropped. In order to reliably check the availability of a client, the eDonkey protocol uses TCP connections on port ‘4662’ for the communication between the clients. A client-to-client connection is terminated by the eDonkey application after an idle period of 40sec. It is worth to be mentioned here, that other P2P file sharing applications like Bearshare [9] or KaZaA [10] have implemented similar *multiple source download* schemes.

Server-to-Server Communications: The communication between eDonkey servers is very limited, cf. [6] in Figure 1. The servers contact each other periodically but with small frequency in order to announce themselves and to send back a list of other servers. In this way the servers maintain an updated list of working servers and affirm the search efficiency of the eDonkey service.

3 eDonkey Traffic Profile

3.1 Measurement Setup

The measurements in this paper have been carried out in Aug. 2003 over a duration of 296h on a 100Mbps, half duplex FE link connecting the department with the university’s campus LAN. The Internet connection of the university is a 155Mbps link to the German Research Network (DFN). The measurements were performed on flow level using TCPdump which was configured to record all TCP flows on the eDonkey client-to-client port ‘4662’. The flows were classified in an semi-off-line procedure into non-download streams and download flows, which contain at least one of the eDonkey / eMule protocol opcodes ‘OP_SENDINGPART’ or ‘OP_COMPRESSEDPART’.

Table 1. General Data on the Investigated eDonkey Data Set

number of observed TCP connections on port '4662'	3431743
number of local hosts	25
number of foreign hosts	242067
total transmitted volume in all flows	$2.95 \cdot 10^{11}$ bytes
total transmitted volume in download connections	$2.08 \cdot 10^{11}$ bytes (70.5%)
number of download connections	77111 (2.24%)
number of inbound download connections	21344 (27.7%)
number of outbound download connections	55767 (72.3%)

Since the eDonkey protocol is semi-proprietary, it can't be excluded that the observed non-download flows contain also download traffic. The analysis given below show that a misclassification is quite unlikely. For the rest of the paper we denote a TCP connection as *inbound* if it was initiated by a eDonkey client residing outside the department network. A TCP connection is said to be *outbound* if it was initiated by a client inside the department's LAN.

3.2 Traffic Profile

Table 1 provide general statistic values on the data set of the measurement. In total almost 3.5 million flows have been investigated which were carrying 295 Gbyte of data (non-download and download). Only 2.24% of all connections were download connections. However, they were carrying 705% of the total traffic.

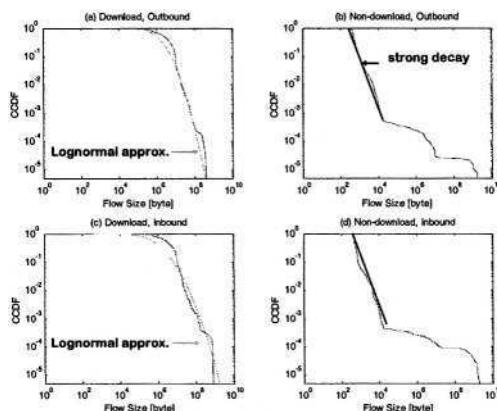
eDonkey Flowsize: The average observed eDonkey flow size during the measurements was 86Kbytes, cf. Table 2. A more detailed inspection shows that the average size of download streams (2.48Mbytes) is two orders of magnitudes larger than the average size of non-download streams (16.7Kbytes). This feature doesn't change much when the direction of the flows is considered, i.e. it doesn't differ for inbound and outbound flows. Figure 2 depicts the complementary cumulative distribution function (CCDF) of the flow sizes. Part (a) and (c) of Figure 2 shows that the download flow size decreases stronger than linear in the log/log plot. That means that the flow sizes don't show a strong "heavy tailed" feature. An approximation of the observed data with a lognormal distribution achieves a good estimate. The reduced strength of the heavy tail feature is not expected, but can be explained: the download flows are limited due to the segmentation of files into chunks and due to the application of the multiple source download principle.

Part (b) and (d) of Figure 2 depicts the size of non-download flows. The probability that a flow is larger than a given value decreases almost exponentially until a limit of approx. 14Kbytes. Beyond this limit, the decrease is not regular. This is an expected behavior since non-download flows are typical signalling flows to renew requests. The above observed features in the flow sizes indicate that the "mice and elephants" phenomenon has not been worsen by eDonkey .

Table 2. eDonkey Flow Statistics

	average	std. deviation
TCP connection interarrival time (all directions)	0.310 sec	0.379 sec
download TCP connection interarrival time (inbound)	49.9 sec	61.4 sec
download TCP connection interarrival time (outbound)	19.1 sec	23.2 sec
non-download TCP connection interarrival time (inbound)	0.830 sec	1.04 sec
non-download TCP connection interarrival time (outbound)	0.515 sec	0.745 sec
flow size (all directions)	86.0 kbytes	5.79 Mbytes
download flow size (inbound)	3.28 Mbytes	15.8 Mbytes
download flow size (outbound)	2.48 Mbytes	5.32 Mbytes
non-download flow size (inbound)	42.3 kbytes	7.17 Mbytes
non-download flow size (outbound)	15.7 kbytes	4.49 Mbytes
TCP connection holding time (all directions)	67.9 sec	265 sec
download TCP connection holding time (inbound)	1010 sec	1460 sec
download TCP connection holding time (outbound)	851 sec	1500 sec
non-download TCP connection holding time (inbound)	47.7 sec	39.2 sec
non-download TCP connection holding time (outbound)	49.7 sec	78.4 sec
plain bandwidth (all directions)	109 bps	23.7 kbps
download plain bandwidth (inbound)	2.77 kbps	5.17 kbps
download plain bandwidth (outbound)	2.41 kbps	2.55 kbps
non-download plain bandwidth (inbound)	44.9 bps	4.61 kbps
non-download plain bandwidth (outbound)	59.9 bps	30.2 kbps
busy bandwidth (all directions)	716 bps	404 kbps
download busy bandwidth (inbound)	3.20 kbps	5.54 kbps
download busy bandwidth (outbound)	2.80 kbps	2.95 kbps
non-download busy bandwidth (inbound)	322 bps	4.75 kbps
non-download busy bandwidth (outbound)	878 bps	520 kbps

TCP Holding Time: The average eDonkey connection holding time on TCP level is 67.9 sec, cf. Table 2. As for the flow sizes, there is a significant difference between download and non-load flows. The mean duration of download connections is 851sec. This more than one orders of magnitudes longer than the duration of non-download streams, which is 47sec. However, the standard deviation of the flow duration is much larger for download flows than for non-download

**Fig. 2.** CCDF of the observed eDonkey Flow Size

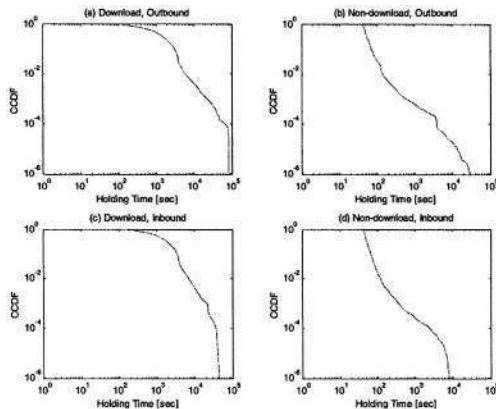


Fig. 3. CCDF of the observed eDonkey Flow Holding Time

streams. This is an expected behavior since non-download connections are short and limited in their sensitivity on TCP flow control. Figure 3 depicts the CCDF of the flow holding times on TCP level. The download connection holding time CCDFs (part (a) and (c)) decrease moderately and reminds more to a linear decay in a log/log plot. The CCDFs of the holding time of non-download stream (part (b) and (d)) decrease rapidly as well as un-regularly. There is only little difference in the non-download case for in the different directions.

Correlation of eDonkey TCP Flow Holding Time and TCP Holding Time: The Figure 4 depicts a scatter plot describing graphically the correlation of the TCP holding time and the size of eDonkey flows. Each dot in the scat-

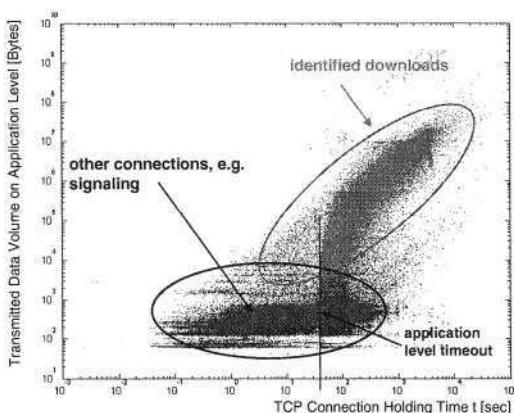


Fig. 4. Correlation of eDonkey TCP holding time and Flow Size

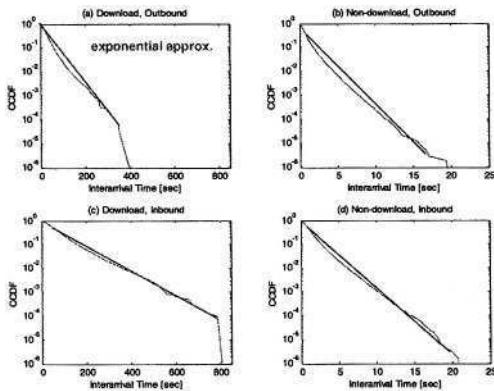


Fig. 5. eDonkey Flow Interarrival Time

ter plot represents an observed eDonkey flow. The brighter dots are identified download flows, the dark dots represent non-download connections.

The scatter plot shows that almost all identified download flows are within the same region. The overlap of both regions is small and therefore the probability of a misclassification is low. This feature enables the possibility to classify download streams by their size and holding time instead of using computationally demanding pattern recognition of protocol opcodes. Furthermore, the application level timeout of 40sec is clearly visible.

eDonkey Flow Interarrival Time: The average flow interarrival time was 0.310sec, cf. Table 2. There is again a significant difference for download flows and non-download streams since download flows are more rarely. The average inter-arrival time of download flows is two orders of magnitudes higher than the one of non-download flows. The CCDFs of the eDonkey flow interarrival time reveals an exponential decay, cf. Figure 5. This is consistent with resent eDonkey measurements [4]. The high frequency of non-download flows, in general, reduces the throughput on links [7].

Average Bandwidth on Application-Level: The average bandwidth of the eDonkey connections was also investigated. In the context of the herein presented measurements, the average bandwidth is defined as the ratio between the amount of transmitted data on application-level and the TCP connection holding time. This bandwidth is widely used and denoted in this paper as the *average plain bandwidth*. The analysis of the eDonkey flows, however, showed that a large number of connections have a significant idle period before the TCP connection is terminated by a FIN or a RST packet. From network point of view, the bandwidth is only experienced during data transmission. Therefore, the idle time is subtracted from the flow duration. The shorter times are used

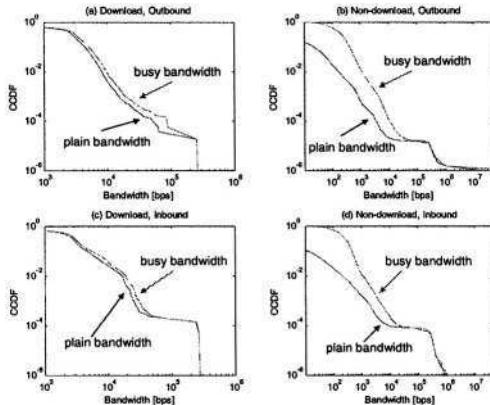


Fig. 6. eDonkey Average Bandwidth

for the calculation of the *average busy bandwidth*. We now compare the observed statistical values and distributions for both bandwidth definitions.

The average plain bandwidth for all eDonkey flows is 109bps, cf. Table 2. This value is very low and strongly influenced by idle period at the end of the TCP flows as the average net bandwidth of 716bps shows. The average plain bandwidth of download flows is typically two orders of magnitude higher than the plain bandwidth of non-download streams. The CCDF of the average plain bandwidth, lower curve in Figure 6, shows strong decay for download flows (part (a) and (c)) and a moderate decay for non-download flows (part (b) and (d)). The CCDF for the average busy bandwidth has a similar behavior however the decay is delayed and even stronger (part (b) and (d)). The comparisons shows that the influence of the idle time is less stronger for the average bandwidth of download flows as for the bandwidth of non-download streams. All features suggest to use the average busy bandwidth instead of the simple approach of the average plain bandwidth.

Origin and Destination of eDonkey Traffic: Finally, the origin and the destination of the observed eDonkey flows were investigated. The IP addresses of the foreign hosts were mapped to the Autonomous Systems (AS) which take care of these addresses. All traffic and connections for an AS were combined in Top7 lists. The Top7 list for the traffic amount, cf. Table 3, is dominated by the traffic originating or terminating in the AS of the German Telecom (DTAG). This characteristic of eDonkey indicates that majority of the observed traffic is locally and not world-wide distributed. Table 4 underlines this result for the number of established connections. A large number of established connections, however, does not necessarily mean a high traffic volume. This feature is caused by the eDonkey protocol requirement to renew download requests.

Table 3. eDonkey TOP 7 Autonomous Systems in Traffic Volume

Owner	Country	AS num.	total bytes	bytes download		bytes non-download	
				outbound	inbound	outbound	inbound
DTAG	.de	AS3320	50258 MB	15890 MB	4798 MB	9231 MB	20331 MB
Polish Tel.	.pl	AS5617	22703 MB	1761 MB	574 MB	9590 MB	10777 MB
France Tel.	.fr	AS3215	10527 MB	2353 MB	811 MB	2910 MB	4452 MB
BTnet UK	.uk	AS2856	8992 MB	1299 MB	720 MB	3197 MB	3776 MB
Verizon	.us	AS19262	6395 MB	0.877 MB	0.001 MB	3196 MB	3197 MB
Arcor IP	.de	AS3209	5579 MB	1133 MB	415 MB	1656 MB	2373 MB
NTL Grp. Ltd	.uk	AS5089	5224 MB	1055 MB	322 MB	1557 MB	2289 MB

4 Related Work

The traffic profile for the eDonkey service presented in this paper is a first step towards a more detailed behavior model and traffic model for the service. In general behavior model and traffic model of P2P services can classified into three main categories: models for multiple peer characteristics, models for the content or the shared resources, and models for the individual peer behavior. The models for multiple peer characteristics comprise characterizations for the P2P overlay network topology, e.g. the Power-Law feature in degree distribution of unstructured P2P networks [11], for the P2P overlay variability [12], and for wide-area network traffic pattern of P2P services [13]. The characterization of the content comprises models for the popularity of files and providing peers [14] and file size [15]. The individual peer behavior can be characterized by state models describing the idle, down, active, searching, or responding state of the peer [16]. A comprehensive traffic model for the Kazaa P2P file sharing service was investigated in [17].

5 Conclusion

In this paper we have presented a traffic profile for the eDonkey P2P filesharing service. The measurement-based study revealed a strong distinction between download flows and non-download stream. Both types of flows have to be considered differently. A single model for P2P flows, as provided in a first analysis in [13], would lead to a significant mischaracterization of the P2P traffic. It turned out that the traffic caused by eDonkey doesn't seem to worsen the "mice and elephants" phenomenon. However, a more detailed investigation is still necessary.

Table 4. eDonkey Top 7 Autonomous Systems in Connections

Owner	Country	AS num.	total conn.	num. conn. download		num. conn. non-download	
				outbound	inbound	outbound	inbound
DTAG	.de	AS3320	2114910	8518	12775	1048937	1044680
TDC	.dk	AS3292	207390	860	801	102835	102894
Arcor IP	.de	AS3209	178412	587	850	88619	88356
AOL Transit	.us	AS1668	176404	845	1342	87357	86860
France Tel.	.fr	AS3215	153900	1224	2415	75726	74535
TDE	.es	AS3352	140402	1145	1417	69056	68784
Polish Tel.	.pl	AS5617	131750	1076	828	64799	65047

In a next step we will define a detailed traffic model for eDonkey flows. The observed origins and destinations of eDonkey flows indicates that it is favorable for network operators trying to keep the traffic within their AS.

Acknowledgement. The author wants to thank M. Brotzeller for carrying out the measurements and P. Tran-Gia for supporting this research.

References

1. Meta Search Inc.- eDonkey2000 Home Page: (<http://www.edonkey2000.com/>)
2. eMule Project Team Web Site: (<http://www.emule-project.net/>)
3. mlDonkey Web Site: (<http://mldonkey.org/>)
4. Azzouna, N., Guillemin, F.: Analysis of ADSL traffic on an IP backbone link. (In: GLOBECOM 2003, San Francisco, California, Dec. 2003.)
5. Paxson, V., Floyd, S.: The failure of the Poisson assumption. IEEE/ACM Trans, on Networking (1995) 226-244
6. Bhattacharyya, S., Diot, C., Jetcheva, J., Taft, N.: Pop-level and access-link-level traffic dynamics in a tier-1 pop. (In: 1nd Internet Measurement Workshop, San Francisco, USA, Nov. 2001.)
7. Boyer, J., Guillemin, F., Robert, P., Zwart, B.: Heavy tailed M/G/1-PS queues with impatience and admission control in packet networks. (In: Proceedings of INFOCOM 2003, San Francisco, USA, April/March 2003.)
8. Lohoff, F.: Lowlevel documentation of the eDonkey protocol: (<http://silicon-verl.de/home/flo/software/donkey/>)
9. Free Peers Inc. - Bearshare: (<http://www.bearshare.com/>)
10. Sharman Networks - KaZaA Media Desktop: (<http://www.kazaa.com/>)
11. Ripeanu, M., Foster, I.: Mapping gnutella network. (In: 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, Massachusetts, March 2002.)
12. de Meer, H., Tuschku, K., Tran-Gia, P.: Dynamic Operation of Peer-to-Peer Overlay Networks. Praxis der Informationsverarbeitung und Kommunikation **26** (2003) 65-73
13. Sen, S., Wong, J.: Analyzing peer-to-peer traffic across large networks. (In: 2nd Internet Measurement Workshop, Marseille, France, Nov. 2002.)
14. Adar, E., Huberman, B.A.: Free riding on gnutella. Research report, Xerox Palo Alto Research Center (2000)
15. Saroiu, S., Gummadi, P., Gribble, S.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of Multimedia Computing and Networking 2002 (MMCN '02), San Jose, CA, USA (2002)
16. Schlosser, M., Condie, T., Kamvar, S.: Simulating a p2p file-sharing network. (In: 1st Workshop on Semantics in Peer-to-Peer and Grid Computing, Budapest, Hungary, May 2003.)
17. Gummadi, K., Dunn, R., Saroiu, S., Gribble, S., Levy, H., Zahorjan, J.: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. (In: Proceedings of 19th ACM Symposium on Operating Systems Principles, Bolton Landing (Lake George), USA, Oct. 2003.)

Bootstrapping in Gnutella: A Measurement Study

Pradnya Karbhari, Mostafa Ammar, Amogh Dhamdhere, Himanshu Raj,
George Riley, and Ellen Zegura

Georgia Institute of Technology, Atlanta, GA-30332

{pradnya, ammar, amogh, rhim, ewz}@cc.gatech.edu
riley@ece.gatech.edu*

Abstract. To join an unstructured peer-to-peer network like Gnutella, peers have to execute a *bootstrapping function* in which they discover other on-line peers and connect to them. Until this bootstrapping step is complete, a peer cannot participate in file sharing activities. Once completed, a peer's search and download experience is strongly influenced by the choice of neighbor peers resulting from the bootstrapping step. Despite its importance, there has been very little attention devoted to understanding the behavior of this bootstrapping function. In this paper, we study the bootstrapping process of a peer in the Gnutella network. We find that (1) there is considerable variation among various servent implementations, and hence in their bootstrapping performance. (2) The neighbors of a peer, which are the outcome of the bootstrapping process, play a very important role in the peer's search and download performance. (3) Even though the GWebCache system for locating peers is designed to operate as a truly distributed caching system, it actually operates more like a centralized infrastructure function, with significant load imbalance. (4) The GWebCache system is subject to significant misreporting of peer and GWebCache availability, due to stale data and absence of validity checks.

1 Introduction

To join an unstructured peer-to-peer network like Gnutella, peers have to execute a *bootstrapping function* in which they discover other on-line peers and connect to them. These initial neighbor peers determine the new peer's location in the overall Gnutella topology, and ultimately its search and download performance. Also, from the user perspective, the time spent by the peer in bootstrapping is critical because until the bootstrapping step is complete, a peer cannot participate in file sharing activities such as searching and downloading. From our experience, this time can vary significantly for different Gnutella servents¹.

Despite the significance of the bootstrapping process in unstructured peer-to-peer networks, it has received very little attention to date. There have been

* This work is supported in part by NSF grants ANI-0240485 and ANI-9973115.

¹ The implementations of Gnutella peers are referred to as *servents* because they function as servers and as clients. We use the terms *peers* and *servents* interchangeably.

various studies[1,2] aimed at characterization of peers based on their uptimes, bottleneck bandwidths, latencies and other factors, and trying to improve a peer's search and download experience[3]. However, none of these have studied the bootstrapping function.

Initially Gnutella users relied on word of mouth to determine the address of an on-line peer that would allow newly joining peers to tap into the network. The use of automated caching servers, as well as caching in the Gnutella servent itself, was introduced at a later time. As Gnutella gained in popularity after Napster was shut down, the caches ultimately became the pre-dominant bootstrapping technique [4]. Anecdotally, it has been observed that the switch from the use of word of mouth to the use of automated caches resulted in a significant change to the structure of the Gnutella network and a worsening of its performance[4].

In this paper, we undertake a measurement study of the current bootstrapping process in the Gnutella network. Our investigation consists of three parts:

1. An analysis and performance comparison of the bootstrapping algorithms of four Gnutella servent implementations: LimeWire[5], Mutella[6], Gtk-Gnutella[7] and Gnucleus[8].
2. A measurement-based characterization of the Gnutella Web Caching System[9] (GWebCaches), a primary component of bootstrapping algorithms.
3. A measurement-based analysis of the role of neighbor peers, resulting from different bootstrapping algorithms, in the search performance of a peer.

Based on our analysis of the data collected, we highlight below our four main findings about the current Gnutella bootstrapping system.

1. Although similar in the basic structure of the algorithm and the data structures used, the servent implementations differ in the details, with significant impact on their bootstrapping times, as seen in our measurements.
2. The neighbors of a peer play an important role in the search performance of the peer, thus pointing to the importance of the bootstrapping process.
3. An analysis of the request rates at different GWebCaches points to the disparity in traffic volume handled by these caches— some caches are very busy, and their host and cache lists evolve much faster than some others. The load balancing goal of any distributed system is not really achieved in this system.
4. The GWebCache system is subject to significant misreporting of peer and cache availability. This is because the data reported in the updates to these caches is not validated by the caches.

The rest of the paper is structured as follows. In Section 2, we give an overview of the bootstrapping process in different Gnutella servents, with special focus on the GWebCache system. We discuss the performance of the different servents with respect to their bootstrapping times in Section 3, and the role of the resulting neighbor peers in the search performance, in Section 4. In Section 5 we discuss the performance of the GWebCache system. In Section 6, we summarize our findings and discuss future work.

Table 1. GWebCache messages

Argument	Cache Response
<i>ping=1</i>	<i>pong</i> message to servent
<i>urlfile=1</i>	list of caches
<i>hostfile=1</i>	list of online hosts
<i>ip=<IPAddress></i>	host list is updated with IP address and port number
<i>url=<URL of cache></i>	cache list is updated with URL
<i>statfile=1</i>	access statistics over last hour

2 Gnutella Bootstrapping

In this section, we describe the bootstrapping process in the Gnutella servents we analyzed, and the functioning of the GWebCache system.

2.1 Gnutella Web Caching System

A peer intending to join the Gnutella network requires the IP addresses of online peers in the network. Currently, the GWebCache system functions as a distributed repository for maintaining this information. Peers can query the caches in this system to get a list of online peers to connect to. In the first execution of a particular Gnutella servent, the only means to locate other online peers is the GWebCache system. In successive executions, individual servent implementations try approaches apart from the GWebCaches, such as maintaining local lists of hosts seen during their earlier runs. We first discuss the GWebCache system, as it is an important component of the bootstrapping functionality, and is essential in the understanding of the servent bootstrapping algorithms.

The GWebCache system[9] is a network of voluntarily-operated caches that maintain a list of online peers accepting incoming connections. When a new peer wants to join the Gnutella network, it can retrieve the host list from one or more of these GWebCaches. The GWebCaches also maintain a list of other caches in the system. Typically each cache maintains a list of 10 other caches and 20 hosts that are currently accepting incoming connections.

The peers in the Gnutella network are responsible for keeping the information in these caches up-to-date; the caches do not communicate with each other at any time. A host accepting incoming connections is supposed to update the caches with its IP address and port number, and with information about some other GWebCache that it believes is alive. The GWebCaches maintain the host and cache lists as first-in-first-out lists.

Table 1 lists the messages sent by a client using the GWebCache protocol. An HTTP request of the form “URL?argument” is sent to the webserver at which the cache is located. The caches respond as shown in the table. Note that the GWebCaches do not maintain any information about the online hosts, other than their IP addresses and port numbers.

1. Initialize the following data structures in memory by reading the corresponding files from disk— list of caches, list of known hosts, list of permanent hosts and list of ultrapeer hosts (except in Gtk-Gnutella).
2. Depending on mode (ultrapeer/normal), determine the minimum number of connections to be maintained.
3. Try to establish the minimum number of connections to peers in the order:
 - In LimeWire and Gnucleus, try to connect to ultrapeers.
 - Try to connect to any host in the known hosts and permanent hosts lists.
 - If the servent is still not connected, request the host list from a GWebCache (multiple GWebCaches in LimeWire) and try to connect to these hosts.
4. Periodically, a connection watchdog checks whether the minimum num of connections (step 2) are alive. If not, try to establish a new connection (step 3).
5. Periodically update a cache with its own IP address and URL of another cache (for LimeWire and Mutella, this is done only if in ultrapeer mode)
6. On shutdown, write the different files to disk, for retrieval on next startup.

Fig. 1. Generic bootstrapping algorithm**Table 2.** Servent implementation differences

Characteristic	Limewire	Mutella	Gtk-gnutella	Gnucleus
Maintains ultrapeers list?	Yes	Yes	No	Yes
Prioritize ultrapeers when connecting?	Yes	No	No	Yes
Host & cache lists prioritized by age?	Yes	No	Yes	No
Updates to GWebCaches	Ultrapeer mode	Ultrapeer mode	Any mode	Any mode
Number of hardcoded caches	181	3	3	2

2.2 Servent Bootstrapping Algorithms

In this section, we discuss the bootstrapping algorithms of the Gnutella servents that we compared, and point out the differences between them. We analyzed Limewire v2.9[5], Gtk-Gnutella v0.91.1[7], Mutella v0.4.3[6] and Gnucleus v1.8.6.0[8]. All these versions support retrieval from and updates to the GWeb-Cache system. The bootstrapping processes in the four servents are similar in their use of the GWebCache system and the local caching of hosts.

The data structures maintained by these servents include a list of known GWebCaches, which is periodically populated with addresses of new GWeb-Caches. Servents also maintain lists of *known* hosts and *permanent* hosts, the definitions of which differ slightly in different servents. Informally, permanent hosts are hosts that the servent managed to contact in current and previous runs. Some servents also maintain a separate list of ultrapeers².

² Ultrapeers[10] are hosts with high bandwidth and CPU power, and long uptimes. Normal or leaf nodes have lower capabilities and typically connect to ultrapeers.

Figure 1 outlines the generic bootstrapping algorithm, and Table 2 summarizes the differences in the implementations, as discussed below.

1. Limewire and Gnucleus maintain a list of ultrapeers and give priority to hosts in this list during connection initiation. Since ultrapeers have relatively long uptimes and the capability to support more incoming connections, prioritizing these peers during connection initiation increases the probability of successfully connecting to a peer. Although Mutella also maintains a list of ultrapeers, this information is not used during bootstrapping. Gtk-Gnutella does not distinguish between ultrapeers and normal peers.
2. LimeWire and Gtk-Gnutella prioritize their host and cache lists by age. This enables them to act on more recent (and hence more accurate) information.
3. Although all four servents we examined support the GWebCache system for retrieving information, LimeWire and Mutella support updates to the GWebCaches only in the ultrapeer mode. This is better for the system because the probability of ultrapeers accepting incoming connections is higher than when in the leaf mode. Gtk-Gnutella and Gnucleus update the GWebCaches even in the leaf mode.
4. The Gnutella servents have a set of hardcoded caches, which are used during the very first run of the servent, before any other information about caches or hosts is known. As seen in Table 2, compared to other servents LimeWire has a very high number of hardcoded caches (181), 135 of which were active when we tried to ping them at the Gnutella level.

In the next section, we will discuss the effects of these differences in bootstrapping algorithms on the performance of different servent implementations.

3 Bootstrapping Measurement at Servent

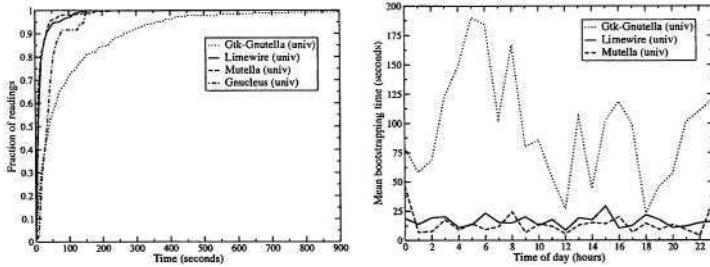
In this section, we compare the performance of the servents considered in our study, based on their *bootstrapping times*. We define the bootstrapping time of a servent as the time between the start of the servent and the initial establishment of the first *stable* Gnutella-level connection³. We say that a connection is *stable* if it is maintained for at least *threshold* seconds.

3.1 Measurement Methodology

We modified the sourcecode of the three Linux-based servents (LimeWire, Gtk-Gnutella and Mutella) to log the times at which the application was started and shut down, and when a Gnutella-level connection was established and terminated. For the Windows-based servent (Gnucleus), we used Windump[11] to collect packet traces, and then analyzed them to determine the connection times.

We started the Linux-based servents once every hour, synchronously at two locations, at a university campus on a Fast Ethernet Link and at a residence on

³ A “Gnutella-level” connection is established after the Gnutella handshake messages are exchanged between the two connecting peers.



(a) CDF of servent bootstrapping times

(b) Mean bootstrapping times acc. to time of day

Fig. 2. Servent bootstrapping times

a DSL link to a local ISP. We started Gnucleus once every three hours at the university location only. Each servent was allowed to run for 15 minutes, after which it was shut down. In the following section we analyze the bootstrapping times measured during an 11-day experiment. One limitation of our study is that both locations in our experiments have high bandwidth access links. We did not run any experiments at a slower access link.

3.2 Performance Measurements

Figure 2(a) shows the cumulative distribution function of the bootstrapping times of the four servents at the university location. In this graph we set *threshold* to 120 seconds. We analyzed the bootstrapping times with different values for *threshold* and observed similar results. The graphs for the bootstrapping times of servents on the DSL link are similar.

The most striking observation is that Gtk-Gnutella performs much worse than Mutella and LimeWire. We conjecture that this is due to the fact that Gtk-Gnutella does not differentiate between ultrapeers and normal peers. Also, once it selects a particular GWebCache to contact for retrieving the host list, it uses it for 8 consecutive updates or retrievals. In Section 5, we will see that cache quality varies; thus, maintaining a poor choice of cache can affect performance. Gnucleus also performs worse than Mutella and LimeWire, but better than Gtk-Gnutella. This is probably because the GWebCache list and the different host lists are not prioritized by age in the Gnucleus implementation.

Figure 2(b) shows the mean bootstrapping times for the three Linux-based servents at the university location for different times of the day. LimeWire and Mutella perform almost the same throughout the day. Gtk-Gnutella, which does not differentiate between ultrapeers and normal peers, performs similar to LimeWire and Mutella around noon or late afternoon, when there are more normal peers online in the system. Early in the morning, with very few normal peers

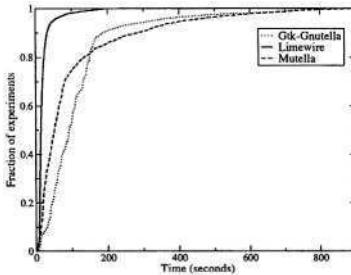


Fig. 3. Time to receive first queryhit

around, Gtk-Gnutella shows a higher mean bootstrapping time. This highlights the importance of ultrapeer awareness on the part of a Gnutella servent.

Although we started multiple instances of Gnutella servents on the same local area network, none of our peers were able to discover each other in any one of our experiments over two weeks. This highlights the lack of Internet location awareness in the GWebCache system and in the local host list of the servents.

In the next section, we discuss the importance of neighbor peers (resulting from the bootstrapping process) in the search performance of peers.

4 Importance of Neighbor Peers

A peer gets access to the peer-to-peer network through its directly connected neighbors. The peers that these neighbors have access to, within an N -hop radius (usually $N=7$), comprise the *neighborhood* of the peer. All query messages originated by the peer will be forwarded to this neighborhood. The number of peers in this neighborhood, the types of files shared, the number of files shared amongst all these peers will reflect on the search performance of a peer.

We studied the effect of neighbors on search performance of LimeWire, Mutella and Gtk-Gnutella for the the 15 most popular search queries [2]. The performance metric we considered is the time to get the first response, which is the time-lag from when the servent is bootstrapped and issues the query, to the time when it gets the first response. Figure 3 shows the CDF of this response time for the top 15 queries issued by that servent during any experiment.

We found that there is usually a significant variation in the time to get the initial response. LimeWire performs the best, primarily because during bootstrapping it prioritizes ultrapeers, who usually have access to a larger neighborhood. Mutella and Gtk-Gnutella perform worse, and take more than 5 minutes to give a result, in about 10% of the experiments.

We conclude that for a good search experience it is very important to have a set of good neighbors that provide access to many peers, sharing many files.

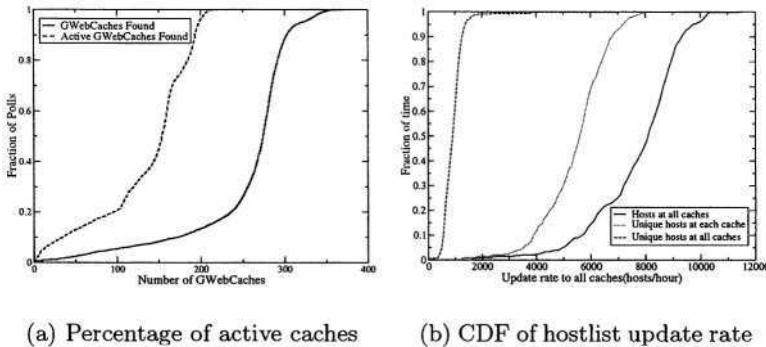


Fig. 4. Cache and host list update rates in all GWebCaches

5 GWebCache Performance

We analyzed the performance of the GWebCache system with reference to the properties of a good distributed caching infrastructure (e.g., sufficient total system capacity, good load balancing, reliability of cached items, and physical or topological proximity of cached items served). With this goal in mind, we performed a measurement study of the system at two levels, globally and locally.

5.1 Global GWebCache System Performance

We studied the global GWebCache system by periodically crawling all the caches. We sent requests in the format shown in Table 1 to each active cache, according to the information required. We collected multiple traces over a five month period (Apr-Sept 2003), with the goal of answering the following questions.

1. How many GWebCaches does the system comprise of? How many of the reported caches are active at any time?

We retrieved the *cache list* every 30 minutes, starting with a seed GWebCache and crawled the caches returned, until we had polled all reachable caches. We also determined the number of active GWebCaches by sending Gnutella *ping* messages to the crawled list.

Although we found URLs of 1638 unique GWebCaches in 5 months, only one-fourth of them (403) were active at any time, and at most 220 of them were active during a single poll. This is quite a low number of reachable GWebCaches, potentially serving about 100000 active hosts⁴ at any time on the Gnutella network, only 10% of which accept incoming connections. This indicates that the GWebCache system might get overloaded.

Figure 4(a) shows the CDF of the number of GWebCaches found during each of our polls, and the number of caches which were actually active (i.e. responded to our Gnutella-level *ping* messages). Most of the time, only about 160 caches

⁴ As shown by the Limewire[5] hostcount, during the period of our study.

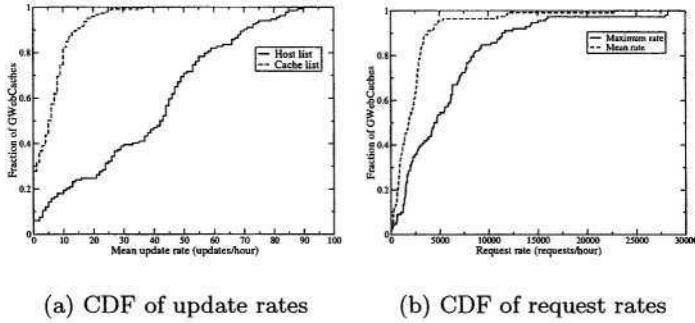


Fig. 5. Update and Request rates at a single GWebCache

out of 280, or about 60% were active. This is because the GWebCache system does not have any means of deleting an introduced cache. Since peers update caches with URLs of caches they know of (probably even cached from previous runs), without necessarily knowing whether they are alive or not, it is quite likely that inactive caches are reported for a long time.

2. *What are the access patterns for different requests (cache list, host list, and updates) at different GWebCaches? What are the differences in access patterns across different GWebCaches and in the whole system?*

We retrieved the *statistics file* every hour from each active GWebCache. The statistics file gives the number of update requests and total requests for cache and host lists the GWebCache received within the last hour.

Figure 5(a) shows the CDF of the mean update rates to the cache and host lists (determined by analyzing lists retrieved in consecutive polls) at a single GWebCache. About 80% of the GWebCaches get cache list update rates of 10 per hour or less, while a few caches receive very high update rates, upto 40 updates per hour. About 60% GWebCaches receive host list update rates of less than 1 per minute, whereas others receive update rates almost twice as much. Similarly, Figure 5(b) shows the CDF of the mean and maximum total request rates (as reported by the statistics files) at a single GWebCache. About 90% of the GWebCaches receive an average request rate of 3000 per hour or less, whereas some caches receive extremely high loads of the order of 20000 requests per hour on an average, with a maximum of 30000 requests per hour.

This points to the disparity in the type of GWebCaches in the system. Some caches are very busy, with their lists evolving faster and receiving high request rates, whereas others are relatively lightly loaded. The servants we studied have some hardcoded GWebCaches, indicating that the request rates to these caches could be very high. This suggests poor load balancing in the GWebCache system.

3. *How does the host list at a single GWebCache and at all GWebCaches evolve? What percentage of new hosts added to the GWebCaches are unique?*

We retrieved the *host list* from the active GWebCaches every 5 minutes, and studied its evolution at a particular cache and in the whole system. As

expected, the host list evolves much faster than the cache list in any GWebCache. During a 15-day period in our study, we saw over 300000 unique IP address:port combinations in all GWebCaches.

Figure 4(b) shows the CDF of the host update rates at all GWebCaches in the system. The rightmost line shows the CDF of the host updates received at all GWebCaches in the system. The dotted line shows the CDF of the host updates with unique IP address:port combination at each cache. The leftmost curve with the dashed line shows the CDF of the unique IP address:port combination seen in the whole system. The average rate for unique IP address:port updates at a particular GWebCache is lower than the actual update rate at that cache. The update rate for IP address:port, unique throughout the system is much lower, almost by a factor of 10. This suggests that the same hosts (presumably ultrapeers) update the GWebCaches frequently with their IP addresses, leading to a high replication rate of the same addresses in multiple caches.

4. In the host list returned by the GWebCaches, how many hosts are alive, how many are accepting Gnutella-level connections, and how many are ultrapeers?

We sent Gnutella-level connect messages to the hosts in the host lists returned by the GWebCaches. If a TCP connection was established, we determined that the host was alive. If a Gnutella-level connection was established, we determined that the host was accepting incoming connections. Out of the hosts that responded with the proper *pong* response, we determined whether the host was an ultrapeer or not, using a field *X-Ultrapeer: True/False* in the response.

When we tried connecting to the hosts in the host lists retrieved, on an average we found 50% peers online, 16% peers accepting incoming Gnutella-level connections, and 14% ultrapeers. This shows that a surprisingly low number of peers indicated in the GWebCaches are actually accepting incoming connections. This could be a cause for the high bootstrapping times of servents in some cases, where peers waste time trying to connect to off-line hosts returned by the GWebCaches. The reliability of content served by the GWebCache system is therefore questionable.

Our measurement methodology has several limitations. Since we polled the GWebCaches starting with a seed GWebCache, we will miss caches in any disconnected components of the GWebCache system. Also, between the times we retrieved the list and tried connecting to the peer, the peer could have gone offline. We assume that the information returned by the GWebCaches during our polls is valid (i.e., the GWebCaches are not misconfigured or misbehaving).

5.2 Experience of a Local GWebCache

We set up a GWebCache locally by modifying a publicly available PHP script for the GWebCache v0.7.5[9] to log request arrivals, and advertised it to the global caching infrastructure. Our cache received update rates of about 7 per hour to the host list and 4 per hour to the cache list, and request rates of about 15-20 per hour for the host list and 5-10 per hour for the cache list. Comparing these rates to those of other GWebCaches seen earlier, we can see that our local cache is used less frequently than the other GWebCaches.

6 Conclusions

In conclusion, our study highlights the importance of understanding the performance of the bootstrapping function as an integral part of a peer-to-peer system. We find that (1) Although servents implement a similar structure for the bootstrapping algorithm, there is considerable variation among various implementations, that correlates to their bootstrapping performance. (2) The neighbors of a peer play an important role in the search performance of the peer. Hence it is important that the bootstrapping process results in good neighbors. (3) Even though the GWebCache system is designed to operate as a truly distributed caching system in keeping with the peer-to-peer system philosophy, it actually operates more like a centralized infrastructure function, with some GWebCaches handling a large volume of requests while others are idle. (4) The GWebCache system is subject to significant misreporting of peer and GWebCache availability due to stale data and absence of validity checks. We further aim to analyze the effect of bootstrapping on the evolution of the Gnutella topology. These studies will lead to our ultimate goal of improving the bootstrapping process in unstructured peer-to-peer networks like Gnutella.

References

1. Saroiu, S., Gummadi, P., Gribble, S.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of Multimedia Computing and Networking. (2002)
2. Chu, J., Labonte, K., Levine, B.: Availability and locality measurements of peer-to-peer file systems. In: Proceedings of ITCom: Scalability and TrafficControl in IP Networks. (2002)
3. Ng, T.E., Chu, Y., Rao, S., Sripanidkulchai, K., Zhang, H.: Measurement-based optimization techniques for bandwidth-demanding peer-to-peer systems. In: Proceedings of IEEE Infocom. (2003)
4. Oram, A.: Peer-To-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly (2001)
5. LimeWire. (<http://www.limewire.com>)
6. Mutella. (<http://mutella.sourceforge.net>)
7. Gtk-Gnutella. (<http://gtk-gnutella.sourceforge.net>)
8. Gnucleus. (<http://www.gnucleus.com>)
9. Gnutella Web Caching System. (<http://www.gnucleus.com/gwebcache>)
10. Ultrapeer Specifications. (<http://www.limewire.com/developer/Ultrapeers.html>)
11. Windump. (<http://windump.polito.it>)

Testing the Scalability of Overlay Routing Infrastructures

Sushant Rewaskar and Jasleen Kaur

University of North Carolina at Chapel Hill,
`{rewaskar,jasleen}@cs.unc.edu`

Abstract. Recent studies have demonstrated the utility of alternate paths in improving connectivity of two end hosts. However studies that comprehensively evaluate the tradeoff between its effectiveness and overhead are lacking. In this paper we carefully characterize and evaluate the trade-off between (1) the efficacies of alternate path routing in improving end-to-end delay and loss, and (2) the overheads introduced by alternate routing methodology. This would help us to test the scalability of an overlay network.

We collected ping data on *PlanetLab* and studied the above trade-off under different parameter settings such as path sampling frequency, overlay-connectivity, number of overlay hops etc. Results from ten sets of measurements using 35 to 36 of the *PlanetLab* nodes are used to test the effect of the various parameters. We find that changing epoch duration and connectivity helps reduce the overheads by a factor of 4 and 2 respectively at the cost of some lost performance gain.

1 Introduction

Recently, the idea of using overlay networks to find routes between a pair of Internet hosts has received much attention [1,2,3]. In these works, hosts associate with the “nearest” node belonging to an overlay network, and route traffic through that node. The overlay node forwards traffic to the specified destination along the current best overlay path in such a manner as to provide better end-to-end services such as delay, loss, jitter, and throughput. In order to find the best path, overlay nodes will: (i) monitor the state of routes to all overlay nodes, (ii) periodically exchange this state information with neighboring overlay nodes, and (iii) compute an up-to-date snapshot of the best paths between any pair of overlay nodes using a link-state routing protocol. Clearly, the gains in end-to-end performance achieved through the use of an overlay network come at the cost of processing and transmission overhead (overlay state monitoring and distribution, and path computation). Past work has shown that it is possible to construct an overlay mesh of up to 50 nodes without incurring significant overheads [2]. In this study, we ask the question: *can one scale an overlay up to a larger number of nodes?*

The fundamental issue is that the tradeoff between the performance gains and overheads of an overlay routing infrastructure is governed by several factors, including (i) the number of nodes in the overlay, (ii) the average fraction

of “neighbors” of a node in the overlay with whom a node exchanges routing information (referred to as *logical connectivity* or simply *connectivity*), (iii) the frequency with which routes in the overlay network (“overlay links”) are computed (the duration between route updates is referred to as an *epoch*), and (iv) the maximum number of overlay “hops” used in computing routes. Past work has primarily investigated single points in this parameter space in wide-area Internet measurements. Here we attempt a more comprehensive analysis of the limits and costs of the scalability of an overlay routing infrastructures by a more controlled study of the parameter space. Our specific goal is to identify points in the parameter space that would enable the operation of larger overlay routing infrastructures with acceptable overheads and satisfactory performance gains.

We have conducted an extensive measurement and simulation study of a wide-area overlay routing infrastructure using PlanetLab. Based on this study our main findings are:

1. Reducing the average logical connectivity of overlay nodes by a factor of 2, reduced overlay routing maintenance and messaging overhead by almost a factor of 4, while reducing by only 40% the number of overlay routes having better latency than the default path, and reducing by only 30% the number of overlay routes having lower loss rates than the default route.
2. Doubling epoch duration reduces overhead by 50%. However, as epoch duration increases, routing data becomes more stale (less valid) and the performance of the overlay routes computed based on this data is less certain. For large epoch duration, computed overlay routes underperform nearly 30% of the time for loss rates and nearly 10% of the time for latency. The selection of wrong routes due to stale information is termed as mis-predictions.
3. Increasing the number of nodes in the overlay by 33% increases the number of better paths by 5 to 10% while increasing the overhead by about 60% in the worst case (when nodes have 100% logical connectivity).
4. Using more than one intermediate path for calculating an overlay route does not provide much benefit.

The rest of this paper is organized as follows. In section 2 we place our work in context of other related work, highlighting the similarity and differences. Section 3 describes the experimental and analysis methodology. The dataset is described in brief in section 4. We present the results in section 5. We summaries our results and describe scope for future work in Section 6.

2 Related Work

Overlay networks provide a framework for building application-layer services on top of the existing best-effort delivery service of the Internet. Key to this work is the understanding of the dependence of end-to-end performance on route selection on the Internet. This problem has been studied extensively. For example, the detour[5] study quantifies the inefficiencies present in the Internet direct default paths. They collected data over a period of 35 days over 43 nodes. Based

on this data they concluded that overlay paths can improve network performance. The Savage *et al.* [3] study conducts a measurement-based study where they have compared the improvement in performance that can result from the use of an alternate path instead of the direct default path. They studied various metrics such as loss and round trip delay over a set of diverse Internet hosts. They conclude that alternate paths could be helpful in 30 to 80% of the cases. Most recently Anderson *et al.* [2] showed the use of an overlay network to achieve quick recovery in case of network failures. Unlike detour, the RON work analyzed the use of alternate paths for improving the performance on a time scale small enough to reflect the actual dynamic changes in the Internet.

All these papers considered only a single point in the parameter space to study system performance. None studied the effect of varying the parameters and their effects on the various metrics. For example, based on a single point in the parameter space, [2] predicts the size to which such a network could be scaled.

3 Overlay Network Emulation Facility

3.1 Network Node Architecture

We emulate an overlay network by gathering ping data from a collection of nodes in a real wide-area network and then subsampling the data as appropriate to emulate an overlay network with the desired degree of logical connectivity between nodes. Each node in the network runs two programs: a “ping” module that emulates the probing mechanism in an overlay and a “state exchange” module that emulates mechanism used to propagate measurement data to other nodes.

At the beginning of each epoch, the ping module collects ping measurements to all other nodes. It then summarizes this ping information and records it in a local file for off-line analysis. Following each 24 or 48 hour measurement period, data from the nodes is collected and analyzed. To emulate different average connectivity of nodes, the analysis considers only information that would be acquired about neighboring nodes. The analysis tool uses the ping summaries to compute, for each service metric such as delay and loss, all “better” overlay routes to all other overlay nodes. Only those alternate paths are considered that use one of the neighbors as the next-hop. We also ran the ping module on each node and used a link state routing protocol to flood the summary of the measurements to all nodes. The bytes exchanged by this module was used to determine the state-exchange overhead.

3.2 Parameters

The tradeoff between performance gains and overheads of a routing overlay is governed by several factors:

Epoch: This is the interval at which the probes are run and routing updates are conducted. The larger is the epoch duration, the less frequent are route computations, and smaller are the overheads. On the other hand, smaller epochs help maintain an up-to-date view of network state and best routes.

Average connectivity: The larger is the set of neighbors used to re-route traffic, the greater is the likelihood of finding better alternate paths. However, the overhead of exchanging state and computing best paths also increases with connectivity.

Overlay size: The larger is the number of nodes in the overlay, the greater is the likelihood of finding better alternate paths. However, the overheads grow as well. In fact, our main objective in this study is to find out if the other parameters can be tuned to allow the operation of larger overlay.

Maximum length of overlay routes: We expect that considering only routes that traverse no more than 2-3 overlay links will be sufficient for locating better paths, if there are any.

Set of service metrics: The likelihood of finding better alternate paths between a pair of overlay nodes is a function of the service metrics - such as delay, loss, and jitter - of interest. Furthermore, applications that desire good performance in terms of more than one metric stand a smaller chance of finding paths that do better than the direct paths. The overheads remain unaffected by the set of service metrics.

These are the set of parameters what can be tuned to increase or decrease the performance gains and overheads of the overlay for desired results.

3.3 Metrics

We test the scalability limits of overlay routing infrastructures by conducting several experiments with different settings of the parameters mentioned in previous section and measuring the gains and overheads from each. Gains and overheads are quantified as follows.

Gain Metrics. Four metrics are used to quantify the performance benefits achievable with overlay routing:

1. The number of node-pairs for which there is at least one alternate path that is better than the direct path.
2. The number of better alternate paths for a node pair.
3. The degree of performance improvement achieved by using the best alternate path.
4. The number of mis-predictions that occur due to stale state information, when large epoch durations are used.

Overhead Metrics. We measure overheads in two different ways:(i) the ping overhead, and (ii) the state exchange overhead, computed in terms of the bits introduced into the network per second. For really huge overlay networks the computational cycles required to calculate a alternate path will also become significant but are not covered here.

4 Data Collection

Our experiments were performed on the PlanetLab testbed [4]. PlanetLab is an open, globally distributed testbed for developing, deploying and accessing planetary-scale network services. The PlanetLab testbed consists of sites on both commercial ISPs and the Internet2 network. We selected 36 nodes all over the North American continent. We did not select sites outside the North American continent as many alternate path to these sites would, in most case, share the same transoceanic link and would have provided less chance for performance gain. For 100% connectivity a total of up to ($36 \times 35 = 1260$) paths are monitored.

To analyze the gains we collected five datasets as described in Table 1. Each data set was then analyzed offline with 100%, 75%, 50%, 25% and 12% connectivity. The neighbors of each node were selected at random. The dataset was also used to estimate the effect of increasing epoch duration. Analysis were done on the first dataset to study the effect of reduced number of nodes in the overlay. Performance for 9, 18 and 27 randomly selected nodes was studied.

We also studied the overhead of routing with the flooding protocol implemented in our system (number of bytes received and send by each node), as a function of logical connectivity. Table 2 describes this data.

Table 1. Characteristics of the Ping datasets used in this paper.

dataset	Dates	Duration	No.of nodes
D1	2-3 Feb,04	24 Hours	36
D2	3-4 Feb,04	24 Hours	36
D3	5-6 Feb,04	24 Hours	36
D4	6-7 Feb,04	24 Hours	36
D5	7-9 Feb,04	48 Hours	35

Table 2. Dataset for the state exchange overhead information

dataset	Dates	Connectivity	No.of nodes
D61	17-18 Aug,03	100%	36
D7	8-9 Sep,03	50%	34
D8	3-4 Sep,03	25%	36
D9	10-11 Sep,03	12%	34
D10	11-12 Sep,03	6%	34

5 Planetlab Results

5.1 Effect on Gain

Here we present the effect of varying parameters on the gains achieved by the overlay network.

Connectivity. We analyzed the dataset for different degree of average node connectivity. Connectivity is the total number of nodes about which a node has information. It could have information about a remote node because it pinged it or because it has been pinged by the remote node and received the information from the remote node. Thus to achieve 100% connectivity a node need to ping only 50% of the nodes as long as all the remaining 50% of the nodes ping it.

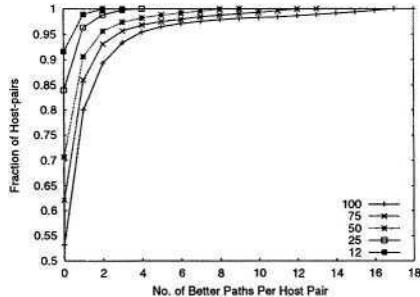


Fig. 1. Number of Better alternate paths for Latency

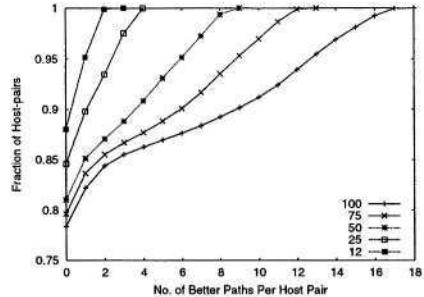


Fig. 2. Number of Better alternate paths for Loss

Figure 1. shows the number of better alternate paths per host-pair for latency. We can see that reducing the connectivity reduces the number of host-pairs for which better paths can be found, however, there are still significant number of host-pairs with better alternate paths. For e.g. for 100% connectivity 53% of host-pairs had no better path i.e. 47% of the paths had at least one better path. Reducing the connectivity to 75% increased the number of host-pairs with no better paths to 61% i.e. still over 39% of the paths had a better alternate path. Here decrease in the number of host-pairs having better alternate paths is around 17%. It is also seen that in many cases there exists more then one better alternate paths. This suggests that even after removing a few nodes from the neighborhood of a node it should be able to find at least one better path.

Figure 2. shows a similar plot for loss. Reducing connectivity to 75% still allows us to find a better path for 20.3% of the host-pairs against 22% at 100% connectivity. The decrease in the number of host-pairs with better alternate paths is only around 2%.

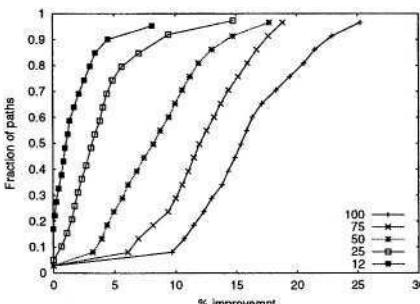


Fig. 3. % Improvement for Latency

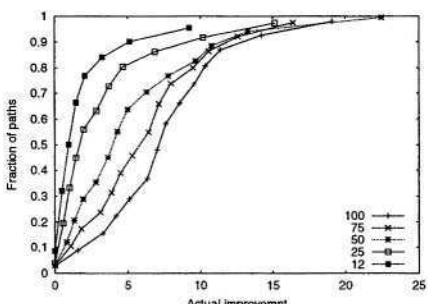


Fig. 4. Actual Improvement for loss

Figure 3. and 4. shows the improvement that is achieved for latency and loss respectively. For latency percentage improvement is plotted. However, for loss in many cases the percentage improvement is 100% (loss changing from a high value to zero) and hence the actual improvement is plotted. When connectivity is reduced from 100% to 75% the median of the improvement achieved for latency goes down from 15% to around 12% and for loss it goes down from 7% to 6% .

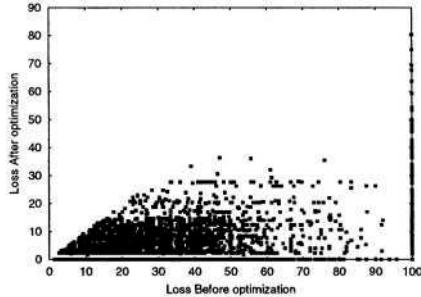


Fig. 5. Actual Improvement for loss

For Protocols like TCP an improvement of 3% in the loss is more significant when the change is from 3.5 to 0.5% rather then 93 to 90%. In Fig 5. we see that in most of the cases the improvement in loss is significant in these terms too i.e. the loss on the best path is very close to zero. We also observed that this type of improvement is independent of the connectivity or epoch duration for the overlay.

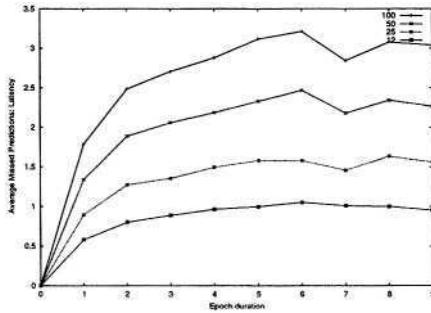


Fig. 6. Number of Mis-prediction as a fraction of total predictions for Latency

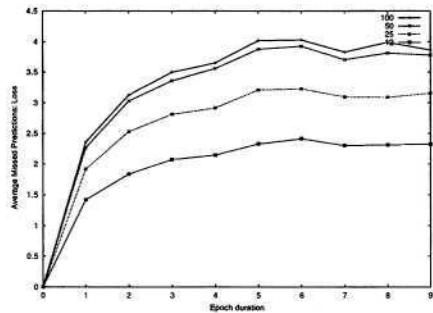


Fig. 7. Number of Mis-prediction as a fraction of total predictions for Loss

Epoch duration. Figure 6. shows the number of erroneous best path predictions for latency resulting from an increase in the epoch duration keeping the connectivity constant. The fraction of mis-predictions with a larger epoch is

computed relative to the epoch duration used in the measurements. The X-axis of the graph indicates the epoch duration and the Y-axis shows the number of mis-predictions as a fraction of the total better alternate path computed. By just doubling the epoch duration the number of mis-predictions may go up by around 10% at 100% connectivity. Varying connectivity from 100% to 75% increase the number of mis-predictions by around 2%.

Figure 7. shows a similar graph for the number of erroneous predictions for loss. Doubling the epoch duration results in around 30% mis-predictions at 100% connectivity. Here the number of mis-predictions seems to be less affected by connectivity.

5.2 Overlay Size, Number of Hops, and Service Metrics

In this section we discuss the effect of the number of nodes in the overlay, number of intermediate hops and service metrics.

It is seen that increasing the overlay size aids in identifying more number of better path for both loss and latency. This is expected, as we increase the number of nodes the number of paths scanned for a better path increases and hence the probability of finding a better path increases. The amount of improvement that can be achieved also increases. We find that increasing the overlay size by 33% increases the number of better path identified by around 5 to 10% and the improvement on these paths by around 5%. The number of mis-predictions does not seem to be affected by the scale of the network. This however would increase the overheads by about 60% for 100% connectivity.

We studied the effect of using multiple nodes to find a better alternate path for a given host-pair (i.e. instead of going through a single intermediate node we use 2 or more intermediate nodes). We found that multiple hops do not contribute significantly in improving the performance seen by a given node.

We have conducted the experiments to identify better path with respect to loss and latency. If application requires better paths for different metric or multiple metrics the performance gain may differ. For example, if we need to find better path for three metrics (loss, latency and jitter) simultaneously, the total number of better alternate paths is just 1%. If we need to find better paths for only loss and jitter about 7 to 8% of the host-pairs where seen to have a better alternate path.

5.3 Overheads

For computing better paths, an overlay needs to collect and distribute the network state. The process of sampling the network and distributing the state causes extra traffic on the network (overheads). There are two types of overhead associated with our method:

- Ping: Ping is used to collect current information about loss and delay on the network. Each node is pinged for maximum of six seconds with the amortized rate of pinging being 6 packets per second. In our experimental

set up with 100% connectivity and 36 nodes, ping introduced a traffic of 3Kbps during the active period (when we are sampling the network). Now the actual epoch length consists of the active periods and the passive periods (when no sampling is done) hence the total network traffic generated over a epoch is the number of bits sent divided by the epoch length and would be even smaller then 3 Kbps.

- **State Exchange:** Once the nodes collect information regarding other nodes it is connected to, we use a link state protocol to transmit this information to all nodes. The overhead in this is the numbers of bytes that have to be sent and received by each node.

We analyze the effect of varying parameters on both types of overheads

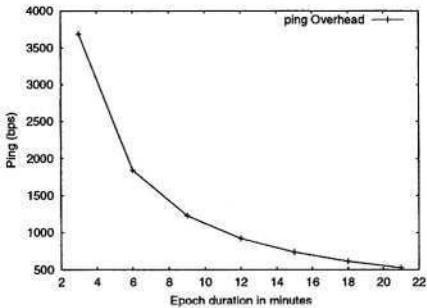


Fig. 8. Ping Overheads

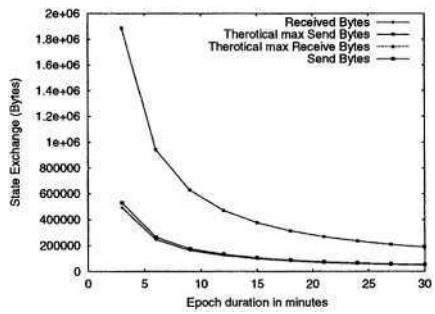


Fig. 9. State exchange overhead

Ping. Figure 8 shows the effect of increasing epoch duration on ping overhead. As can be seen as the epoch duration increases the average amount of bytes sent in the network for active measurements decreases. To estimate whether an alternate path is better then a direct path we need to have information regarding the direct path between all host-pairs. Hence even for reduced connectivity we ping all the nodes. Connectivity only affects the amount of data exchanged and not the ping overhead.

Figure 9. shows the effect of epoch duration on the number of bytes sent and received by Kupl1.ittc.ku.edu with all other nodes. The theoretical maximum amount of bytes exchanged corresponds to the amount of bytes that would have been exchanged if we exchange the complete information gathered at any epoch. However in our setup we exchange only the information that has changed since the last epoch and hence the total amount of information exchanged is very low. Changing epoch duration obviously reduces the amount of data transmitted. Reducing the connectivity also affect the amount of data exchanged. Reducing connectivity from 100% to 50% reduces the amount of data transmitted by almost a factor of 4 from 11kbps to 3kbps.

Ping as well as the state exchange overhead increases with the increase in the overlay size. Doubling the overlay size increases the amount of overhead bytes by about 4 times.

6 Future Work and Conclusion

This paper presents an overview of the tradeoff between performance gain in an overlay network with the overheads incurred. Studying these we can tune the overlay to achieve the required performance while controlling the overheads.

Based on the analysis using actual network measurements we conclude that alternate paths help improve loss and delay over a network. Moreover, connectivity influences these improvements. Reducing connectivity by half reduces the overhead by four times at the cost of reducing the number of better alternate paths by almost 40% for latency and over 30% for loss. Similarly increasing the epoch duration by two reduces the overheads by two. However, this may lead to mis-predictions in computing the best paths for loss in almost 30% of the cases, and for latency in 10% of the cases. As the epoch duration increases the number of mis-predictions increases but soon stabilizes, however the maximum number of mis-prediction may be quite high in some cases. Thus by reducing the connectivity by half and increasing the epoch duration by four we can achieve a network 8 times the current size with the same amount of overhead but some lost performance.

There are several issues that warrant further investigation. We want to investigate the possibility of using passive sampling to gather network conditions instead of active sampling of the network. The effect of adding nodes outside the North American continent also needs to be studied. Also we need to calculate the computational overhead incurred for path calculation at any node.

References

1. D. Andersen, A. Snoeren, and H.Balakrishna: Best-Path vs. Multi-Path Overlay Routing. Proc. of the ACM SIGCOMM Internet Measurement Conference. Miami, FL, October 2003.
2. D. Andersen, H.Balakrishna, F Kaashoek, and R. Morris: Resilient Overlay Networks. Proc. of the 18th ACM Symposium on Operating System Principles, October 2001.
3. S. Savage, A. Collins. E. Hoffman, J. Snell and T. Anderson: The End-to-end Effects of Internet Path Selection. Proc. of ACM SIGCOMM, September 1999.
4. Peterson, L., Anderson, T., Culler, D., Roscoe, T : A Blueprint for Introducing Disruptive Technology into the Internet. Proc. of ACM HOTNET, Oct. 2002
5. Savage,S.,Anderson,T., Et Al., Detour:A Case for Informed Internet Routing and Transport. IEEE Micro 19, 1 (Jan. 1999), 50-59.

Toward a Measurement-Based Geographic Location Service

Artur Ziviani^{1,2}, Serge Fdida¹, José F. de Rezende², and
Otto Carlos M.B. Duarte²

¹ Laboratoire d’Informatique de Paris 6 (LIP6)
Université Pierre et Marie Curie (Paris 6)
Paris, France
{Artur.Ziviani,Serge.Fdida}@lip6.fr

² Grupo de Teleinformática e Automação (GTA)
COPPE/Poli – Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
{rezende,otto}@gta.ufrj.br

Abstract. Location-aware applications require a geographic location service of Internet hosts. We focus on a measurement-based service for the geographic location of Internet hosts. Host locations are inferred by comparing delay patterns of geographically distributed landmarks, which are hosts with a known geographic location, with the delay pattern of the target host to be located. Results show a significant correlation between geographic distance and network delay that can be exploited for a coarse-grained geographic location of Internet hosts.

1 Introduction

Location-aware applications take into account where the users are accessing from and thereby can offer novel functionalities in the Internet. Examples of these novel location-aware applications are: targeted advertising on web pages, automatic selection of a language to first display the content, accounting the incoming users based on their positions, restricted content delivery following regional policies, and authorization of transactions only when performed from pre-established locations. In peer-to-peer networks, location-aware construction of overlay networks can avoid unnecessary high latency hops, thus improving routing performance [1]. Multimedia delivery systems, such as Content Distribution Networks (CDNs), can also benefit from knowing the location of their clients [2]. For example, benefits include the indication of close servers to clients or the adaptation of multimedia content based on the location of clients. Lakhina *et al.* [3] investigate the geographic location of Internet components to create a base for the development of geographically-driven topology generation methods. In the current Internet, however, there is no direct relationship between a host identification and its physical location. The novel location-aware applications then require the deployment of a geographic location service for Internet hosts.

We focus on a measurement-based geographic location service of Internet hosts to support location-aware applications. We build upon GeoPing [4] that adopts an empirical approach based on the observation that hosts sharing similar delays to other fixed hosts tend to be near each other geographically. In a previous work [5], we have evaluated different similarity models to compare the delay patterns gathered from different reference hosts. In this paper, we carry out live experiments to evaluate the correlation between geographic distance and network delay as well as the achieved distance accuracy. Our findings indicate that contrary to conventional wisdom there is a significant level of correlation between distance and delay. This correlation becomes stronger as connectivity within the network becomes richer. Moreover, such a correlation can be exploited to perform a coarse-grained geographic location of Internet hosts.

This paper is organized as follows. Section 2 briefly reviews schemes to determine the geographic location of Internet hosts. In Section 3, we formalize the measurement-based geographic location of Internet hosts. We introduce the adopted measures of similarity in Section 4. Section 5 presents our experiments and results. Finally, in Section 6 we conclude and discuss future work.

2 Related Work

A DNS-based approach to provide a geographic location service of Internet hosts is proposed in RFC 1876 [6]. Nevertheless, the adoption of the DNS-based approach is restricted since it requires changes in the DNS records and administrators have no motivation to register new location records. Tools such as IP2LL [7] and NetGeo.[8] query Whois databases in order to obtain the location information recorded therein to infer the geographic location of a host. Such an information, however, may be inaccurate and stale. Moreover, if a large and geographically dispersed block of IP addresses is allocated to a single entity, the Whois databases may contain just a single entry for the entire block. As a consequence, a query onto the Whois databases provides the registered location of the entity that controls the block of IP addresses, although the concerned hosts may be geographically dispersed.

Padmanabhan and Subramanian [4] investigate three important techniques to infer the geographic location of an Internet host. The first technique infers the location of a host based on the DNS name of the host or another nearby node. This technique is the base of GeoTrack [4], VisualRoute [9], and GTrace [10]. Quite often network operators assign names to routers that have some geographic meaning, presumably for administrative convenience. For example, the name `bcr1-so-2-0-0.Paris.cw.net` indicates a router located in Paris, France. Nevertheless, not all names contain an indication of location. Since there is no standard, operators commonly develop their own rules for naming their routers even if the names are geographically meaningful. Therefore, the parsing rules to recognize a location from a node name must be specific to each operator. The creation and management of such rules is a challenging task as there is no standard to follow. As the position of the last recognizable router in the path toward

the host to be located is used to estimate the position of such a host, a lack of accuracy is also expected. The second technique splits the IP address space into clusters such that all hosts with an IP address within a cluster are likely to be co-located. Knowing the location of some hosts in the cluster and assuming they are in agreement, the technique infers the location of the entire cluster. An example of such a technique is GeoCluster [4]. This technique, however, relies on information that is partial and possibly inaccurate. The information is partial because it comprises location information for a relatively small subset of the IP address space. Moreover, such an information may be inaccurate because the databases rely on data provided by users, which may be unreliable to provide correct location information. The third technique is based on delay measurements and the exploitation of a possible correlation between geographic distance and network delay. Such a technique is the base of GeoPing [4]. The location estimation of a host is based on the assumption that hosts with similar network delays to some fixed probe machines tend to be located near each other. Therefore, given a set of landmarks with a well known geographic location, the location estimation for a target host to be located is the location of the landmark presenting the most similar delay pattern to the one observed for the target host.

3 Measurement-Based Geographic Location Service

We formalize the problem of inferring a host location from delay measurements as follows. Consider a set $\mathcal{L} = \{L_1, L_2, \dots, L_K\}$ of K landmarks. Landmarks are reference hosts with a well known geographic location. Consider a set $\mathcal{P} = \{P_1, P_2, \dots, P_N\}$ of N probe machines. Fig. 1 illustrates the steps in inferring a host location from delay measurements, which are detailed along this section. The probe machines periodically determine the network delay, which is actually the minimum RTT of several measurements, to each landmark (Fig. 1(a)). Therefore, each probe machine $P_x \in \mathcal{P}$ keeps a delay vector $\mathbf{d}_x = [d_{1x}, d_{2x}, \dots, d_{Kx}]^T$, where d_{ix} is the delay between the probe machine P_x and the landmark $L_i \in \mathcal{L}$. Suppose one wants to determine the geographic location of a given target host T . A location server that knows the landmark set \mathcal{L} and the probe machine set \mathcal{P} is then contacted. The location server asks the N probe machines to measure the delay to host T (Fig. 1(b)). Each probe machine $P_x \in \mathcal{P}$ returns to the location server a delay vector $\mathbf{d}'_x = [d_{1x}, d_{2x}, \dots, d_{Kx}, d_{Tx}]^T$, i.e., the delay vector \mathbf{d}_x plus the just measured delay to host T (Fig. 1(c)). After receiving the delay vectors from the N probe machines, the location server is able to construct a delay matrix \mathbf{D} with dimensions $(K + 1) \times N$:

$$\mathbf{D} = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1N} \\ d_{21} & d_{22} & \dots & d_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ d_{K1} & d_{K2} & \dots & d_{KN} \\ d_{T1} & d_{T2} & \dots & d_{TN} \end{bmatrix} \quad (1)$$

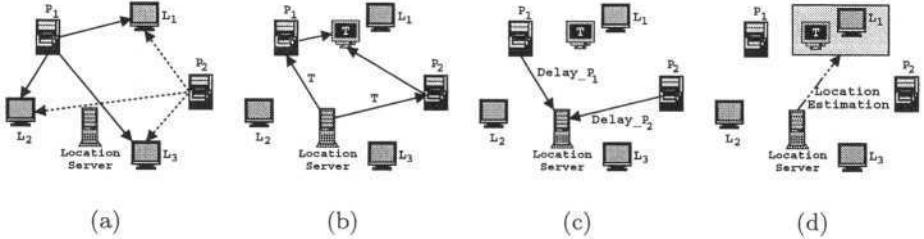


Fig. 1. Inferring a host location from delay measurements.

The delay vectors gathered by the demanding location server from the probe machines correspond to the columns of the delay matrix \mathbf{D} . The location server then compares the lines of the delay matrix \mathbf{D} to estimate the location of host T . The delay matrix \mathbf{D} combined with the knowledge of the location of the landmarks of the set \mathcal{L} compose a delay map recording the relationship between network delay and geographic location.

4 Measuring the Similarity between Delay Patterns

In this section, we investigate how to best measure the similarity between the delay pattern of each landmark and the one observed for the target host. The delay patterns result from the partial viewpoints gathered by the distributed probe machines. The landmark that presents the most similar delay pattern with respect to the one of the target host provides the location estimation of that host. Measuring the similarity of the concerned delay patterns is thus a key point for the accuracy of the host location from delay measurements.

The function $\mathcal{S}(\mathbf{x}, \mathbf{y})$ is defined to measure the degree of dissimilarity between two delay patterns \mathbf{x} and \mathbf{y} of size N , where N is the number of adopted probe machines. These delay patterns are gathered by the probe machines from each landmark and from the target host to be located. To formalize the dissimilarity evaluation, we also define a line vector $\mathbf{1}_i$ of size $K + 1$ that has all elements equal to 0, except for the i^{th} element that has a value of 1. The landmark L that provides the location estimation of the target host T is the landmark that gives the minimum dissimilarity

$$S_{\min} = \arg \min_{i=1, \dots, K} \mathcal{S}(\mathbf{1}_i \mathbf{D}, \mathbf{1}_{K+1} \mathbf{D}). \quad (2)$$

We first consider distance-based measures of dissimilarity [11] to compare delay patterns. The generalized form to represent a distance metric is given by

$$\mathcal{S}_\gamma(\mathbf{x}, \mathbf{y}) = \left[\sum_{i=1}^N |x_i - y_i|^\gamma \right]^{\frac{1}{\gamma}}, \quad \gamma > 0. \quad (3)$$

When $\gamma = 1$, we have the Manhattan or city-block distance. In contrast, for $\gamma = 2$, we have the Euclidean distance. It is shown that the Chebyshev distance, i.e. $\gamma = \infty$, can be expressed as

$$\mathcal{S}_\infty(\mathbf{x}, \mathbf{y}) = \lim_{\gamma \rightarrow \infty} \mathcal{S}_\gamma(\mathbf{x}, \mathbf{y}) = \max_i |x_i - y_i|.$$

We also consider the Canberra distance given by

$$\mathcal{S}_{\text{canb}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^N \frac{|x_i - y_i|}{x_i + y_i}. \quad (4)$$

If both x_i and y_i are zero the ratio of the difference to the sum is taken to be zero. The Canberra distance is suitable for variables taking non-negative values and is sensitive to small changes close to zero values [11].

The two delay patterns \mathbf{x} and \mathbf{y} can also be thought of as two vectors in a N -dimensional delay space. The similarity $\mathcal{S}_{\cos}(\mathbf{x}, \mathbf{y})$ between them is measured by computing the cosine of the angle θ between these two vectors. The cosine of the angle θ between the delay vectors \mathbf{x} and \mathbf{y} is computed by

$$\cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \quad (5)$$

where “.” denotes the dot-product of the two vectors and $\|\mathbf{x}\|$ is the Euclidean size of vector $\mathbf{x} \in \mathbb{R}^N$, i.e. $\|\mathbf{x}\| = \sqrt{\sum_{i=1}^N x_i^2}$.

An alternative measure of similarity is to compute the coefficient of correlation between the two delay patterns \mathbf{x} and \mathbf{y} . This correlation-based similarity model is denoted by $\mathcal{S}_{\text{cor}}(\mathbf{x}, \mathbf{y})$. The coefficient of correlation is defined as

$$\text{corr}(\mathbf{x}, \mathbf{y}) = \frac{\sigma_{\mathbf{x}\mathbf{y}}^2}{\sigma_{\mathbf{x}} \sigma_{\mathbf{y}}}, \quad (6)$$

where $\sigma_{\mathbf{x}\mathbf{y}}^2$ denotes the covariance between delay patterns \mathbf{x} and \mathbf{y} , and $\sigma_{\mathbf{x}}$ is the standard deviation of \mathbf{x} .

5 Experimental Results

In this paper, we analyze the results of live experiments to evaluate basic properties of a measurement-based geographic location service of Internet hosts.

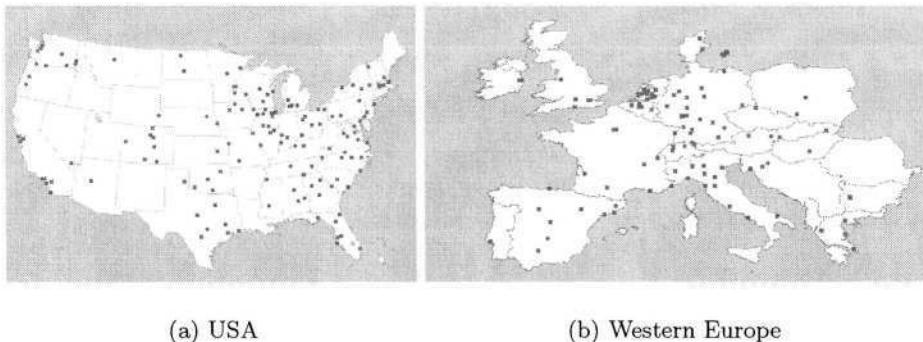


Fig. 2. Geographic location of landmarks.

5.1 Experimental Setup

For the experiments, we use 9 measurement boxes from the NIMI (National Internet Measurement Infrastructure) project [12] as our probe machines. They are geographically distributed as follows: 5 in Western Europe, 3 in the U.S., and 1 in Japan. Recent works [4,13] indicate that 7 to 9 dimensions provide sufficient network distance representation. The experimental set of landmarks comes from two datasets:

- LibWeb – this set of hosts is mainly composed of university sites extracted from library web (LibWeb) servers around the world [14].
 - RIPE – these hosts are part of the Test Traffic Measurements (TTM) project of the RIPE network [15]. All hosts on the RIPE network are equipped with a GPS card, thus allowing their exact geographic position to be known.

The resulting experimental dataset totals to 397 landmarks that are sparsely distributed worldwide. The geographic distribution of these landmarks is as follows: 199 in North America (U.S. and Canada), 156 in Western Europe, 19 in Eastern Europe, 13 in Latin America, 9 in Asia and Oceania, and 1 in the Middle East. This distribution is intended to at least roughly reflect the distribution of users (hosts) to be located. In a previous work [16], we propose the demographic placement of landmarks to better represent the location of users (hosts). It should be noted that landmarks are unsuspecting participants in the procedure since a landmark may be any host, with a known geographic location, able to echo ping messages. Figure 2 shows the geographic location of the landmarks in the U.S. and in Western Europe, which are regions likely to have rich connectivity and host most users to be located.

The probe machines measure the delay toward the set of landmarks. The delay metric is actually the minimum of several RTT measurements to avoid taking into account congestion delays. The measurements toward each landmark from the different probe machines are enough spaced to avoid triggering detection systems against DDoS (Distributed Denial of Service) attacks.

5.2 Correlation between Geographic Distance and Network Delay

In this section, we evaluate the correlation between geographic distance and network delay. Until recently, common sense claimed that there is a weak correlation between distance and delay within the network. Claffy [17] mentions this as one of some myths about Internet data. Our results show that few landmarks present very large delays, probably due to poor connectivity. To avoid taking into account these outliers in our evaluation, we consider data within the 98th, 95th, and 90th percentiles of the measured network delay. The observed correlation between geographic distance and network delay is moderate to strong in these cases, resulting in $R=0.6229$, $R=0.8093$, and $R=0.8767$, respectively. Figures 3(a), 3(c), and 3(e) present the corresponding scatter plots for these results that cover landmarks located worldwide.

We also observe that poor connectivity weakens the correlation between geographic distance and network delay. We then identify the landmarks located in North America and Western Europe. These regions are likely to offer the richest connectivity linking their hosts. We observe an even stronger correlation on these well connected regions, indicating that the correlation becomes stronger as connectivity becomes richer. The coefficients of correlation for the data within the 98th, 95th, and 90th percentiles in these well connected regions are $R=0.7126$, $R=0.8595$, and $R=0.8958$, respectively. The corresponding scatter plots for these results are shown in Figures 3(b), 3(d), and 3(f) for landmarks located in North America and Western Europe (NA-WE). Recent findings [3,18] indicate a strong correlation between population and router density in economically developed countries. Moreover, most users, and consequently most hosts to be located, are likely to be in these regions with richer connectivity, whereby a stronger correlation between distance and delay holds.

5.3 Distance Accuracy

In this section, we consider the whole set of landmarks distributed worldwide, in a total of 397 landmarks. To evaluate the distance accuracy, we take one landmark as a target host and use the remaining landmarks to infer a location estimation for this target. The distance accuracy is measured by the error distance from the location estimation to the location of the target host. We apply the different measures of dissimilarity presented in Section 3 to compare the delay patterns gathered by the probe machines from the landmarks.

Figure 4 shows the probability density function (pdf) of the error distance worldwide for the different measures of dissimilarity. The Canberra distance performs slightly better than the others, providing smaller error distances to more hosts. This distance measure is known to be suitable for non-negative values, such as network delay, and more sensitive to values near zero. This favors a more accurate location of some hosts in comparison with the other measures of dissimilarity since eight out of the nine probe machines are in the U.S. or in Western Europe. For the Canberra distance, we observe that the median value of the error distance is 314 km with a kurtosis of 40.79, showing that the observed distribution of the error distance is heavy-tailed. This is because, for some target

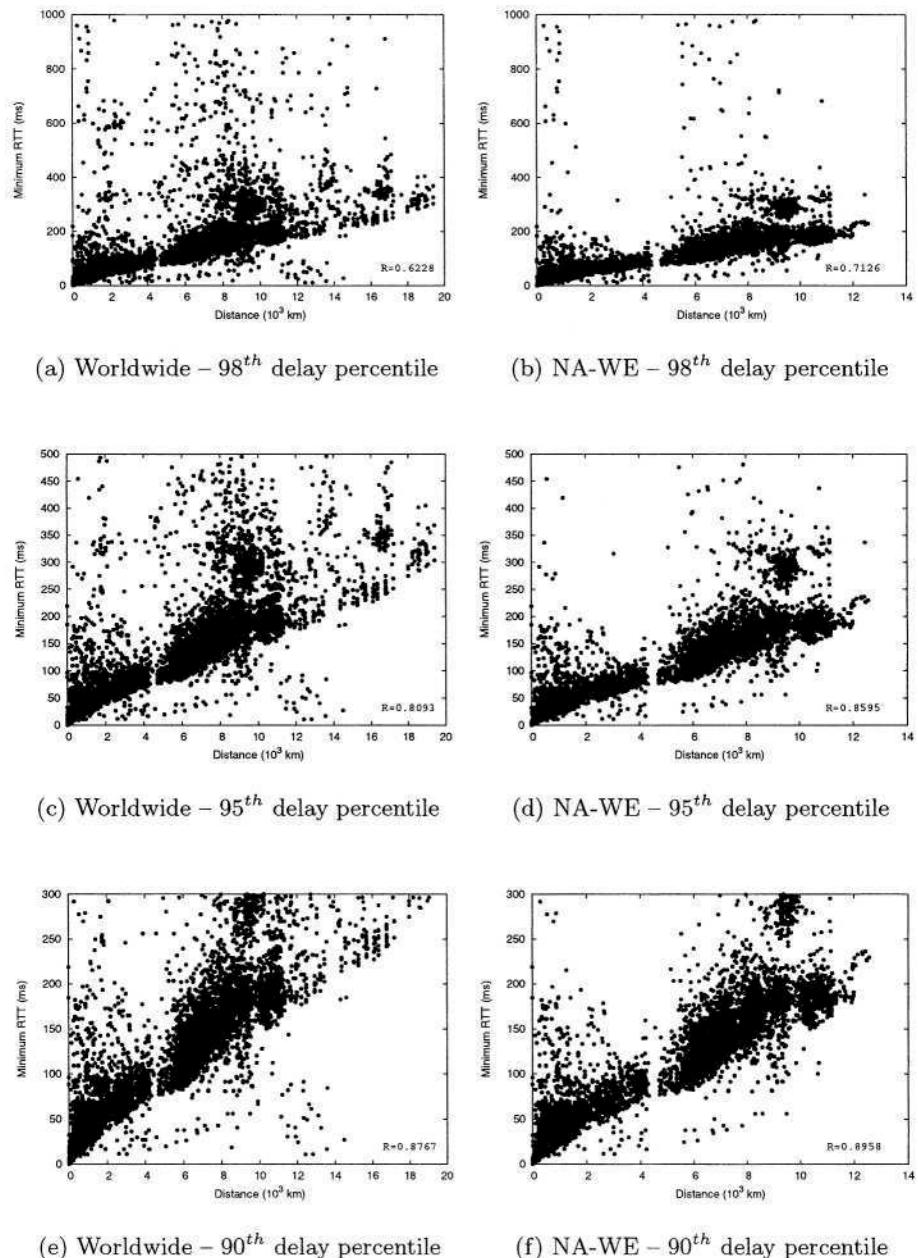


Fig. 3. Correlation between geographic distance and network delay, both worldwide and within the North America and Western Europe (NA-WE) regions.

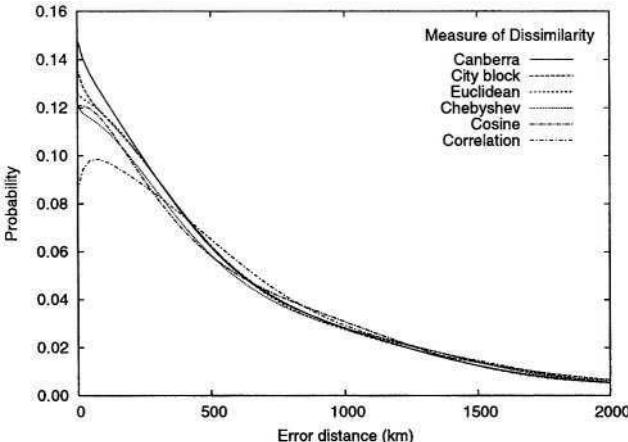


Fig. 4. PDF of the error distance.

hosts, even if the elected landmark is the geographically closest landmark to the target, not necessarily it is nearby the target host. These results indicate that delay measurements can indeed be exploited to determine the geographic location of Internet hosts, although at a coarse granularity.

6 Conclusion

This paper investigates some key properties toward a measurement-based geographic location service of Internet hosts. Such a service can be viewed as an underlying infrastructure for the deployment of novel location-aware applications in the Internet. Live experiments have been carried out to evaluate the correlation between geographic distance and network delay as well as the achieved distance accuracy for different measures of dissimilarity. Our findings indicate that contrary to conventional wisdom there is a significant correlation between geographic distance and network delay. We show that this correlation can be exploited to provide a coarse-grained geographic location of Internet hosts. The location estimation of a host is the location of the landmark presenting the most similar delay pattern with respect to the target host. This poses a fundamental limit: the system has a discrete space of answers since the number of possible answers correspond to the number of landmarks adopted.

As future work, we intend to investigate methods to adopt a continuous space of answers instead of a discrete one. Recent works [13,19,20] propose to infer network proximity without direct measurements by embedding network distances such as delay into a coordinate system of reduced dimensions. Similar concepts can be applied to the measurement-based geographic location of Internet hosts to provide more accurate estimations using fewer measurements.

Acknowledgment. This work is supported by CAPES/COFECUB, FUJB, and CNPq. Authors are thankful to Andrew Adams (PSC) and Vern Paxson (ICIR) for the access to the NIMI network.

References

1. Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Topologically-aware overlay construction and server selection. In: Proc. of the IEEE INFOCOM'2002, New York, NY, USA (2002)
2. Sripanidkulchai, K., Maggs, B., Zhang, H.: Efficient content location using interest-based locality in peer-to-peer systems. In: Proc. of the IEEE INFOCOM'2003, San Francisco, CA, USA (2003)
3. Lakhina, A., Byers, J.W., Crovella, M., Matta, I.: On the geographic location of Internet resources. *IEEE Journal on Selected Areas in Communications* **21** (2003) 934–948
4. Padmanabhan, V.N., Subraraanian, L.: An investigation of geographic mapping techniques for Internet hosts. In: Proc. of the ACM SIGCOMM'2001, San Diego, CA, USA (2001)
5. Ziviani, A., Fdida, S., de Rezende, J.F., Duarte, O.C.M.B.: Similarity models for Internet host location. In: Proc. of the IEEE International Conference on Networks - ICON'2003, Sydney, Australia (2003) 81–86
6. Davis, C., Vixie, P., Goowin, T., Dickinson, I.: A means for expressing location information in the domain name system. *Internet RFC 1876* (1996)
7. University of Illinois at Urbana-Champaign: (IP Address to Latitude/Longitude) <http://cello.cs.uiuc.edu/cgi-bin/slamm/ip2ll/>.
8. Moore, D., Periakaruppan, R., Donohoe, J., Claffy, K.: Where in the world is netgeo.caida.org? In: Proc. of the INET'2000, Yokohama, Japan (2000)
9. Visualware Inc.: (VisualRoute) <http://www.visualware.com/visualroute/>.
10. CAIDA: (GTrace) <http://www.caida.org/tools/visualization/gtrace/>.
11. Gordon, A.D.: Classification: Methods for the Exploratory Analysis of Multivariate Data. Chapman and Hall (1981)
12. Paxson, V., Mahdavi, J., Adams, A., Mathis, M.: An architecture for large-scale Internet measurement. *IEEE Communications Magazine* **36** (1998) 48–54
13. Tang, L., Crovella, M.: Virtual landmarks for the Internet. In: ACM Internet Measurement Conference 2003, Miami, FL, USA (2003)
14. (LibWeb) <http://sunsite.berkeley.edu/Libweb>.
15. (RIPE Test Traffic Measurements) <http://www.ripe.net/ttm/>.
16. Ziviani, A., Fdida, S., de Rezende, J.F., Duarte, O.C.M.B.: Demographic placement for Internet host location. In: Proc. of the IEEE Global Communications Conference - GLOBECOM'2003, San Francisco, CA, USA (2003)
17. Claffy, K.: Internet measurement: myths about Internet data. Talk at NANOG24 Meeting (2002) <http://www.caida.org/outreach/presentations/Myths2002/>.
18. Yook, S.H., Jeong, H., Barabási, A.L.: Modeling the Internet's large-scale topology. *Proc. of the National Academy of Sciences (PNAS)* **99** (2002) 13382–13386
19. Ng, T.S.E., Zhang, H.: Predicting Internet network distance with coordinates-based approaches. In: Proc. of the IEEE INFOCOM'2002, New York, NY, USA (2002)
20. Lim, H., Hou, J.C., Choi, C.H.: Constructing Internet coordinate system based on delay measurement. In: ACM Internet Measurement Conference 2003, Miami, FL, USA (2003)

An Incremental Super-linear Preferential Internet Topology Model

Extended Abstract

Sagy Bar¹, Mira Gonen², and Avishai Wool³

¹ School of Electrical Engineering, Tel Aviv University, Ramat Aviv 69978, ISRAEL.
sagyb@eng.tau.ac.il.

² Dept. Computer Science, Tel Aviv University, Ramat Aviv 69978, ISRAEL.
gonenmir@tau.ac.il.

³ School of Electrical Engineering, Tel Aviv University, Ramat Aviv 69978, ISRAEL.
yash@acm.org.

Abstract. By now it is well known that the distribution of node degrees in the graph induced by the peering arrangements between Autonomous Systems (ASs) exhibits power laws. The most appealing mathematical model that attempts to explain the power-law degree distribution was suggested by Barabási and Albert (the BA model). We introduce two new models that are extensions to the BA model: the “Incremental Edge Addition” (InEd) model, and the “Super-Linear Preferential Attachment” (SLiP) model. We prove that both our models are more successful in matching the power-law exponent, in producing leaves , and in producing a large dense core. Beyond mathematical analysis, we have also implemented our models as a synthetic network generator we call TANG (Tel Aviv Network Generator). Experimentation with TANG shows that the networks it produces are more realistic than those generated by other network generators.

1 Introduction

1.1 Background and Motivation

The connectivity of the Internet crucially depends on the relationships between thousands of Autonomous Systems (ASes) that exchange routing information using the Border Gateway Protocol (BGP). These relationships can be modeled as a graph, called the AS-graph, in which the vertices model the ASes, and the edges model the peering arrangements between the ASes.

Significant progress has been made in the study of the AS-graph’s topology over the last few years. In particular, it is now known that the distribution of vertex degrees (i.e., the number of peers that an AS has) is heavy-tailed and obeys so-called power-laws [SFFF03]: The fraction of vertices with degree k is proportional to $k^{-\gamma}$ for some fixed constant γ . This phenomenon cannot be explained by traditional random network models such as the Erdős-Renyi model [ER60].

1.2 Related Work

Barabási and Albert [BA99] introduced a very appealing mathematical model to explain the power-law degree distribution (the BA model). The BA model is based on two mechanisms: (i) networks grow incrementally, by the adding new vertices, and (ii) new vertices attach preferentially to vertices that are already well connected. They showed, *analytically*, that these two mechanisms suffice to produce networks that are governed by a power-law.

While the pure BA model [BA99] is extremely elegant, it does not accurately model the Internet's topology in several important aspects:

- The BA model does not produce any leaves (vertices with degree 1), whereas in the real AS-graph some 30% of the vertices are leaves.
- The BA model predicts a power law distribution with a parameter $\gamma = 3$, whereas the real AS-graph has a power law with $\gamma \approx 2.11$. This is actually a significant discrepancy: For instance, the most connected ASes in the AS graph have 500–2500 neighbors, while the BA model predicts maximal degrees which are roughly 10 times smaller on networks with comparable sizes.
- It is known that the Internet has a surprisingly large *dense core* [SARK02], [SW03a]: The AS graph has a core of 43 ASes, with an edge density ϱ^1 of over 70%. However, as recently shown by Sagie and Wool [SW03b], the BA model is fundamentally *unable* to produce synthetic topologies with a dense core larger than $\ell = 6$ with $\varrho(\ell) \geq 70\%$.

These discrepancies, and especially the fact that the pure BA model produces an incorrect power law parameter $\gamma = 3$, were observed before. Barabási and Albert themselves refined their model in [AB00] to allow adding links to existing edges, and to allow rewiring existing links. However, as argued by [CCG⁺02], the idea of link-rewiring seems inappropriate for the AS graph, and [BT02] showed that the rewiring probability needs to be as high as 50% for the [AB00] model to produce values of γ which are closer to reality.

The work closest to ours is that of Bu and Towsley [BT02]. The authors attempted to produce a BA-like model that will (i) produce a more realistic power law parameter γ , while (ii) still remaining amenable to mathematical analysis. The model they suggest meets these goals, however, we claim that it is still not satisfactory. Their model is rather unnatural, and involves a crucial technical parameter that does not correspond to any intuitive feature of the development of the AS graph. The authors themselves admit that the main reason for considering such a counter-intuitive model is that it can be analyzed mathematically—and that other, more natural, extensions, greatly increase the difficulty of analysis.

1.3 Contributions

Our main contribution is a new extension to the BA model, that has the following features:

¹ The density $\varrho(\ell)$ of a subgraph with ℓ vertices is the fraction of the $\ell(\ell-1)/2$ possible edges that exist in the subgraph.

- It addresses the discrepancies of the BA model with respect to (i) the lack of leaves, (ii) value of the power law parameter γ , and (iii) the lack of a dense core.
- It is natural and intuitive, and follows documented and well understood phenomena of the Internet’s growth.
- We are able to analyze our model, and rigorously prove many of its properties.

Our model hinges on two ideas, which we call “Incremental Edge Addition” (InEd) and “Super-Linear Preferential Attachment” (SLiP)

Beyond mathematical analysis, we also implemented our model as a synthetic network generator we call TANG (Tel Aviv Network Generator). Experimentation with TANG shows that the networks it produces are more realistic than those generated by other network generators such as BRITE [MLMB01], and Inet [WJ02]. TANG is freely available from the authors [Woo04].

Organization: In the next section we give an overview of the BA model. In Section 3 we introduce the Incremental Edge Addition (InEd) model. Section 4 presents the Super-Linear Preferential Attachment (SLiP) model. In Section 5 we analyze the expected number of leaves in a model combined from the InEd model and the SLiP model. Section 6 describes TANG and the results of our simulations. We conclude with Section 7.

We omit most of the proofs in this space-limited extended abstract.

2 Overview of the BA Model

The BA model works as follows, (i) Start with a small number (m_0) of arbitrarily connected vertices, (ii) At every time step, add a new vertex with $m(\leq m_0)$ edges that connect the new vertex to m different vertices already present in the system. (iii) The new vertex picks its m neighbors randomly, where an existing vertex i , with degree k_i , is chosen with probability $p(k_i) = k_i / \sum_j k_j$.

Since every time step introduces 1 vertex and m edges, it is clear that the average degree of the resulting network is $\approx 2m$.

Observe that new edges are added in *batches* of m . This is the reason why the pure BA model never produces leaves, [SW03a], and the basis for the model’s inability to produce a dense core. Furthermore, empirical evidence [CCG⁺02], shows that the vast majority of new ASes are born with a degree of 1, and not 2 or 3 (which is necessary to reach the AS graph’s average degree of ≈ 4.33).

3 The Incremental Edge Addition (InEd) Model

Our first model modifies the way in which edges are introduced into the BA model. In this section we give the model’s definition, analyze its degree distribution and prove that it is close to a power-law distribution. We also analyze the expected number of leaves.

3.1 Model Definition

The basic setup in the InEd model is the same as in the BA model: We start with m_0 nodes. At each time step we add a new node, and m edges. However, the edges are added in the following way: one edge connects the new node to nodes that are already present. An existing vertex i , with degree k_i , is chosen with probability $p(k_i) = k_i / \sum_j k_j$. (That is, $p(k_i)$ is linear in k_i , as in the BA model). The remaining $m - 1$ edges connect *existing* nodes. One endpoint of each edge is uniformly chosen, and the other endpoint is connected preferentially, choosing a node i with probability $p(k_i)$ as defined above.

Note that this is reminiscent of the [AB00] model. In that model nodes are all added with degree m , and additionally, nodes that are chosen uniformly at random grow more edges with some fixed probability p . In our model, all nodes start with degree 1, as found empirically by [CCG⁺02]. Moreover, we avoid the extra parameter p .

Our analysis shows that the InEd model produces a remarkably accurate number of leaves, and a power-law degree distribution, albeit with a parameter γ which is still too high. The predicted maximal degree improves as well: it is about twice that predicted by the BA model.

3.2 Power Law Analysis

We show that the InEd model produces a near-power-law degree distribution. We analyze our model using the “mean field” methods in Barabási-Albert [BA99]. As in [BA99], we assume that k_i changes in a continuous manner, so k_i can be interpreted as the average degree of node i , and the probability $p(k_i)$ can be interpreted as the rate at which k_i changes.

Theorem 3.1. *In the InEd model, $\Pr[k_i(t) = k] \propto (k + 2m - 2)^{-3}$.*

We prove the theorem using the following lemma.

Lemma 3.1. *Let t_i be the time at which node i was added to the system. Then $k_i(t) = (2m - 1)\sqrt{\frac{t}{t_i}} - 2(m - 1)$.*

Proof: At time t the sum of degrees is $2mt$. The change in an existing node’s degree is influenced by the probability of it being chosen preferentially, and by the probability that it is selected uniformly. Thus we get the following differential equation:

$$\frac{\partial k_i}{\partial t} = m \cdot \frac{k_i}{2mt} + \frac{m - 1}{t} = \frac{k_i}{2t} + \frac{m - 1}{t}.$$

The initial condition for node i is $k_i(t_i) = 1$. Solving for $k_i(t)$ proves the Lemma. ■

Corollary 3.1. *The expected maximal degree in the InEd model is*

$$(2m - 1)(\sqrt{t} - 1) + 1.$$

Proof: By setting $t_i = 1$ in Lemma 3.1 we get the result. ■

Proof of Theorem 3.1: Using Lemma 3.1 the probability that a node has a degree $k_i(t)$ smaller than k , $\Pr[k_i(t) < k]$, can be written as

$$\begin{aligned}\Pr[k_i(t) < k] &= \Pr\left[(2m - 1)\sqrt{\frac{t}{t_i}} - 2(m - 1) < k\right] = \Pr\left[t_i > \left(\frac{2m - 1}{k + 2m - 2}\right)^2 t\right] \\ &= 1 - \Pr\left[t_i \leq \left(\frac{2m - 1}{k + 2m - 2}\right)^2 t\right] = 1 - \left(\frac{2m - 1}{k + 2m - 2}\right)^2 \frac{t}{t + m_0}\end{aligned}$$

Thus

$$\Pr[k_i(t) = k] = \frac{\partial}{\partial k} \left[1 - \left(\frac{2m - 1}{k + 2m - 2}\right)^2 \frac{t}{t + m_0} \right] \propto (k + 2m - 2)^{-3} \quad ■$$

Theorem 3.1 shows that the InEd model produces a near-power-law distribution, but the coefficient γ is still ≈ 3 .

3.3 Analysis of the Expected Number of Leaves

The pure BA model is unable to produce any leaves: each new node has degree m . In contrast, the InEd model produces a realistic number of leaves. Note that nodes in the InEd model start as leaves. We now compute the probability that a node that entered at time t_i will remain a leaf at time n , and compute the expected number of leaves in the system at time n .

Let v_i be the node that entered at time t_i , and let $\deg_n(v_i)$ be the degree of v_i after time n .

Theorem 3.2. *In the InEd model, $E[\#\text{leaves}] \leq \frac{n}{m+1/2}$.*

Computer simulations show that this upper bound is very accurate: for $n = 10,000$, $m = 2$, the bound of Theorem 3.2 is 40% leaves, and our simulation show that about 3,995 leaves are generated.

4 The Super-Linear Preferential Attachment (SLiP) Model

In this model, we generalize the BA model in a different way: We assume that the utility of joining a highly-connected node is super-linear in its degree. This assumption agrees with the observations of [CCG⁺02]. As in Section 3, we give the model's definition, analyze its degree distribution and prove that it is close to power-law distribution.

4.1 Model Definition

In the SLiP model, at each time step we add a new node, and m edges, in the following way: All m edges connect the new node to nodes already present in the network (as in the pure BA model). However, an existing node i is chosen as an endpoint with probability

$$p(k_i) = \frac{k_i^{1+\varepsilon}}{\sum_j k_j^{1+\varepsilon}},$$

for some $\varepsilon > 0$. Thus the preferential attachment is super linear. Note that setting $\varepsilon = 0$ gives the pure BA model.

4.2 Power Law Analysis

As in the analysis of the InEd model, we show that the SLiP model produces a near-power-law distribution. As before we assume that k_i changes in a continuous manner, so the probability $p(k_i)$ can be interpreted as the rate at which k_i changes.

A main technical difficulty in the SLiP model is that the denominator $\sum_j k_j^{1+\varepsilon}$ is not fixed. Therefore, we start by bounding $\sum_j k_j^{1+\varepsilon}$.

Lemma 4.1. *For any network over t nodes and mt edges, and any $\varepsilon > 0$,*

$$t(2m)^{1+\varepsilon} \leq \sum_j k_j^{1+\varepsilon} \leq (2mt)^{1+\varepsilon}$$

Corollary 4.1. $\sum_j k_j^{1+\varepsilon} \approx (2m)^{1+\varepsilon} t^{1+\varepsilon/2}$

Lemma 4.2. *In the SLiP model, $k_i(t) = m / \left(1 - \frac{1}{2^\varepsilon t_i^{\varepsilon/2}} + \frac{1}{2^\varepsilon t_i^{\varepsilon/2}}\right)^{1/\varepsilon}$*

Corollary 4.2. *The expected maximal degree in the SLiP model is $\leq 2m\sqrt{t}$*

Corollary 4.2 shows that the SLiP model, on its own, achieves essentially the same (expected) maximal degree that is achieved by the InEd model (recall Corollary 3.1). This maximal degree is about twice higher than that of the pure BA model.

Theorem 4.1. *In the SLiP model $\Pr[k_i(t) < k] = 1 - \left[\frac{(1/t+m_0)^{\varepsilon/2}}{2^\varepsilon + \frac{1}{t^{\varepsilon/2}} - \left(\frac{2m}{k}\right)^\varepsilon}\right]^{2/\varepsilon}$*

Note that the SLiP model does not produce any leaves since nodes are added with degree m .

5 The Combined InEd/SLiP Model

Since the InEd and SLiP models modify the BA model in different ways, we can easily combine them into a single model, which would enjoy the benefits offered by each model. Unfortunately, we are unable to show, analytically, that the combined model has a power-law behavior—the differential equations we obtain are too difficult.

5.1 Analysis of the Expected Number of Leaves

In contrast, we are able to analyze the expected number of leaves in the combined model. Theorem 5.1 shows that the bound of Theorem 3.2 almost holds for the combined model as well, up to a small constant factor.

As in the InEd Model, let v_i be the node that entered at time t_i , and let $\deg_n(v_i)$ be the degree of v_i after time n .

Theorem 5.1. *In the SLiP model, $E[\#\text{leaves}] \leq \frac{n}{m}$.*

6 Implementation

We implemented the combined SLiP/InEd model as a synthetic network generator we call TANG (Tel Aviv Network Generator). TANG accepts the desired number of vertices (n), the average degree (d), and the utility function's exponent ($a = 1 + \varepsilon$), as arguments. The average degree is allowed to be fractional. Setting the exponent to 1 (i.e., $\varepsilon = 0$) causes TANG to use the linear InEd model. TANG is also able to produce pure BA-model networks.

We used TANG to generate synthetic topologies with Internet-like parameters. We used $n = 15,000$ and $d = 4.33$, which match the values reported in [SW03a]. We generated 10 random topologies for each setting of $\varepsilon = 0, 0.1, 0.2, 0.3$, and 10 random topologies for the pure BA model. We compared these networks to the AS-graph snapshot collected by [SW03a].

6.1 Power Law Analysis

Fig. 1 shows the Complementary Cumulative Density Function (CCDF)² of the degree distribution in the Internet's AS-graph and in the TANG-generated synthetic networks. For the synthetic networks, each CCDF curve is the average taken over the 10 randomly generated networks.

The figure clearly shows that the AS graph obeys a power-law degree distribution, with a CCDF exponent of $\eta = 1.17$. The figure also shows the shortcomings of the pure BA model: (a) we can see that $CCDF(2) = CCDF(1) = 1$, which indicates that BA networks do not contain any leaves; and (b) it is clear that slope of the BA model's CCDF is too steep: the power-law exponent is $\eta = 1.96$.

² For any distribution of degrees, $CCDF(k) = \Pr[\deg_n(v) \geq k]$. Note that if $\Pr[\deg_n(v) = k] \propto k^{-\gamma}$ then $CCDF(k) \propto k^{-\eta} = k^{1-\gamma}$.

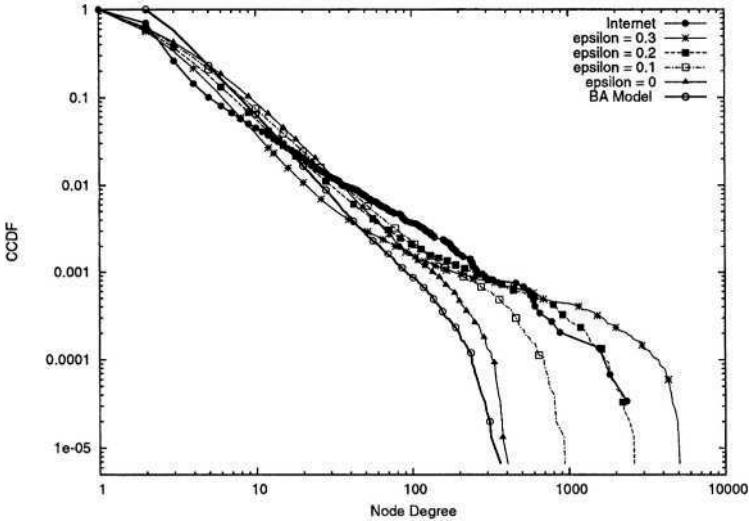


Fig. 1. The CCDF of the degree distribution for the Internet’s AS-graph, the combined SLiP/InEd networks with $\varepsilon = 0, \dots, 0.3$, and the pure BA model (log-log scale).

The figure shows that the InEd model ($\varepsilon = 0$) brings the number of leaves in the network to a fairly realistic level: 37.5% leaves in the InEd model versus 30% in the AS-graph. Note that Theorem 3.2 predicts that when the average degree is 4.33 (i.e., $m = 2.165$) the number of leaves will be $1/(2.165 + 0.5) = 37.52\%$: a very accurate estimate. We can see that the power law produced by the InEd model is slightly better than that of the BA model ($\eta = 1.83$), but still too steep.

The figure shows that the SLiP model shifts the CCDF curve closer to the Internet curve as ε grows to 0.1 and 0.2. However, when ε reaches 0.3 the CCDF overshoots the Internet curve in the high-degree area (above 800 neighbors), and undershoots the Internet curve in the mid range (10-800 neighbors). This “S” shape becomes even more pronounced with $\varepsilon = 0.4$ or higher (curves omitted). Intuitively, the SLiP model makes the high-degree nodes more attractive at the expense of low- and mid-degree nodes, and setting ε too high amplifies this behavior beyond what is observed in reality. We can see that the networks with the most realistic degree distribution are generated with $\varepsilon = 0.2$, in which case the power-law exponent is $\eta = 1.13$.

6.2 Dense Core Analysis

In order to find the Dense Core in the networks, we used the Dense k -Subgraph (DkS) algorithms of [FKP01,SW03a]. These algorithms search for the densest cluster (sub-graph) of a prescribed size ℓ . Fig. 2 shows the edge density of the densest cluster found by the algorithms, as a function of ℓ . For the synthetic

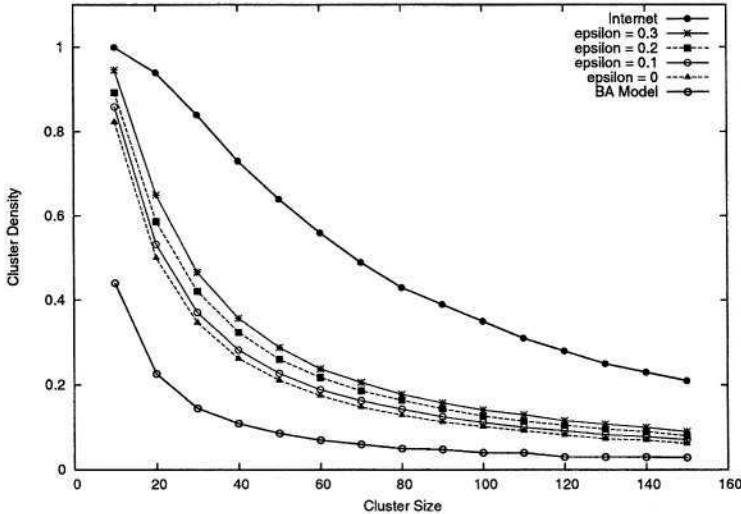


Fig. 2. The edge density $\varrho(\ell)$ of the densest ℓ -cluster, as a function of the cluster size ℓ .

networks, each point on the curves is the average over 10 random networks generated with the same parameters.

The figure clearly shows that for Internet-like parameters, the pure BA model does not produce significant a dense core: This is not surprising in view of the results of Sagie and Wool [SW03b], who proved that the BA model is fundamentally *unable* to produce synthetic topologies with a dense core larger than $\ell = 6$ with $\varrho(\ell) \geq 70\%$. In contrast, the real AS graph has a dense core of $\ell = 43$ ASes with $\varrho(\ell) \geq 70\%$.

The figure does show that the TANG-generated networks have dense cores that are closer to reality than those produced by the pure BA model: we see that a density of $\varrho(\ell) \geq 70\%$ is achieved around $\ell \in [17, 20]$, and that higher values of ε produce larger dense cores. In fact, for any value of ℓ , the density $\varrho(\ell)$ of the TANG-generated networks is at least twice the density of the BA networks. Thus, as far as dense clusters go, TANG is significantly closer to reality than the BA model.

However, the figure also, shows that dense cores of TANG networks still fall short: they are roughly half as dense as their counterparts in the AS graph. Furthermore, increasing ε only produces a slow increase in the density of the core, and we already saw in Section 6.1 that increasing ε beyond 0.2 distorts the degree distribution away from a power law. Thus, we conclude that the SLIP/InEd model is a significant improvement in terms of the dense core—but it is not sufficient to produce realistic cores.

7 Conclusions and Future Work

We have shown that our extensions to the BA model, the InEd and SLiP models, significantly improve upon the pure BA model in terms of matching the power-law parameter, producing leaves, and producing a large dense core. Our models are amenable to mathematical analysis, and are implemented as a freely available network generator.

However, more work is possible: The current model still does not produce a satisfactory dense core. It seems that new ideas are necessary to create a model that can (i) produce larger dense cores, (ii) maintain a power law degree distribution, and (iii) remain simple enough to analyze.

References

- [AB00] Réka Albert and Albert-László Barabási. Topology of evolving networks: Local events and universality. *Physical Review Letters.*, 85(24):5234–5237, December 2000.
- [BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 15 October 1999.
- [BT02] T. Bu and D. Towsley. On distinguishing between Internet power-law generators. In *Proc. IEEE INFOCOM’02*, New-York, NY, USA, April 2002.
- [CCG⁺02] Q. Chen, H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. The origin of power laws in Internet topologies revisited. In *Proc. IEEE INFOCOM’02*, New-York, NY, USA, April 2002.
- [ER60] P. Erdős and A. Renyi. On the evolution of random graphs. *Magyar Tud. Akad. Mat. Kutato Int. Kozl.* 5:17–61, 1960.
- [FKP01] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001.
- [MLMB01] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. In *Proceedings of MASCOTS’01*, August 2001.
- [SARK02] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz. Characterizing the Internet hierarchy from multiple vantage points. In *Proc. IEEE INFOCOM’02*, New-York, NY, USA, April 2002.
- [SFFF03] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos. Power-laws and the AS-level Internet topology, 2003. To appear in IEEE/ACM Trans. on Networking.
- [SW03a] G. Sagie and A. Wool. A clustering approach for exploring the Internet structure. Technical Report EES2003-7, Dept. Electrical Engineering Systems, Tel Aviv University, 2003. Available from <http://www.eng.tau.ac.il/~yash/ees2003-7.ps>.
- [SW03b] G. Sagie and A. Wool. A clustering-based comparison of Internet topology models, 2003. Preprint.
- [WJ02] Jared Winick and Sugih Jamin. Inet-3.0: Internet topology generator. Technical Report UM-CSE-TR-456-02, Department of EECS, University of Michigan, 2002.
- [Woo04] A. Wool. TANG: Tel aviv university network generator, v1.4, 2004. Available from <http://www.eng.tau.ac.il/~yash/topology/index.html>.

Geometric Exploration of the Landmark Selection Problem

Liying Tang and Mark Crovella

Department of Computer Science, Boston University, Boston, MA 02215
`{litang, crovella}@cs.bu.edu`

Abstract. Internet coordinate systems appear promising as a method for estimating network distance without direct measurement, allowing scalable configuration of emerging applications such as content delivery networks, peer to peer systems, and overlay networks. However all such systems rely on landmarks, and the choice of landmarks has a dramatic impact on their accuracy. Landmark selection is challenging because of the size of the Internet (leading to an immense space of candidate sets) and because insight into the properties of good landmark sets is lacking. In this paper we explore fast algorithms for landmark selection. Whereas the traditional approach to analyzing similar network-based configuration problems employs the graph structure of the Internet, we leverage work in coordinate systems to treat the problem geometrically, providing an opening for applying new algorithms. Our results suggest that when employing small numbers of landmarks (5-10), such geometric algorithms provide good landmarks, while for larger numbers of landmarks (20-30) even faster methods based on random selection are equally effective.

1 Introduction

Internet coordinate schemes assign coordinate vectors to hosts in the Internet, and attempt to do so in a manner such that the Euclidean distance between hosts approximates their network distance, usually taken to be minimum RTT [7]. The power of such coordinate systems is that they allow network distance to be estimated in the absence of measurement. As a result, applications for which network distance matters (such as content delivery networks, peer to peer applications, end-system multicast, and others) can be rapidly configured in a network-sensitive manner based on static input data (*i.e.*, a database of nodes and their coordinates).

A number of proposals have been put forward for such systems, ranging from initial proposals based on expensive nonlinear optimization [7,14] to more recent fast methods based on Lipschitz embedding and dimensionality reduction [6,13]. All such proposals start from a set of inter-node measurements. For scalability, it is assumed that all nodes desiring coordinates cannot measure all distances of interest (*i.e.*, distances to all other nodes), and so must choose a set of *landmarks* by which to establish their coordinates. Thus, the usual arrangement is that all nodes measure their distances to a fixed set (or at least to a fixed number) of landmarks, and use those measurements to determine their coordinates, so that measurement load in the network scales linearly with the number of nodes. Early work assumed a fixed set of landmarks, which recent proposals have

suggested that measurement load may be distributed by using multiple landmark sets [13,10].

Thus, in order to deploy coordinate systems, specific landmarks must be chosen. There is reason to believe that the accuracy of Internet coordinate systems is sensitive to the particular choice of landmarks. First, larger landmark sets are generally more accurate, because they bring more information to bear on the coordinate construction step. Second, two landmarks that are “close” to each other in the Internet may provide little discriminatory power – measurements to both landmarks will be nearly the same and will therefore add little information in the coordinate construction step.

Because of the large number of possible landmarks (hundreds of millions in the Internet) and the complex relationship between landmarks used and the accuracy of the resulting coordinate scheme, optimal landmark selection is a very challenging problem. A natural approach in seeking heuristics to guide the search is to look for structure in the distance measurements found in the Internet, and to try to place landmarks in ways that exploit that structure.

In this paper we examine the structure of Internet distance measurements, and ask whether that structure can help us select good landmarks in an efficient way. While network structure is usually thought of as a property of topology, we take a novel approach that leverages the notion of internet coordinates: we examine the *geometric* structure of network distances. We show that this structure is characterized by a high degree of clustering.

We then proceed to ask what sorts of geometric algorithms, including those explicitly making use of clusters, can be used to efficient landmark selections. We study this question using the Virtual Landmarks embedding method and find that the answer depends on the size of the landmark set. Our initial results show that when using a small number of landmarks (5-10), the proper choice of landmarks is quite important; but when a larger number of landmarks is used, even random selection is as effective as geometrically-informed approaches

Our results point the way to the ability to perform landmark selection in very large datasets (since the geometric methods are relatively scalable) and suggest that if 20-30 landmarks can be used, then landmark selection can be done quite efficiently.

2 Related Work

Internet coordinate schemes were first proposed in [7]. In this work, we use the Virtual Landmarks method for forming internet coordinates as described in [13]. The Virtual Landmarks method is based on two ideas: First, it uses a Lipschitz embedding of nodes into a high dimensional space. In a Lipschitz embedding, the distances to a set of landmarks are taken as the coordinates of the given node. Second, it uses dimensionality reduction via Principal Component Analysis to reduce the higher-dimensional space of Lipschitz embedding to a lower-dimensional space. In this paper, all coordinates are 7-dimensional, which was found in [13] to provide a reasonable tradeoff between low dimension and accuracy. Thus, the algorithms in this paper are concerned with selecting landmarks for the Lipschitz embedding, while all error metrics are measured based on the coordinate assignments after reducing to 7 dimensions.

The question of how to select landmarks has not been explicitly addressed in previous Internet coordinate work. However it has relevance not just for Internet coordinates, but for all methods that employ Lipschitz embeddings on round-trip times.

A number of papers have used Lipschitz embeddings to assign coordinates to hosts. In [16], the goal is to map an arbitrary host to a nearby landmark whose geographic position is known. A landmark whose Lipschitz coordinates are similar to that of the host is used as the estimated location of the host. A demographic placement approach was proposed to improve the representativeness of landmarks with respect to the hosts to be located. Like this work, they also explore placing probe machines in a geographically distributed manner, to avoid shared paths.

In [9], the authors used a Lipschitz embedding to find nearby nodes for the purpose of geolocation. The number of probe machines and locations was studied in order to improve accuracy. However, specific algorithms for landmark selection were not investigated.

Placement of other Internet resources, such as caches and mirrors, has been investigated (*e.g.*, in [8] and [3]) however these studies have not looked at geometrically inspired algorithms.

In computer vision, Lipschitz embeddings have been used to generate fast indexing schemes for images. The problem for the choice of landmarks in this setting has been studied in [2,15]. The authors in [2] propose four methods, including minimizing average distortion, which is similar to our Greedy method. Maximum distance methods similar to those we employ have been studied in [15]. However, the results are mainly relevant to the kinds of similarities found among images and it is not clear that their conclusions would apply to the problem of Internet distances.

Finally, a number of papers have proposed scalability methods for Internet coordinate schemes based on construction of multiple landmark sets [13,10]. Such methods assume the existence of algorithms for constructing landmark sets on the fly; the work in this paper can inform the choice of those algorithms.

3 Data

We use 3 datasets in this work, which are the same as used in [13].

NLANR AMP. The NLANR Active Measurement Project [1] collects a variety of measurements between all pairs of participating nodes. Most participating nodes are at NSF supported HPC sites, and so have direct connections to the Abilene network; about 10% are outside the US. The dataset we use was collected on January 30, 2003 and consists of measurements of a 116×116 mesh. Each host was pinged once per minute, and network distance is taken as the minimum of the ping times over the day's worth of data.

Skitter. The Skitter project [11] is a large-scale effort to continuously monitor routing and connectivity across the Internet. It consists of approximately 19 active sites that send probes to hundreds of thousands of targets. Targets are chosen so as to sample a wide range of prefixes spanning the Internet. Results reported in [5] suggest that about 50%

of the targets in this dataset are outside the US. The dataset we used was collected in December 2002; each target was pinged approximately once per day. Network distances are the minimum ping time over 12 days. The set of targets varies among active sites; selecting the largest set of rows for which complete data is available yields a 12×12 symmetric dataset, and a $12 \times 196,286$ asymmetric dataset with 2,355,565 entries.

Sockeye. Our last dataset was collected at Sockeye Networks [12]. It consists of measurements from 11 active sites to 156,359 locations. Targets were chosen through a scheme designed to efficiently explore as much of the routable Internet as possible. Each active site sent a ping to each target on an hourly basis. Network distance was taken as the minimum of all pings over a single day.

4 Algorithms

Each algorithm we consider selects a set of ℓ landmarks $\{L_i\}, i = 1, \dots, \ell$ taken from datasets containing n hosts $H = \{h_i\}, i = 1, \dots, n$. The network distance (RTT) between h_i and h_j is denoted $d(h_i, h_j)$. Each algorithm takes as input an Internet coordinate scheme $\phi : H \rightarrow \mathbb{R}^d$ and a distance function $\delta : \mathbb{R}^d \rightarrow R$. In this paper ϕ represents the mapping obtained from the Virtual Landmarks embedding into Euclidean space with $d = 7$, and δ is the Euclidean norm.

Greedy. The most expensive approach is the *Greedy* algorithm [2]. The Greedy algorithm chooses the set of landmarks that minimizes the average distortion of distances at each step. We define the relative error function $Q(h_i, h_j, \phi)$ for a pair of hosts $h_i, h_j \in H$ as

$$Q(h_i, h_j, \phi) = \frac{\delta(\phi(h_i), \phi(h_j)) - d(h_i, h_j)}{d(h_i, h_j)}$$

The accuracy of the embedding ϕ can be calculated by the *average absolute error*:

$$\bar{Q}(\phi) = 2/(n^2 - n) \sum_{i>j} |Q(h_i, h_j, \delta)|$$

Exact minimization of $\bar{Q}(\phi)$ is computationally infeasible for large n and ℓ . A step-wise approach that is tractable, but still quite expensive, is the Greedy algorithm, which proceeds iteratively. The first landmark L_1 is chosen at random. Subsequent landmarks are chosen so as to give the lowest average absolute error when used together with the already-chosen landmarks. That is, in step m we choose L_m so as to minimize $\bar{Q}(\phi)$ when used with landmark set L_1, \dots, L_m .

K-means. The *k-means* algorithm is a geometric algorithm that operates on distances as computed using an Internet coordinate scheme. *K-means* finds ℓ disjoint clusters in the data [4]. It starts by choosing ℓ hosts to act as centroids (c_1, \dots, c_ℓ) at random. Each centroid c_j has an associated cluster G_j . The algorithm then repeatedly performs the following three steps:

1. Assign each host $h \in H$ to the cluster G_j with the nearest centroid c_j .
2. For each cluster G_j , calculate a new center $c = \sum_{h \in G_j} \frac{h}{|G_j|}$
3. Assign a new centroid c_j to G_j as the host nearest to this center c .

These steps are iterated until the centroids and the clusters become stable. Then, these ℓ centroids chosen by the above k-means algorithm are assigned as the landmarks.

Maximum Distance. The *Maximum Distance* algorithm is also a geometric algorithm. It attempts to find the maximally-distributed landmarks from the set of n hosts, based on the intuition that landmarks must be spread far apart to give useful location information [15]. The ℓ landmarks are chosen in an iterative manner. The first landmark L_1 is chosen from the set H at random. In iteration m ($1 < m \leq \ell$) the distance from a host h_i to the set of already chosen landmarks L_1, \dots, L_{m-1} is defined as the $\min_{L_j} \delta(\phi(h_i), \phi(L_j))$. The algorithm selects as landmark L_m the host that has the maximum distance to the set L_1, \dots, L_{m-1} .

Random. The *Random* method is the most efficient and simplest among all four algorithms. The ℓ landmarks are randomly chosen from the n hosts.

5 Clusters

One reason for considering geometric algorithms for landmark placement is that Internet hosts show a high degree of clustering. In this section we explore empirically this clustering in our datasets.

As described in Section 2 our data consists of Internet hosts whose coordinates are points in \mathbb{R}^n with $n = 7$. We can start to examine the evidence for clustering in our datasets by looking at projections of the points onto various subspaces. When using the Virtual Landmarks method for coordinate assignment, coordinates are assigned in order of significance. The first coordinate captures the largest amount of variation possible in a single dimension, the next coordinate captures the largest amount of variation in the remaining dimensions, and so on [13]. Thus the most significant projections of the resulting data are onto the subspaces spanned by the initial axes of the coordinate system.

In Figure 1(a) we show the projection of the Skitter dataset onto the space spanned by the first two axes. In Figure 1(b) we show the corresponding plot for the Sockeye dataset. To reduce plot file size we plot a random sample of 30,000 hosts from each of our datasets; plots of the entire dataset show exactly the same features.

These plots show the remarkable amount of clustering (as well as other types of structure) that is present when Internet hosts are embedded in Euclidean space. Although the two plots represent data that was collected at different times from completely different measurements infrastructures (having different probes and target locations), they show very similar structure. The number, sizes, and relative positions of the largest clusters in each dataset are very similar.

Although these two datasets were collected in different ways, the underlying data collection strategy in each case was to try to sample all portions of the space of live

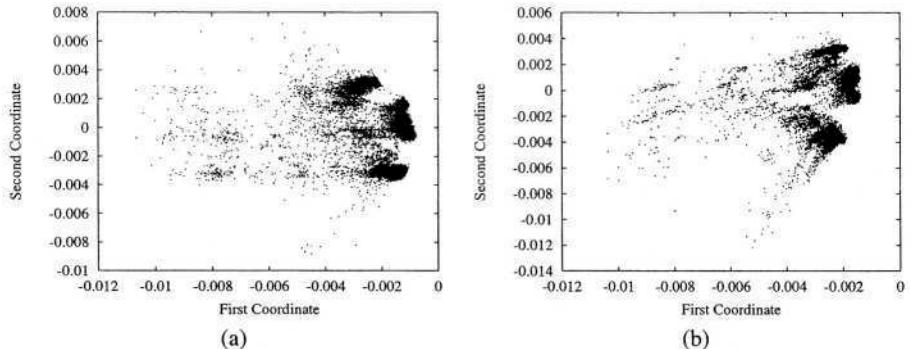


Fig. 1. Clusters in (a) Skitter Data and (b) Sockeye Data

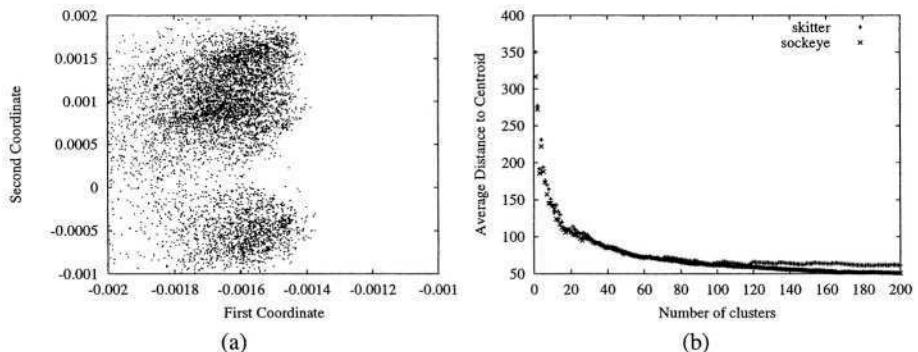


Fig. 2. (a) Zoom In on Clusters in Sockeye Data (b) Average Distance to Cluster Centroid as a Function of Number of Clusters

Internet addresses. Thus we interpret the similarity between Figures 1(a) and 1(b) as evidence that the structure exhibited reflects real properties of Internet host connectivity.

We can also observe that clustering is present on smaller scales as well. Figure 2(a) zooms in on a part of Figure 1(b) corresponding to the largest visible cluster. This figure shows that even within a cluster, host density varies across different regions, suggesting the presence of clustering at multiple scales.

To assess the number of clusters present in our data, we adopted the following approach. Using the *k*-means clustering algorithm as described in Section 4, we constructed clusters for varying values of *k* and measured the mean cluster diameter. The resulting curves for the skitters and sockeye datasets are shown in Figure 2(b). This figure shows an inflection point around 20 clusters, indicating the presence of at least 20 clusters in both datasets.

It is very likely that these clusters are influenced by the geographical location of hosts. This can be seen in Figure 3 where we have identified the clusters formed with $k = 6$ for the AMP dataset. Although the clustering algorithm uses no direct information about geographical location, it produces clusters that are in fact geographically distinct.

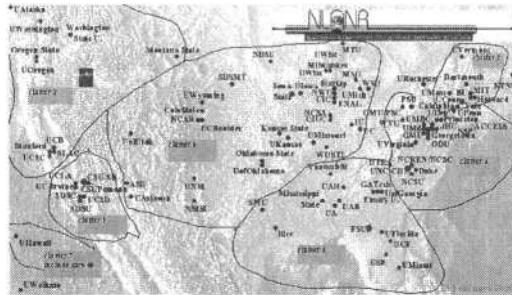


Fig. 3. Clusters in AMP data (based on graphic from [1]).

6 Algorithms for Landmark Selection

Motivated by these geometric considerations, in this section we evaluate algorithms for landmark selection.

We average absolute relative error as our principal metric. That is, for each algorithm for landmark selection, leading to an embedding ϕ , we report $\bar{Q}(\phi)$.

In addition, the entire distribution of relative error is of interest. In particular it is important to know the variability of relative error for various landmark selection methods. For these reason we also report the 90th percentile of relative error.

In applying the algorithms for landmark selection we proceed as follows. In the case of the AMP dataset, we select ℓ landmarks from the 116 hosts using each of the algorithms. We then evaluate the relative error of the Virtual Landmarks embedding of the remaining $116 - \ell$ hosts. In the case of the Skitter and Sockeye datasets, we subsample 2000 hosts at random from the set of measurement targets. We then select ℓ landmarks from this set, and evaluate the relative error of the Virtual Landmarks embedding on the active sites. For Skitter, this means we are evaluating the relative error over 12 hosts, and for Sockeye, 11 hosts. Although these sample sizes are small, the consistency of our results suggests that they are reasonably representative.

In Figures 4, 5, and 6 we present our main results. On the left in each figure is the average absolute relative error, and on the right is the 90th percentile of absolute relative error.

In the case of the AMP hosts (Figure 4), the Greedy algorithm performs distinctly better than the others, regardless of the number of landmarks. The Random algorithm, and the geometric algorithms (K -means and Maximum Distance) have very similar performance, although the worst-case performance of Random is slightly poorer than the other two.

These results may be understood in light of the nature of the AMP hosts, which are generally sites in North America, with good connections to the high-speed Abilene network. For these hosts, the particular choice of landmark set is not too critical, although an expensive (Greedy) approach yields some benefits.

The situation is rather different when looking at lardmark sets that span the Internet as a whole, with more heterogeneous connectivity (Skitter and Sockeye datasets in Figures 5 and 6). Here the Greedy algorithm is still best, as expected, but there is

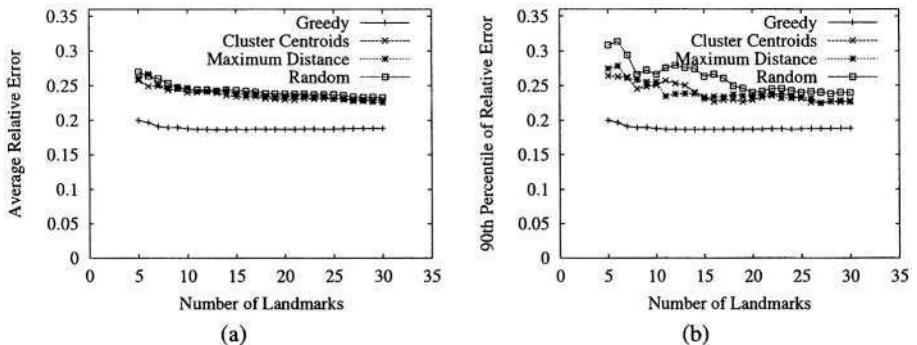


Fig. 4. (a) Average Absolute Relative Error and (b) 90th Percentile of Relative Error of Landmark Selection Algorithms for AMP Data

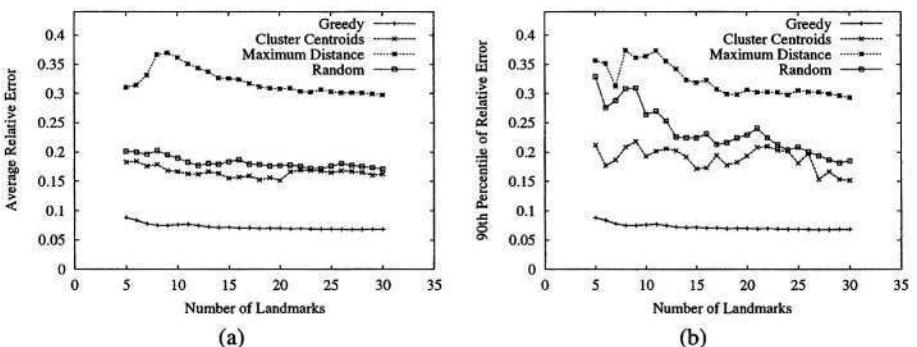


Fig. 5. (a) Average Absolute Relative Error and (b) 90th Percentile of Relative Error of Landmark Selection Algorithms for Skitter Data

a distinct difference between the others. Maximum Distance performs rather poorly, possibly because the landmarks it finds, while far from most hosts, have relatively little path diversity to the set of hosts, and therefore provide poor location information. K -means performs well even when using only a small set of landmarks. The performance of Random is poor when using a small set of landmarks, but surprisingly, it is comparable to k -means when using a large set of landmarks. This effect is especially strong when the metric of interest is the 90th percentile of absolute relative error.

These results suggest the following conclusions. First, the best-performing approach is always the expensive Greedy algorithm; if resources permit, this is the best algorithm. However, if a more efficient algorithm is needed, then for a small number of landmarks (5-10) the geometrically-based k -means algorithm is best; however, an even simpler and faster option is to expand the larkdmk set (to approximately 20-30 landmarks) and simply use Random selection.

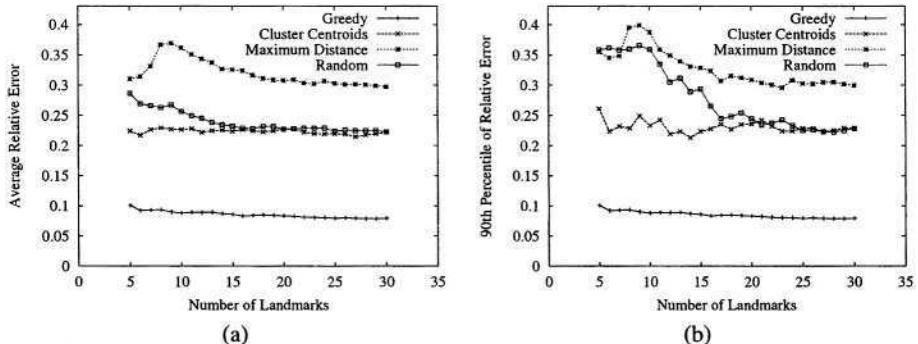


Fig. 6. (a) Average Absolute Relative Error and (b) 90th Percentile of Relative Error of Landmark Selection Algorithms for Sockeye Data

7 Conclusions

In summary, we have advocated a geometric approach to Internet location analysis, and based on that we have described and evaluated useful algorithms for landmark selection. Motivated by the evidence of significant clustering among Internet hosts, we explored algorithms based on *k-means* and maximum distance, and compared them to greedy and random alternatives. We find that the greedy algorithm is best in all cases; however, it is computationally expensive. Among the more efficient algorithms, we find that exploiting clustering (*k-means*) is effective when the number of landmarks to be chosen is small (5-10). However, if 20-30 landmarks are employed, then even simple random selection works well. These results suggest that effective landmark selection appears feasible even on the scale of the Internet.

Acknowledgements. The Sockeye data used in this paper was collected at Sockeye Networks, www.sockeye.com. The NLANR AMP data used in this paper was collected by the National Laboratory for Applied Network Research with funding under National Science Foundation Cooperative Agreement ANI-9807479. The Skitter data was collected as part of CAIDA's Skitter initiative, <http://www.caida.org>. Support for Skitter is provided by DARPA, NSF, and CAIDA membership. The authors wish to express their appreciation to all these people and organizations for their data collection and distribution efforts.

This work was performed while Mark Crovella was at Laboratoire d'Informatique de Paris 6 (LIP6), with support from Centre National de la Recherche Scientifique (CNRS) France.

This work was partially supported by NSF grants ANI-9986397, ANI-0095988, and ANI-0322990.

References

1. The NLANR active measurement project. <http://amp.nlanr.net/active/>.
2. V. Athitsos and S. Sclaroff. Estimating 3D hand pose from a cluttered image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2003.
3. E. Cronin, S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. *IEEE Journal on Selected Areas in Communications*, 20(7): 1369–1382, 2002.
4. L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley and Sons, Inc., 1990.
5. A. Lakhina, J. W. Byers, M. Crovella, and I. Matta. On the geographic location of Internet resources. *IEEE Journal on Selected Areas in Communications, Special Issue on Internet and WWW Measurement, Mapping, and Modeling*, 2003.
6. H. Lim, J. C. Hou, and C.-H. Choi. Constructing Internet coordinate system based on delay measurement. In *Proceedings of the ACM/SIGCOMM Internet Measurement Conference (IMC-03)*, 2003.
7. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proceedings of Infocom*, 2002.
8. Y. S. P. Krishnan, Danny Raz. The cache location problem. *IEEE/ACM Transactions on Networking*, 8(5):568–582, 2000.
9. V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for Internet hosts. In *Proceedings of ACM/SIGCOMM '01*, August 2001.
10. M. Pias, J. Crowcroft, S. Wilbur, S. Bhatti, and T. Harris. Lighthouses for scalable distributed location. In *Second International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Feb 2003.
11. The skitter project. <http://www.caida.org/tools/measurement/skitter/>.
12. Sockeye Networks. <http://www.sockeye.com/>.
13. L. Tang and M. Crovella. Virtual landmarks for the Internet. In *Proceedings of the ACM/SIGCOMM Internet Measurement Conference 2003*, October 2003.
14. T. Tankel and Y. Shavitt. Big-bang simulation for embedding network distances in Euclidean space. In *Proceedings of IEEE INFOCOM 2003*, April 2003.
15. J. Vleugels and R. C. Veltkamp. Efficient image retrieval through vantage objects. *Visual Information and Information Systems(VISUAL), LNCS 1614*, pages 575–584, June 1999.
16. A. Ziviani, S. Fdida, J. F. de Rezende, and O. C. M.B.Duarte. Demographic placement for Internet host location. In *Proceedings of the IEEE Global Communications Conference*, 2003.

The Interdomain Connectivity of PlanetLab Nodes

Suman Banerjee¹, Timothy G. Griffin², and Marcelo Pias²

¹ University of Wisconsin, Madison. suman@cs.wisc.edu

² Intel Research, Cambridge UK. {tim.griffin,marcelo.pias}@intel.com

Abstract. In this paper we investigate the interdomain connectivity of PlanetLab nodes. We note that about 85 percent of the hosts are located within what we call the *Global Research and Educational Network* (GREN) — an interconnected network of high speed research networks such as Internet2 in the USA and Dante in Europe. Since traffic with source and destination on the GREN is very likely to be transited solely by the GREN, this means that over 70 percent of the end-to-end measurements between PlanetLab node pairs represent measurements of GREN characteristics. We suggest that it may be possible to systematically choose the placement of new nodes so that as the PlanetLab platform grows it becomes a closer and closer approximation to the Global Internet.

1 PlanetLab

The primary goal of PlanetLab is to provide a geographically distributed platform for overlay-based services and applications [12,1]. Currently there are over 120 participating sites with a total of over 300 hosted nodes. The open and shared nature of PlanetLab is enabling an exciting range of innovative experimentation in overlay techniques. Without its collectively supported infrastructure most participating research groups would not have the means to set up such a rich environment. For many of the same reasons, the PlanetLab infrastructure has attracted researchers working with measurements of the *legacy Internet*. That is, PlanetLab nodes are employed to actively probe and collect various Internet metrics. We argue that this is an *opportunistic* use of PlanetLab in the sense that providing an Internet measurement infrastructure has never been among the primary goals of the project. This does not mean that PlanetLab is not a useful Internet measurement platform, only that PlanetLab measurements cannot *automatically* be taken as representative of the global Internet.

In this paper we investigate the interdomain connectivity of PlanetLab nodes. We note that about 85 percent of the hosts are located within what we call the *Global Research and Educational Network* (GREN) — an interconnected network of high speed research networks such as Internet2 in the USA and Dante in Europe. Since traffic with source and destination on the GREN is very likely to be transited solely by the GREN, this means that over 70 percent of the end-to-end measurements between PlanetLab node pairs represent measurements of GREN characteristics. Whether or not such measurements are representative of the global Internet is something that needs more investigation.

This should in no way be misconstrued as a criticism of PlanetLab — we are only stating that those using PlanetLab for measurements of the global legacy Internet need to present their arguments with care. On the other hand, the GREN is in some respects

more attractive to measurement researchers than the Internet at large. Primarily this is because it is more *transparent* — that is, there is more publicly available information about the connectivity of the GREN and more willingness on the part of its operators to share information with the research community. We suggest that it may be possible to systematically choose the placement of new nodes so that as the PlanetLab platform grows it becomes a closer and closer approximation to the Global Internet. We advocate that there is still a large amount of untapped diversity within GREN which can be explored for similar effect on the PlanetLab.

We present some preliminary results on a case study. We generate a site-to-site distance matrix for PlanetLab sites where distances between sites is taken to be minimum round trip time. We then enumerate all possible triangles formed by three sites and investigate the violations of the triangle inequality. Low violations are important for the feasibility of various proposals to generate synthetic coordinate systems for the Internet based on round trip time measurements [11,13,17]. We find that when we classify triangles as “research triangles” (all nodes on the GREN), “commercial triangles” (all nodes off of the GREN), and “mixed triangles” (combination of commercial and GREN sites). We find that the distribution of “bad triangles” is lowest for research triangles (about 12 percent), higher for mixed triangles (about 20 percent), and highest for commercial triangles (about 25 percent).

2 The Global Research and Education Network (GREN)

The global Internet is comprised of a large collection of autonomously administered networks. Some of these networks are operated by commercial enterprises, while others are operated by nonprofit organizations. Perhaps the largest nonprofit networking organizations exist to provide connectivity between research and academic institutions. This includes the Geant backbone network run by the Dante organization in Europe and the Abilene backbone network run by Internet2 in North America. Such backbones connect many regional and national research networks into a large global network providing connectivity between diverse academic and research organizations. We refer to this network as the *Global Research and Education Network* (GREN).

Figure 1 presents a simplified picture of the current GREN. The GREN is *not* by any means a single administrative entity. All of the networks are independently administered, and exhibit various degrees of cooperation. There is also large diversity in the primary goals of these networks. Some regional networks are targeted toward a specific set of users, while others serve a larger research and education community. For example, the CERN network exists largely to provide high bandwidth connectivity between physics laboratories around the world. On the other hand, the WiscNet of Wisconsin (<http://www.wiscnet.net>) exists to provide connectivity between K-12 educational and university level institutions. Some GREN networks provide transit to commercial network providers, while others do not. For example, the backbone of Internet2, the Abilene network, does not provide commercial connectivity, while WiscNet does. The ARENA project (<http://arena.internet2.edu>) provides a very useful online compendium of information about the networks making up the GREN. It should also be remembered that the GREN is continually changing and expanding. For example, to Latin America from

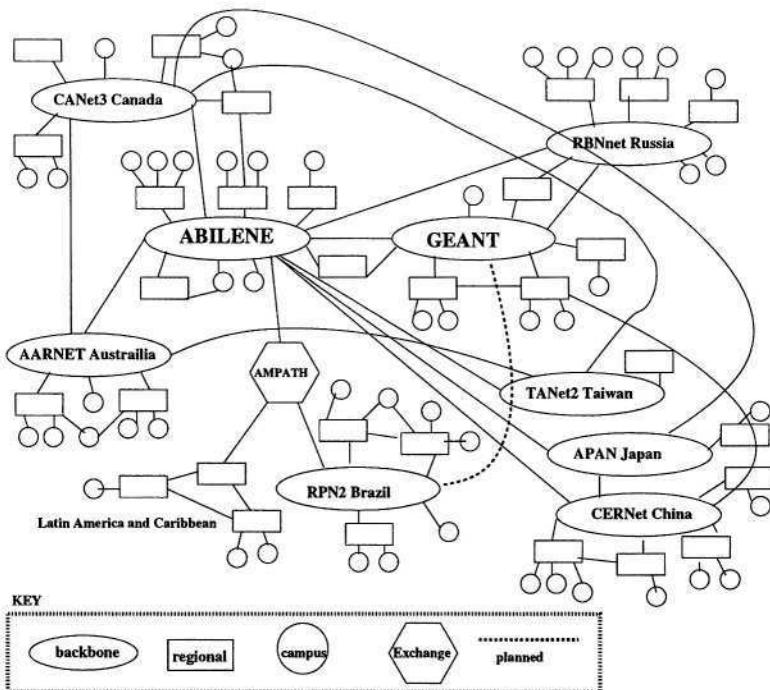


Fig. 1. A highly simplified and incomplete schematic of the GREN.

Europe, GREN traffic is routed today through the USA (Florida). A directed link between Geant and the Latin American research networks is planned within the scope of the ALICE project [15]. This will create a “short-cut” between these networks, which is likely to have an effect on how experimental results should be interpreted. Similarly, other new links are planned between Europe and USA.

The logical connectivity of the GREN networks does not immediately tell us how traffic between them is *routed*. Figure 2 depicts two sites A and B that have connectivity to both the GREN and the commercial Internet. One would normally expect that both sites A and B prefer routes from the research network over routes learned from their commercial providers. There are several reasons for this, the primary one being that research connectivity is normally paid for out of a special source of funds, and it may not be metered as it is normally done with commercial traffic. In addition, there may be the perception that for research work, the research network will give better performance. In this way, we can think of the research network as providing a “short cut” between A and B that bypasses the commercial Internet. Of course, this may be occasionally overridden by local routing policies. The actual connectivity of most sites is more complex than Figure 2. For example, Figure 3 shows the connectivity of two PlanetLab sites: planetlab2.cs.unibo.it and planetlab2.cs.wisc.edu. We have verified our assumptions about routing policies with “AS level traceroutes”. For the above example this yields:

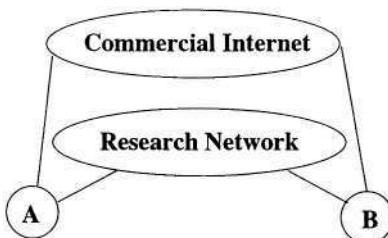


Fig. 2. Simple schematic of the routing choices as A and B

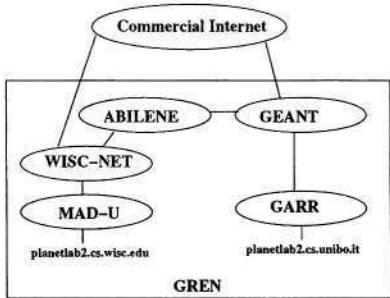


Fig. 3. A slightly more complicated example.

AS 137 : GARR	--- Italian REN	(5 routers)
AS 20965 : GEANT	--- Dante Backbone	(3 routers)
AS 11537 : ABILENE	--- Internet2 backbone	(2 routers)
AS 2381 : WISCNET	--- Wisconsin REN	(3 routers)
AS 59 : MAD U	--- U. of Wisconsin, Madison	(5 routes)

We compute these by first performing router-level traceroutes between PlanetLab nodes using Scriptroute [2] and then post-processing the results to associate intermediate routers with ASNs. With only a few exceptions, our expectations about routing were confirmed.

3 PlanetLab and the GREN

Since those engaged in the building of PlanetLab and the deployment of overlay services are largely academically oriented researchers, it should not be surprising that the majority of PlanetLab sites are located in the GREN. However, as far as we know there has never been an attempt to systematically quantify this relationship. We must determine whether a given PlanetLab host or site is connected to the GREN or the commercial Internet. We use the following approach. First, we obtain a reasonable list of ASes in the GREN. This was done by extracting from Route-Views [3] BGP routes only those routes that were announced by Abilene (this can be found in the “oix full snapshot” source). This BGP session with Route-Views announces only research destinations (about 8,000 routes). In a perfect world, the ASNs in the AS-PATHS of these routes would be exactly the ASNs of the GREN. However, a small number of routes from the commercial Internet get “leaked” into these tables (for reasons still unclear), and so we must eliminate some ASes that are known to be commercial providers (Sprint, MCI, and so on). On January 15, 2004, this procedure left us with a list of 1,123 ASNs for the GREN.

Next, for each IP address associated with a PlanetLab node, we find the originating ASN associated with the longest matching prefix. In order to generate a reasonable mapping of originating ASNs to prefixes, we merged routing information from 20 BGP tables taken from RIPE [10] and Route-Views. This is essentially the basis of the technique described in [9,8] for obtaining AS-level traceroutes. If the originating AS is in our GREN list, we classify the node as a GREN node. Otherwise it is a commercial node.

If the address is associated with Multiple Originating ASNs, some being commercial and others research, we classify the site as MOAS. With the 276 production nodes of January 15, 2004, we obtain the following breakdown as shown in Table 1.

Table 1. Breakdown of hosts.

class	number	percent
MOAS	14	5.1
Commercial	27	9.8
Research	235	85.1

Table 2. Breakdown of sites.

class	number	percent
MOAS	7	5.7
Commercial	12	9.8
Research	104	84.5

We then group the hosts into *sites* containing multiple hosts. This produces 123 sites with a breakdown very close to that of the hosts (Table 2).

Note that this means that over 70 percent of the end-to-end measurements between PlanetLab node pairs represent measurements of GREN characteristics. We then classified these sites into one of four broad geographical regions: North America (NA), Europe the Middle East and Africa (EMEA), Asia Pacific (AP), and Latin America (LA). The breakdown of the sites are listed in Table 3.

Table 3. Breakdown by region.

region	number	percent
LA	1	0.8
AP	6	4.9
EMEA	25	20.3
NA	91	74.0

Table 4. Combination of regions and types.

class	region	number	percent
Commercial	EMEA	1	0.8
Research	LA	1	0.8
Research	AP	6	4.9
Commercial	NA	11	8.9
Research	EMEA	23	18.7
Research	NA	74	60.2
Research	NA or EMEA	97	78.9

We now look at the combinations of these two classifications (Table 4). In this table we have ignored the MOAS sites (although they are still counted when computing percentages) Note that this means that over 60 percent of the end-to-end measurements between PlanetLab node pairs represent measurements of the NA and EMEA portion of the GREN. In summary, we note that not only are PlanetLab nodes situated in the GREN corner of the Internet, but inhabits a fairly small part of that world as well. The majority of site-to-site data flows are carried over the Abilene and Geant networks. Contrast this with the rich diversity illustrated in Figure 1.

3.1 Case Study — The Triangle Inequality

A number of applications would benefit from a global topological map of the Internet, which incorporates information such as the number of hops and the network latency between Internet hosts. Towards this goal, research in recent years have focused on distributed techniques to construct an *Internet Coordinate System*, in which each host

can be assigned a virtual coordinate and the distance between the virtual coordinates of any two hosts will approximate their actual distance for some specific network performance metric. Network latency is an example of such a metric. There may be one such coordinate system for each metric of interest. Some of the proposed techniques are Global Network Positioning (GNP) [11], Lighthouses [13], Virtual Landmarks [17], ICS [7] and Practical Internet Coordinates (PIC) [4]. The goal in these techniques is to significantly reduce the number of measurements required, thereby scaling to large number of hosts. There are many applications that can leverage such an Internet Coordinate System, e.g. topologically-aware construction of peer-to-peer systems [5], network latency estimation tools [16], and distributed resource discovery mechanisms [14]. The general problem to construct such a coordinate system can be abstracted as the problem of mapping, or *embedding*, a graph into a metric space. A metric space M is defined by the pair (X, d) where X represents the set of valid objects and d is a metric. A metric is a function $d : X \times X \rightarrow \mathbb{R}$ such that for $x_i, x_j, x_k \in X$, d satisfies the following properties:

1. $d(x_i, x_j) \geq 0$ (positiveness),
2. $d(x_i, x_j) = d(x_j, x_i)$ (symmetry),
3. $d(x_i, x_j) \leq d(x_i, x_k) + d(x_k, x_j)$ (triangle inequality)

In contrast, a vector space V is a particular metric space in which $X = \mathbb{R}^k$ and it has distance function D that satisfies properties (1), (2) and (3). The embedding problem consists of finding a scalable mapping $\gamma : X \rightarrow \mathbb{R}^k$ that transforms *objects* $\{x_1, \dots, x_n\}$, of the original space, in this case the Internet graph, onto *points* $\{v_1, \dots, v_n\}$ in a target vector space V of dimensionality k .

The concept of distortion is used to measure the quality of an embedding technique. It measures how much larger or smaller the distances in the vector space $D(v_i, v_j)$ are when compared to the corresponding distances $d(x_i, x_j)$ of the original space. The distortion is the lowest value $c_1 c_2$ that guarantees that [6]:

$$\frac{1}{c_1} \cdot d(x_i, x_j) \leq D(v_i, v_j) \leq c_2 \cdot d(x_i, x_j)$$

Different techniques have been recently proposed to embed the Internet graph into a metric space, while trying to minimize the distortion at the same time [11,13,17,7,4, 5]. The distance metric used in these techniques is the minimum round trip time (RTT) over a significantly large time interval.

3.2 Triangle Inequality on the Internet

The AS-level traffic on the Internet is forwarded based on dynamic BGP routing policies. In general, service providers are free to set their own BGP policies and make local arrangements with peering providers and customers. Moreover, service providers are often multi-homed, which means they have multiple connections to the Internet for various reasons such as links resilience by using backup links and traffic load balancing. Also, the customer-provider relationship can be regarded as one reason why shortest path routing may not be used in a given fraction of the network. There will be cases where

providers, for business-related reasons, will prefer to route traffic via another provider as opposed to a customer, even if the shortest path is through its customer.

Thus, there is no reason to expect that the triangle inequality, an important property of metric spaces, holds on the Internet. In this section we enumerate all possible triangles formed by three PlanetLab sites. We then analyze the violations of the triangle inequality, as well as a measure of how good these triangles are.

3.3 Data and Methodology

We used RTT data¹ collected between all PlanetLab hosts from November 17th to November 23rd 2003. We then calculated the minimum RTT between each pair of hosts available between those dates on consecutive 15-minute periods. Thus for each day in this period there were 96 matrices of RTT measurements (each entry represented a pairwise RTT measurement), and the size of each matrix was 261×261 . Over the seven day period we therefore had 672 such matrices.

Our goal in this evaluation was to identify the various triangle inequality violations between these pairs of hosts. Since we wanted to perform this computation over our estimate of propagation delays on the paths only (and not the queueing and congestion effects), we first constructed a single matrix, in which each entry represented the minimum RTT between a pair of PlanetLab nodes over the entire 7-day period. This avoided biases in the results due to high variations in RTTs, e.g. during congested periods. Our analysis of the data indicated that by taking the minimum over a 7-day period, we can filter out congestion related effects which have periodic weekly patterns.

Many PlanetLab sites had multiple nodes per site. For instance, the Computer Laboratory (University of Cambridge) site hosted three nodes `planetlab1`, `planetlab2` and `planetlab3` under the domain `c1.cam.ac.uk`. The minimum RTTs between nodes within a site were very small, often of the order of 1 ms.

Examining triangles between all triples of nodes was therefore wasteful and biased our results by the distribution of nodes in PlanetLab sites. Thus, we selected a representative node in each site so as to build a *site by site* matrix D' , reducing the distance matrix to 123×123 RTTs.

3.4 Preliminary Results

We defined a metric r , termed the *goodness* of a triangle, to test violations of the triangle inequality on the D' matrix. By convention a is the longest side of a triangle, and the other remaining sides are termed b, c . The metric r is defined as:

$$r = \left(\frac{a}{b + c} \right) \times (1 + (a - (b + c)))$$

This metric is made of two terms. The first one distinguishes between ‘good’ and ‘bad’ triangles (violators). Values of r less or equal to 1 represent ‘good’ triangles. In

¹ The all-pairs RTT measurement data was being continuously collected by Jeremy Stribling (MIT) and it is available from http://www.pdos.lcs.mit.edu/~strib/pl_app/

contrast, any triangle with r greater than 1 is a *violator*, i.e. the relation between its sides lengths do not satisfy the triangle inequality property. This first term is multiplied by a factor which tells the ‘goodness’ of a triangle. The higher the value of r , the worse it is the triangle. We believe that triangles with higher r have higher impact on the embedding distortion factors c_1 and c_2 . However, further research is required to quantify such a distortion.

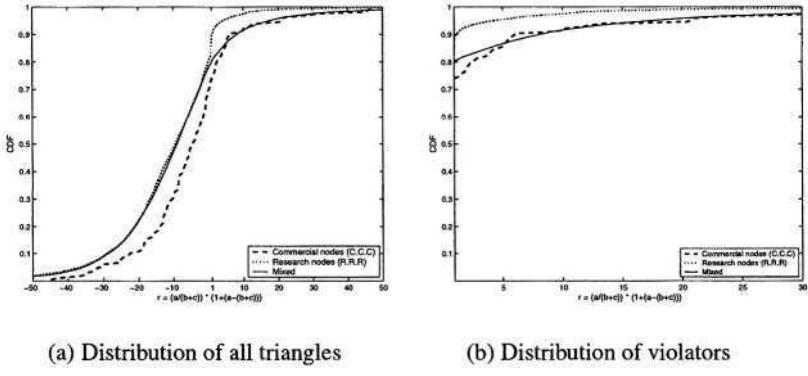


Fig. 4. CDF of the goodness metric r

In Figure 4, we plot the CDFs of metric (r) of triangles derived from the upper triangular part of D' . The three lines in each graph are: (1) RRR, which includes triangles formed only by sites in GREN, (2) CCC, which includes only those triangles that are formed by sites in the commercial Internet, and (3) Mixed, which includes all the other triangles (combination of commercial and GREN sites). Figure 4(a) shows the distributions of ‘good’ and ‘bad’ triangles; whereas Figure 4(b) presents only the cases of violators, i.e. values of r greater than 1.

We can make two interesting observations from this figure. First, there are fewer violations of the triangle inequality in the GREN than in the commercial Internet. Violators represent about 12% of the cases in the GREN and about 25% for the commercial Internet. When mixed triangles were tested, around 20% of them were violators.

And second, the triangles in the GREN are ‘better’ formed than the triangles in the commercial Internet. This means that ‘bad’ triangles in the commercial Internet tended to be larger than the ones of the GREN and also have a very long side.

At this stage, we speculate that this difference might be the reason why the distribution of violators for mixed triangles is closer to the distribution of commercial triangles (Figure 4(a)). One would expect the opposite, as the number of GREN sites forming mixed triangles are predominantly larger.

It seems that commercial sites when combined to GREN sites influenced the shape of the triangles, i.e. distorting them. The nature and cause of these differences need further detailed investigation.

4 Recommendations for Improving PlanetLab Diversity

The results presented in our case study leads to broad research questions — How can we choose a set of nodes in the Internet so that inter-node experiments can reasonably be taken to represent behavior on the Internet as a whole? In the PlanetLab context this might be reformulated as follows — How can we systematically choose the placement of new nodes so that as the PlanetLab platform grows it becomes a closer and closer approximation to the Global Internet?

One aspect of this is the distinction between commercial and research networks. One direction towards diversity might be to increase the number of sites on the commercial networks. This may be desirable, but as a general strategy it has several shortcomings. The main problem with this approach is that many research sites are buried deep inside of department/campus/regional networks and find it difficult if not impossible to get direct access to the commercial Internet. For example, the PlanetLab hosts at the computer science department in UW Madison are many router hops away from a commercial router. It obtains connectivity through a departmental network, then a campus network, and finally through a state-wide research network (WiscNet). It is only WiscNet that provides access to the commercial Internet. A solution to add diversity in PlanetLab paths might be to work with the upstream networks and have some PlanetLab net blocks announced only to the commercial networks upstream from Wiscnet. Perhaps this could be implemented with some type of BGP community sent up on the routes when they are announced into WiscNet. It probably can be done, but requires a lot of BGP magic.

We would advocate a different approach. The entire GREN is very large and comprises of a diverse set of networks. For example, CERN runs a network that is to GREN what the GREN is to the global Internet. Out of the 1123 ASes that we found in the GREN, PlanetLab nodes are located in 80, i.e. less than 8% of them. Perhaps the best strategy (in terms of cost and feasibility) is to make a targeted and systematic attempt to exploit the existing diversity of the GREN. Our exploration and research of the characteristics of GREN lead us to believe that there are a lot of diverse networks within GREN itself. For example, While some research divisions in academic environments may have relatively low bandwidth connectivity to the Internet, other departments and entities, e.g. some astronomy and high-energy physics communities, have very high bandwidth connectivity and we should target this diversity of groups and communities outside of Computer Science to add on as PlanetLab sites. We should also aggressively consider adding further geographic diversity by recruiting more sites in Latin America, Asia, Australia, Russia and so on. The connectivity characteristics of such diverse locations would certainly enhance the diversity on the PlanetLab. One possible approach to do this might be to examine various routing and topological data that are available from sites like Caida (<http://www.caida.org>), e.g. Skitter data, and identify specific sites within GREN that will expand PlanetLab diversity.

Focusing on the GREN for diversity has additional advantages. In general, the GREN is more *transparent*—there is more publicly available information about the connectivity of the GREN and more willingness to share information with the research community. Hence if we can find suitable diversity just within GREN, such an enhancement is certainly worth exploring further as we attempt to establish greater diversity within PlanetLab.

Acknowledgements. We would like to thank the following people whose comments on this work were extremely helpful — Randy Bush, Jon Crowcroft, Christophe Diot, Steven Hand, Gianluca Iannaccone, Z. Morley Mao, Andrew Moore, David Moore, Jennifer Rexford, Timothy Roscoe, Larry Peterson, James Scott, Colleen Shannon, and Richard Sharp. Thanks to Jose Augusto Suruagy Monteiro of UNIFACS, Sidney Lucena and Ari Frazao Jr. of RNP Brazil for helping us better understand the connectivity of the Brazilian research networks. We would like to thank Jeremey Stribling for collecting the PlanetLab round trip data and making it publicly available. In addition, this work would not be possible without the existence of public archives of interdomain routing data including Route Views and RIPE.

References

1. PlanetLab home page, <http://www.planetlab.org>.
2. Scriptroute Home Page, <http://www.scriptroute.org>.
3. University of Oregon Route Views Archive Project, <http://www.routeviews.org>.
4. M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet Coordinates for Distance Estimation. In *24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, Tokyo, Japan, March 2004.
5. R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris. Practical, Distributed Network Coordinates. In *ACM Workshop on Hot Topics in Networks (HotNets-II)*, November 2003.
6. G. Hjaltason and H. Samet. Properties of Embedding Methods for Similarity Searching in Metric Spaces. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(5), May 2003.
7. H. Lim, J. Hou, and C. Choi. Constructing Internet Coordinate System Based on Delay Measurement. In *ACM SIGCOMM Internet Measurement Conference (IMC'03)*, Miami (FL), USA, October 2003.
8. Z. Morley Mao, David Johnson, Jennefer Rexford, Jia Wang, and Rany Katz. Scalable and Accurate Identification of AS-Level Forwarding Paths. In *INFOCOM '04*, 2004.
9. Z. Morley Mao, Jennefer Rexford, Jia Wang, and Rany Katz. Towards an Accurate AS-level Traceroute Tool. In *ACM SIGCOMM Technical Conference '03*, August 2003.
10. Ripe NCC. Routing Information Service Raw Data. <http://abcoude.ripe.net/ris/rawdata>.
11. T.S. Eugene Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *IEEE INFOCOM'02*, New York, USA, June 2002.
12. L. Peterson, T. Anderson, and D. Culler. A Blueprint for Introducing Disruptive Technology into the Internet. In *ACM Workshop on Hot Topics in Networks (HotNets-I)*, October 2002.
13. M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for Scalable Distributed Location. In *2nd International Workshop on Peer-to-Peer Systems*, February 2003.
14. D. Spence. An implementation of a Coordinate based Location System. Technical Report UCAM-CL-TR-576, University of Cambridge, November 2003. Available at <http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-576.pdf>.
15. C. Stover and M. Stanton. Integrating Latin American and European Research and Education Networks through the ALICE project. Document available at <http://www.dante.net/upload/doc/AlicePaper.doc>.
16. M. Szymaniak, G. Pierre, and M. van Steen. Scalable Cooperative Latency Estimation. Submitted for publication. Draft available at http://www.globule.org/publi/SCOLE_draft.html, December 2003.
17. L. Tang and M. Crovella. Virtual Landmarks for the Internet. In *ACM SIGCOMM Internet Measurement Conference (IMC'03)*, Miami (FL), USA, October 2003.

Searching for Self-similarity in GPRS

Roger Kalden and Sami Ibrahim

Ericsson Research, Ericsson Eurolab Deutschland GmbH, Aachen, Germany

Roger.Kalden@ericsson.com,

phone +49 2407 575 7831

Abstract. Based on measurements in live GPRS networks, the degree of self-similarity for the aggregated WAP and WEB traffic is investigated by utilizing six well established Hurst parameter estimators. We show that in particular WAP traffic is long-range dependent and its scaling for time scales *below* the average page duration is not second order self similar. WAP over UDP can also determine the overall traffic scaling, if it is the majority traffic. Finally we observe that the minor traffic exhibits a larger Hurst value than the aggregated traffic, in case of WAP as well as in case of WEB traffic.

1 Introduction

Based on live GPRS traffic measurements, we investigate the packet arrival process and the data volume arrival process of WAP and WEB traffic on statistical self-similarity.

Many studies have been looking at various network types and found evidence for self-similarity (e.g., [1],[2],[3]). This property is regarded as an invariant of network traffic and has serious performance implications. In the case of self-similar traffic the applied statistics for performance analysis and network dimensioning are different from those when applied to statistically more simple traffic, which can be modeled with Markovian processes ([4],[5]). For instance the queue tail behavior is heavy-tailed in the case of self-similar input traffic [6]. This leads to heavy-tailed packet delay distributions, which can influence the TCP round-trip time estimations. Furthermore, the traffic does not smooth out in the case of aggregation, leading to congestion situations and packet-drops due to the burstiness of the traffic. Consequently, it is important to understand the self-similar nature of the traffic in a network in order to apply the right statistical methods for performance investigations and network dimensioning. In previous studies the reason for self-similarity has been identified as the heavy-tailedness of many statistical properties of Internet traffic, on the user and application level. In [7] and [18] the authors showed that heavy-tailed sessions and file size lengths lead to self-similarity for large aggregation scales (long-range dependency). The authors in [8] and [14] have furthermore shown that self-similarity (in particular the multiscaling) behavior at small timescales (e.g., smaller than the average round-trip time) is due to protocol interactions of TCP.

But GPRS traffic and in general cellular access network traffic has not yet been investigated on its self-similarity property. GPRS is not merely a new access technolo-

gy, it also introduces novel applications such as WAP and MMS, and provides Internet access in a mobile or nomadic usage environment. This yields a special traffic composition, different from wireline Internet traffic. We have investigated GPRS networks with more than 60% of the traffic volume consisting of UDP traffic. This is in sharp contrast to the usual 80% of TCP traffic in the fixed Internet (cf. [9] for earlier figures on GPRS traffic). Additionally, WAP and MMS file sizes are in general much shorter than WEB and FTP file sizes [17]. For these reasons we counter-check the property of self-similarity in GPRS. Based on our results, which show that GPRS and in particular WAP traffic, is asymptotically second order self-similar (long-range dependent), we propose for further research to study the scaling nature of GPRS traffic and to explore in particular the reasons for this for WAP traffic.

The remainder of the paper is structured as follows. Firstly, we give a brief overview of self-similarity together with the commonly used methods to test self-similarity. Next, we describe our measurement set-up and the investigated traces. In our main section we present the results of the Hurst parameter estimation methods to assess the degree of self-similarity in GPRS. Finally, in the concluding section we list open issues and future work items.

2 Self-similarity

Self-similarity, in a strict sense, means that the statistical properties (e.g., all moments) of a stochastic process do not change for all aggregation levels of the stochastic process. That is, the stochastic process “looks the same” if one zooms in time “in and out” in the process. Let $X = (X_t : t = 0, 1, 2, \dots)$ be a covariance-stationary stochastic process, with constant and finite mean and finite variance and autocorrelation function $\rho(k)$, which only depends on k . Further, let $X^{(m)} = (X_{k(m)} : k = 1, 2, 3, \dots)$ be a new time series generated by averaging the original time series X over non-overlapping blocks of size m . That is, for each $m = 1, 2, 3, \dots$ the series is given by $X_{k(m)} = 1/m(X_{km-m+1} + \dots + X_{km})$, $k = 1, 2, 3, \dots$ and $\rho^{(m)}(k)$ the corresponding autocorrelation function. The stochastic process $X = (X_t : t = 0, 1, 2, \dots)$ is then called *exactly second-order self-similar* with self-similarity parameter $H = 1 - \beta/2$, if the autocorrelation functions $\rho^{(m)}(k)$ of the processes $X^{(m)}$ are all equal to the autocorrelation function $\rho(k)$ of the original process X . That is: $\rho^{(m)}(k) = \rho(k)$, $k = 1, 2, 3, \dots$ for all $m = 1, 2, 3, \dots$ X is called *asymptotically second-order self-similar* with self-similarity parameter $H = 1 - \beta/2$ if the correlation structure of the aggregated time series $X^{(m)}$ is indistinguishable from the correlation structure of the original time series X as $m \rightarrow \infty$. That is, the above holds asymptotically for large aggregation levels.

H expresses the degree of self-similarity; large values indicate stronger self-similarity. If $H \in (0.5, 1)$ X is called long-range dependent (LRD). Both, exactly and asymptotically second-order self-similar traffic can include long-range dependency,

however, as it is often the used case, we will use long-range dependent for asymptotically second-order self-similar traffic.

Asymptotical second-order self-similar processes have a few interesting characteristics. For instance, the autocorrelation function of an LRD process is decaying hyperbolically. That is (with β the same as in H, above):

$$\lim_{k \rightarrow \infty} \rho(k) \sim |k|^{-\beta} \quad (0 < \beta < 1).$$

Also the variance of the aggregated time series is very slowly decaying. That is:

$$\text{Var}[X^{(m)}] \sim m^{-\beta} \quad (0 < \beta < 1).$$

Another property is that the power spectrum is singular as the frequency is approaching 0. That is:

$$S(w) \underset{w \rightarrow 0}{\sim} 1/|w|^{1-\beta} \quad (0 < \beta < 1).$$

3 Estimation Methods for Self-similarity

Various methods for estimating the Hurst parameter H exist for deducing self-similarity or long-range dependency [15]. The estimation methods can be grouped into time-based and frequency-based methods. We will briefly provide an overview of the methods used to estimate the value of the Hurst parameter.

The first four methods are time-based:

R/S method

This method is based on empirical observations by Hurst and estimates H are based on the R/S statistic. It indicates (asymptotically) second-order self-similarity. H is roughly estimated through the slope of the linear line in a log-log plot, depicting the R/S statistics over the number of points of the aggregated series.

Variance Method

The Variance Method is based on the slowly decaying variance property as stated above. It indicates long-range dependency. The slope β of the straight line in a log-log plot, depicting the sample variance over the block size of each aggregation, is used for roughly estimating H. H is given by $H = 1 - \beta/2$.

Absolute Moment Method

This method is related to the variance method computed for the first moment. The slope α of the straight line in a log-log plot, depicting the first moment of the aggregated block over the block size, provides an estimator for H, by $H = 1 + \alpha$.

Ratio of Variance of Residuals

This uses the empirical observation, that is, the sample variance of residuals, plotted over the aggregation level, yields a slope equivalent to roughly $2H$. It indicates some self-similarity.

The next two methods are frequency-based:

Periodogram method

This method is based on the power-spectrum singularity at 0-property as stated above. The slope of the straight line, approximating the logarithm of the spectral density over the frequency as the frequency approaches 0, yields H.

Abry-Veitch method

This method is based on the multi-resolution analysis and the discrete wavelet transformation. H is estimated by fitting a straight line to the energy in the series over octave j (expressing the scaling level in the time and the frequency domain) in a log-log plot.¹ This method is the most comprehensive and robust method for determining the scaling behavior of traffic traces. Its strength follows from the fact that the multi-resolution analysis itself has a structural affinity to the scaling process under study. That is, multi-resolution analysis itself exploits scaling, but transfers the complex scaling process to a much simpler wavelet domain, in which short range dependent (SRD) statistics can be applied to infer answers on the scaling of the process [10].

We show results from our traces for all mentioned methods. In particular we will use the Abry-Veitch method to derive the scaling nature of the process.

All used methods provide some intermediate statistics, which is used to derive the Hurst value. For instance, in the case of the Variance Method, these are the aggregated variance values for each aggregation level; or, in the case of the Abry-Veitch method, the intermediate statistics used are the wavelet coefficients. Based on those values linear regression is used to fit a straight line to derive the Hurst value.

Important to consider is that typically the linear regression should not consider all of the values from the intermediate statistic. In case of the R/S method, the Variance Method and the Absolute Moment method, it is recommended not to use the results of the first few aggregations levels and neither the last few aggregation levels. The reason for this is that these values are not very reliable because either the aggregation level is too low (sampling too few points per block) or it is too high (sampling all points in just a few blocks). In the case of the Periodogram Method it is recommended only to use approximately the first 10 percent of the results, close to the frequency 0. This is justified by the asymptotic LRD property close to the frequency 0. The Abry-Veitch method is the most robust of all estimation methods [10]. It actually shows the scaling of the process over all aggregation levels (octave j), which allows to optimally select an appropriate starting point for the regression. This starting point is indicated by a χ^2 -goodness-of-fit test. In the case of assumed LRD traffic, the regression line is fitted from this starting point to the largest available octave in the data.

We use the SELFIS tool [11] to derive the intermediate statistics for all but the Abry-Veitch method. Additionally, we use the LDestimate-code which implements the Abry-Veitch method [12]. In the case of the SELFIS tool we applied our own linear regression on the intermediate results obtained from SELFIS, as explained above. This is necessary as SELFIS estimates H , always based on a linear regression over all available points, which can heavily bias the results due to outliers at the end points. For this reason we shortened the intermediate statistical results to all but the first 2 and the last 2 aggregation levels. In all cases, applying the manual regression, the fit is better than the SELFIS tool directly provides. The differences in the values of H , between manually applied linear regression and the final results of SELFIS, are

¹ More precisely, based on the wavelet coefficient $d_{j,k}$ the amount of energy $|d_{j,k}|^2$ in the signal at about the time $t=2^j k$ and about the frequency $2^j \lambda$ is measured. $E[|d_{j,k}|^2]$ is the amount of energy at octave j . If the initial resolution of the time series is t_0 (bin size), the time resolution at each scaling level j is $t_j = 2^j t_0$.

sometimes quite large, which stresses the importance to apply manual post-processing. The LD estimator function for the Abry-Veitch test suggests an optimal starting point for j which we always used. It furthermore plots the scaling behavior over all octaves j , allowing to judge on the type of scaling. We discuss those results as well.

4 Data Traces

In cooperation with Vodafone we conducted measurements in two live GPRS networks. We captured the IP packet headers at the Gi interface for all users in a geographical region of the operator's GPRS network. The Gi interface connects the mobile network to external packet switched networks (like the Internet, a corporate Intranet or Email and WAP/MMS proxies). All IP packets from or to mobile terminals traverse this interface. The traces we focus on were taken during summer 2003.

We collect for every packet crossing the network monitor interface a time stamp, with $1\mu\text{s}$ accuracy, with the total length of the packet in bytes. For our further investigations we generate from the original trace the Packet Arrival Process (PAP) as a discrete time-series process by counting the number of packets and the Data Volume Process (DVP) as the total number of bytes within a time interval (bin) or 100 ms.

We are interested in the scaling behavior for the aggregated traffic, WAP oriented traffic and WEB oriented traffic. For this purpose we look at three “sup-“sampled traces. Firstly, we investigate the total aggregated traffic (up and downlink traffic), which we measured on the Gi interface. Next, we have split-up the traffic into WEB oriented traffic and WAP oriented traffic. We do this by splitting the data according to the APN (Access Point Names) they are belonging to. GPRS allows users to attach to different APNs provided by the operator. Typically, the APNs are used for different types of traffic and split logically the traffic on the Gi interface. We checked our measurements for the used applications and found that most of the traffic on one APN consists of WEB like applications, including HTTP, FTP, Email, etc. On the other APN we see mostly WAP like traffic, comprising WAP and MMS.² However, the APNs do not filter for applications, hence it occasionally happens that we see WEB traffic on the WAP APN and vice versa.

On one of the network measurement points (Vfe1) the traffic splits up into 25% WEB-APN and 75% WAP APN traffic, while in the other network (Vfe2) the split is 70% WEB APN and 30% WAP APN traffic.

We traced several 24-hour periods and investigated appropriate busy-hour periods spanning several hours, for each of the networks. Our results will be presented only for one selected day for each network. In the case of Vfe1 we chose a 110-minute busy-hour period in the afternoon, in the case of Vfe2 we chose a 430-min (7-hour) busy-hour period covering the whole afternoon. Since all estimator methods (with exception of the Abry-Veitch method) require stationarity and often are very sensitive to underlying trends or correlations in the traffic process [13], we investigated the chosen busy-hour periods on trends by plotting the moving average, on periodicity by in-

² We used the same investigation method we have used in [9] for differentiating the applications.

vestigating the Fourier transformation, and on stationarity by applying the average run test. All tests indicated the suitability of the chosen periods for the estimation methods, i.e., they appear to be stationary and they have no visible periodicities or trends.

5 Results on Self-similarity

We present for all processes detailed results obtained by the Abry-Veitch method, together with Hurst estimations acquired by the other described methods. Table 1 and Table 2 show the results for the estimated Hurst values by the Abry-Veitch method for the packet arrival process. The second row ‘H’ contains the Hurst values and in the third row marked ‘conf.’ the confidence values for the estimated Hurst values are listed. Figure 1 and Figure 2 illustrate the Hurst values obtained from the different methods, for comparison. All values are very similar with Hurst values of about 0.8 and higher. This strongly suggests long range dependency for the PAP for both networks Vfe1 and Vfe2.

Table 1. Results for PAP in Vfe1

Vfe1	Agg	WEB	WAP
H	0.86	0.83	1.06
Conf.	[0.76,0.95]	[0.74,0.92]	[0.96,1.14]
Scaling	Fig. 8	Fig. 8	Fig. 7

Table 2. Results for PAP in Vfe2

Vfe2	Agg	WEB	WAP
H	0.90	1.02	0.89
Conf.	[0.81,0.98]	[0.93,1.11]	[0.79,0.97]
Scaling	Fig. 5	Fig. 5	Fig. 5

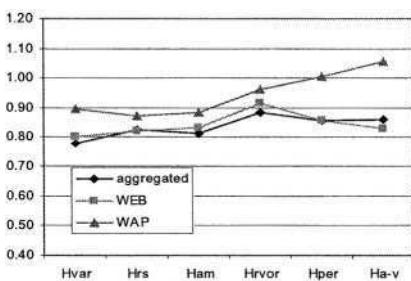


Fig. 1. All Hurst values for PAP and Vfe1

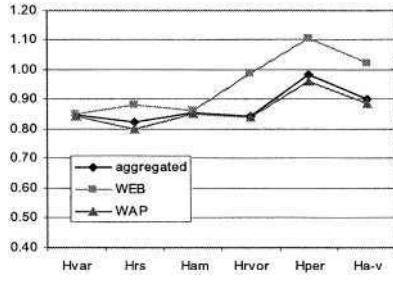


Fig. 2. All Hurst values for PAP and Vfe2

‘Hvar’ stands for Variance Method, ‘Hrs’ for R/S Method, ‘Ham’ for Absolute Moment Method, ‘Hrvor’ for Variance of Residuals, ‘Hper’ for Periodogram Method and ‘Ha-v’ for Abry-Veitch Method

In some cases the Hurst value is above 1, actually precluding LRD, but in the case of the Abry-Veitch method all confidence intervals include also a value of below 1. Furthermore, as explained next, inspection of the output-plots suggests in such cases still asymptotic self-similarity.

For the DVP, Table 3 and Figure 3 list the Hurst values for Vfe1 and Table 3 and Figure 4 list the Hurst values for Vfe2. Again all results strongly indicate LRD.

All estimation methods, except for the Abry-Veitch method, assume an LRD model beforehand. That is, the Hurst estimation value can only be regarded as correct if the assumption of an LRD process holds. In contrast, the Abry-Veitch test is not based on such assumptions. It shows the scaling of the process for all time scales in the diagram. Only by interpreting the results in this Logscale Diagram, the true nature of the process is determined (e.g., self-similarity, long-range dependency, multiscaling) [10]. For our traces we have encountered four basic log-scale diagram types, depicted in Figure 5 to Figure 8. In Table 1 to Table 4 we list in the 4th row for each process the plot that comes closest, respectively. The individual plots looked very similar to the exemplary plots, but with different scales on the y-axes.

Table 3. Results of DVP for Vfe1

Vfe1	Agg	WEB	WAP
H	0.69	0.68	0.92
Conf.	[0.65,0.72]	[0.64,0.71]	[0.88,0.96]
Scaling	Fig. 6	Fig. 6	Fig. 7

Table 4. Results of DVP for Vfe2

Vfe2	Agg	WEB	WAP
H	0.82	1.07	0.81
Conf.	[0.73,0.90]	[0.98,1.15]	[0.72,0.89]
Scaling	Fig 5	Fig. 7	Fig. 5

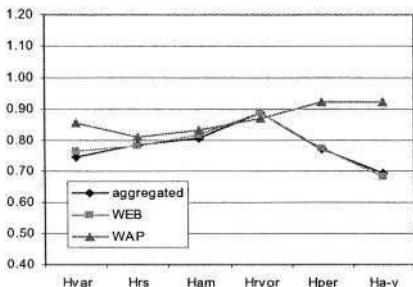


Fig. 3. All Hurst values for DVP and Vfe1

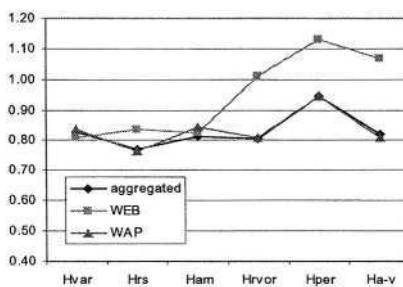


Fig. 4. All Hurst values for DVP and Vfe2

'Hvar' stands for Variance Method, 'Hrs' for R/S Method, 'Ham' for Absolute Moment Method, 'Hrvor' for Variance of Residuals, 'Hper' for Periodogram Method and 'Ha-v' for Abry- Veitch Method

Figure 5 and Figure 7 both show a typical plot for LRD traffic. The second-order scaling starts from a certain scaling point on and continues until the largest available scale in the trace. For small scales we do not see second-order scaling behavior. We have found this scaling behavior for all WAP processes.

Figure 6 also exhibits LRD scaling, but actually has two scaling regions. One from approximately 1 to 8 and one from 8 to maximum scale. This is called bi-scaling. We have found this in the case of DVP for Web traffic in Vfe2. Figure 8 depicts the case where the process has second-order scaling over all scales. This indicates strictly second-order self-similarity. We see this in the case of PAP for WEB traffic also in Vfe2.

We point out some interesting observations (cf. Table 1 to Table 4). Firstly, the Hurst value of the aggregated traffic is always very close to the Hurst value of the majority of the traffic. This is in agreement with [14]. In the case of Vfe1 the major part of the traffic is WAP traffic, in the case of Vfe2 it is WEB traffic. More in detail, even the whole scaling behavior, as depicted by the Logscale Diagram, is very similar between the majority traffic and the aggregated traffic. This implies, by knowing the scaling of the majority traffic, one obtains also the scaling of the aggregated traffic. Secondly, the minor traffic has always slightly higher Hurst values, with changing roles of WAP and WEB in the cases of Vfe1 and Vfe2. We do not have an explanation for this, yet. One reason might be that even so we applied the estimation methods on separated traces per APN, it is not possible to separate them truly: they have been both traveling together through the GPRS network, thereby most likely affecting each other.

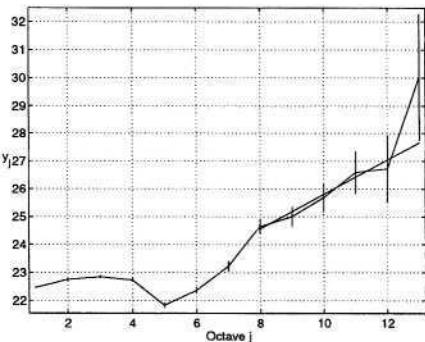


Fig. 5. Typical plot for processes showing long range dependency. Below a certain scale no regular linear scaling exists. The linear part is at $j=8$ divided in two scaling regions

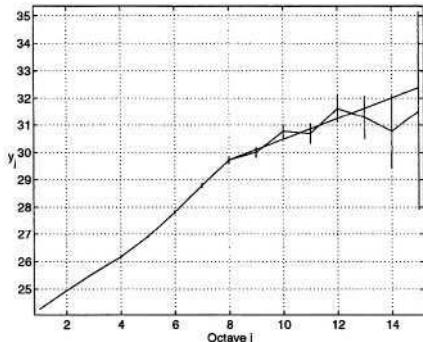


Fig. 6. Typical plot for processes showing biscaling. The second scaling region starts at $j=8$. It also implies long range dependence

An interesting question arises whether it is actually possible to truly identify the Hurst value for each type of traffic separately. In [14] the authors have shown that non-self-similar UDP traffic is affected by self-similar TCP traffic. But this effect is only strong if the self-similar traffic is the major traffic. In our case we observe high Hurst values even in the case the WAP (UDP) traffic is dominating. This suggests that WAP traffic itself is strongly long-range dependent.

Furthermore, we see that WAP traffic has a very different scaling behavior for small scales compared to WEB traffic. The reason for the different small scaling behavior can be assumed to be the very different transport mechanism of TCP and WAP over UDP. As we explained in the footnote to the Abry-Veitch method, the octave j in the logscale diagram also expresses the timescale of the aggregated process. Based on this insight it is possible to determine over which timescales scaling occurs. We start with an initial bin size of 100 ms, which leads to values of $t_j = 0.1, 0.2, 0.4, 0.8, 1.6, 3.2, \dots$ for $j=1, 2, 3, 4, 5, \dots$, respectively. For all processes in which we observed a scaling like the one depicted in Figure 5 and Figure 7, the knee-point at which the linear scaling starts is at about $j=5$ or $j=6$, i.e., respectively 1.6 seconds or 3.2 seconds. In

[16] the authors show that the average download time per WAP page, including embedded objects, is in the order of 1.5-3 seconds. Hence, this timescale marks the demarcation line between the WAP transport layer protocol and the user behavior.

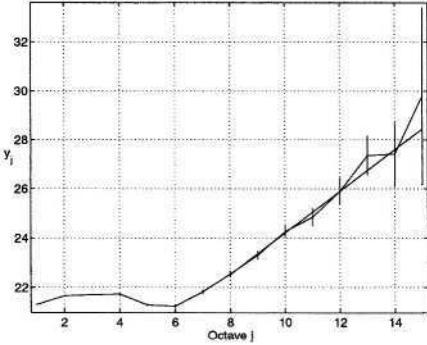


Fig. 7. Typical plot for processes showing long-range dependency. Below a certain scale no regular linear scaling exists

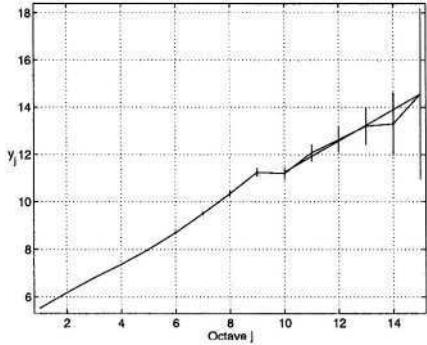


Fig. 8. Linear scaling over the whole range. Although there is a step at $j=9$, the slope on both sides is almost the same. This indicates second-order self-similarity

6 Conclusion

Based on live GPRS measurements, we applied 6 different established Hurst estimation methods, including the comprehensive and robust Abry-Veitch test on packet arrival and data volume processes. We showed the Hurst value as well as the scaling behavior for the busy-hour for 3 different traffic types and two different networks. In the case of aggregated traffic and also in the case of individual WAP and WEB traffic traces, the results strongly suggest long-range dependency.

We confirmed that the dominant traffic type (WAP or WEB) determines the degree of self-similarity of the aggregated traffic. In our case however, the minor traffic always exhibits a Hurst value larger than the Hurst value of the major traffic. This is particularly interesting as WAP traffic is based on UDP.

We showed that WAP traffic has a very different scaling behavior compared to WEB traffic for small scales. We identified the demarcation line between small and larger scales for WAP to coincide with the average page duration. Larger scales can probably be accounted to user behavior.

As future research we wish to investigate the reason(s) why the minor traffic fraction exhibits higher Hurst parameter values for all tests.

Further research also includes investigating the reason(s) of self-similarity of WAP traffic and the exact nature of scaling; whether it is multifractional, monofractional, strictly self-similar or ‘just’ long-range dependent.

Acknowledgements. We thank Vodafone for providing us with live GPRS traces from various networks in Europe. We also thank Thomas Karagiannis, Michalis Faloutsos, the authors of the SELFIS tool, and Darryl Veitch, the author of the LD estimator-code, for making their software available on the Internet.

References

- [1] W. Willinger, M. S. Taqqu, and A. Erramilli, "A bibliographical guide to self-similar traffic and performance modeling for modern high-speed networks," Stochastic Networks: Theory and Applications. In Royal Statistical Society Lecture Notes Series, Oxford University Press, 1996, vol. 4, pp.339–366.
- [2] V. Paxson and S. Floyd, "Wide Area Traffic: The Failure of Poisson Modeling," Proceedings of ACM SIGCOMM'94, 1994.
- [3] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)," IEEE/ACM Transactions on Networking, 1994, vol.2 no.1, pp.1-15.
- [4] A. Erramilli, O. Narayan, and W. Willinger, "Experimental queueing analysis with long-range dependent packet traffic," IEEE/ACM Transactions on Networking, New York, NY, Apr. 1996, vol.4, pp.209–223.
- [5] A. Erramilli, O. Narayan, A. L. Neidhardt, and I. Saniee, "Performance impacts of multi-scaling in wide-area TCP/IP traffic," Proc., IEEE INFOCOM 2000, Tel Aviv, Israel, 2000, vol.1, pp.352–359.
- [6] I. Norros, "A storage model with self-similar input," Queueing Systems, 1994, vol.16, pp.387–396
- [7] M.E. Crovella and A. Bestavros A., "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," Proceedings of ACM SIGMETRICS '96, 1996.
- [8] A. Feldmann, A.C. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control," ACM SIGCOMM '99, 1999.
- [9] R. Kalden, T. Varga, B. Wouters, B. Sanders, "Wireless Service Usage and Traffic Characteristics in GPRS networks", Proceedings of the 18th International Teletraffic Congress (ITC18), Berlin, 2003, vol.2, pp.981-990.
- [10] P. Abry, Flandrin, M.S. Taqqu, D. Veitch, "Wavelets for the analysis, estimation and synthesis of scaling data." In: K. Park and W. Willinger, Eds., "Self Similar Network Traffic Analysis and Performance Evaluation," Wiley, 2000.
- [11] T. Karagiannis, M. Faloutsos, "SELFIS: A Tool For Self-Similarity and Long-Range Dependence Analysis," 1st Workshop on Fractals and Self-Similarity in Data Mining: Issues and Approaches (in KDD), Edmonton, Canada, July 23, 2002.
- [12] Darryl Veitch's home page: <http://www.cubinlab.ee.mu.oz.au/~darryl/>
- [13] T. Karagiannis, M. Faloutsos, "Long-Range Dependence: Now you see it, now you don't" CSE Dept, UC Riverside, Rudolff H. Riedi, ECE Dept, Rice University. In: IEEE Global Internet, 2002.
- [14] K. Park, G.T. Kim, and M.E. Crovella, "On the Relationship Between File Sizes, Transport Protocols, and Self-Similar Network Traffic," Proceedings of IEEE International Conference on Network Protocols, 1996, pp 171-180.
- [15] A. Popescu, "Traffic Self-Similarity", ICT2001, Bucharest, 2001.
- [16] Internal communication, Ericsson ETH, Budapest, 2003
- [17] R. Kalden, H. Ekström, "Searching for Mobile Mice and Elephants in GPRS Networks," submitted for publication, 2004.
- [18] W. Willinger, V. Paxson, M. Taqqu, "Self-Similarity and Heavy Tails: Structural Modeling of Network Traffic", in "A practical Guide To Heavy Tails: Statistical Techniques and Applications", Birkhauser, Boston, 1998

The Effect of Flow Capacities on the Burstiness of Aggregated Traffic*

Hao Jiang and Constantinos Dovrolis

College of Computing, Georgia Tech
`{hjiang, dovrolis}@cc.gatech.edu`

Abstract. Several research efforts have recently focused on the burstiness of Internet traffic in short, typically sub-second, time scales. Some traces reveal a rich correlation structure in those scales, while others indicate uncorrelated and almost exponential interarrivals [1]. What makes the Internet traffic bursty in some links and much smoother in others? The answer is probably long and complicated, as burstiness in short scales can be caused by a number of different application, transport, and network mechanisms. In this note, we contribute to the answer of the previous question by identifying one generating factor for traffic burstiness in short scales: *high-capacity flows*. Such flows are able to inject large amounts of data to the network at a high rate. To identify high-capacity flows in a network trace, we have designed a passive capacity estimation methodology based on packet pairs sent by TCP flows. The methodology has been validated with active capacity measurements, and it can estimate the *pre-trace capacity* of a flow for about 80% of the TCP bytes in the traces we analyzed. Applying this methodology to Internet traces reveals that, if a trace includes a significant amount of traffic from high-capacity flows, then the trace exhibits strong correlations and burstiness in short time scales.

1 Introduction

The (layer-3) capacity of a network link is defined as the maximum IP-layer throughput that that link can deliver [2]. The capacity of a network path, C , is defined as the minimum capacity of the links that constitute that path. Consider a packet trace \mathcal{T} collected at a network link \mathcal{L}_T (“vantage point”). Suppose that f is a TCP flow in \mathcal{T} , and that \mathcal{S}_f is the flow’s source. The capacity of the path between \mathcal{S}_f and \mathcal{L}_T is referred to as the *pre-trace capacity* C_p of flow f . Notice that $C_p \geq C$, and so the pre-trace capacity can be viewed as an upper-bound,

* This work was supported by the “Scientific Discovery through Advanced Computing” program of the US Department of Energy (award number: DE-FG02-02ER25517), by the “Strategic Technologies for the Internet” program of the US National Science Foundation (award number: 0230841), and by an equipment donation from Intel. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the previous funding sources.

potentially tight, of the path capacity. This is important, especially when the latter cannot be estimated accurately from passive measurements at the vantage point.

Our primary objective in this paper is to examine the effect of flow capacities on the burstiness of aggregated Internet traffic. To do so, however, we first need to estimate C , or at least C_p , for the flows that constitute \mathcal{T} . There are several path capacity estimation techniques and tools, such as [3,4,5], but they are based on active measurements. With passive measurements, on the other hand, we are only given a packet trace from a network link. Two passive approaches to estimate the capacity of TCP flows from end-host traces are Paxson's PBM methodology [6] and Lai's *nettimer* [7]. The problem that we consider is different, however, because we estimate the capacity of a TCP flow from a uni-directional trace collected at a vantage point in the network, rather than at the sender or receiver of the flow.

The dispersion of two successive packets of the same flow at a network link is the time spacing (interarrival) between the last bit of those packets. Our passive capacity estimation technique is based on packet pair dispersion analysis [4]. However, it differs significantly from the technique presented in [4] in two major ways. First, active capacity probing always sends back-to-back packet pairs, with the two packets of each pair having equal size. In passive capacity estimation, we do not know whether two successive packets were sent back-to-back, and they may not have the same size. Second, in passive capacity estimation we need to differentiate between the end-to-end capacity and the pre-trace capacity. As will be explained in the next section, both capacities may be visible in the distribution of packet pair dispersions of a TCP flow.

The paper structure is as follows. The pre-trace capacity estimation methodology is given in Section II. That methodology has been validated with active measurements, as summarized in Section III. Section IV presents measurements of pre-trace capacity distributions from various traces. The connection between flow capacities and traffic burstiness is shown in Section V.

2 Pre-trace Capacity Estimation

Our pre-trace capacity estimation methodology is applicable only to *TCP data flows*. We expect that a TCP data flow will include several packet pairs, meaning two successive packets sent back-to-back, due to the delayed-ACK algorithm. Based on that algorithm, a TCP receiver should typically acknowledge every second packet, and so the sender responds to every ACK with at least two back-to-back packets (as long as it has data to send).

Consider a TCP flow with pre-trace capacity C_p and path capacity C . In the following, we illustrate that both capacities would be measurable from the dispersion of successive TCP packets, when there is no cross traffic. In the presence of cross traffic, however, it may not be possible to estimate C . To simplify the following example, we assume that the dispersion of ACKs is not affected by

queueing in the reverse path, and that the sender and receiver do not introduce delays in the transmission of data packets and ACKs.

In Figure 1(a) and 1(b), we show the sequence of successive data packets ($\dots, D_k, D'_k, D_{k+1}, D'_{k+1}, \dots$), as well as the corresponding ACKs ($\dots, A_k, A_{k+1}, \dots$), assuming that the dispersion of the TCP flow's packets is not affected by cross traffic. In round i , the sender S sends a window of W_i packets of size L back-to-back. These packets arrive at the receiver R with a dispersion L/C . The receiver responds to every second packet, and so the ACK dispersion is $2L/C$. Upon receiving the first ACK of round i , the sender starts the next round $i + 1$. For each new ACK received, the sender replies with two more back-to-back packets, plus any additional packets due to window increases. If $C_p=C$, the dispersion of successive packets at the vantage point is L/C , as shown in Figure 1(a). If $C_p > C$, the dispersion between packets D_k and D'_k is L/C_p , while the dispersion between packets D'_k and D_{k+1} is $\frac{2L}{C} - \frac{L}{C_p} > L/C_p$. So, within a single round, and if there is no queueing due to cross traffic, the dispersion of successive packets at the vantage point is directly determined by either C_p , or by C and C_p . In that case, it would be relatively simple to estimate both capacities from the location of the two major modes in the distribution of dispersion measurements.

In practice, however, the dispersion of TCP data packets is often affected by cross traffic queueing. Furthermore, increased dispersions in round i can also affect the dispersions in round $i + 1$. Figure 1(c) illustrates this scenario. Cross traffic is introduced at the narrow link in round i , increasing the dispersion between two successive packets to a value X that is unrelated to C and C_p . The dispersion X can be propagated to round $i + 1$, even if there is no cross traffic queueing in that round. On the other hand, even with cross traffic queueing, every new ACK at the sender still triggers the transmission of a back-to-back packet pair. So, we expect that about 50% of the data packets are sent back-to-back, and so their dispersion at the vantage point is independent of previous rounds. The dispersion of packets triggered by different ACKs, however, is more susceptible to cross traffic queueing, because those dispersions are correlated from round to round. Consequently, we expect that the analysis of a TCP trace will provide a strong mode at the dispersion L/C_p , even in the presence of cross traffic queueing, but it may not create a strong mode at the dispersion $\frac{2L}{C} - \frac{L}{C_p}$. This explains why we focus on the estimation of the pre-trace capacity C_p , rather than on the end-to-end capacity C . In the following, when we refer to “capacity estimation” we mean pre-trace capacity estimation.

Figure 2 shows the distribution of packet interarrivals for two bulk TCP flows. The interarrivals are normalized by $L/100\text{Mbps}$, where L is the size of the second packet, and then rounded to the nearest integer. Figure 2(a) represents the case $C_p=C$, with $C \approx 100\text{Mbps}$. Note that about 90% of the interarrivals are concentrated in a single mode around the dispersion that corresponds to the capacity. In Figure 2(b), on the other hand, there are two distinct modes. The first represents about 50% of the interarrivals and it corresponds to $C_p \approx 100\text{Mbps}$. The second mode represents about 40% of the interarrivals, and it probably corresponds to the capacity $C \approx 1.3\text{Mbps}$. In practice, it is often the case that even

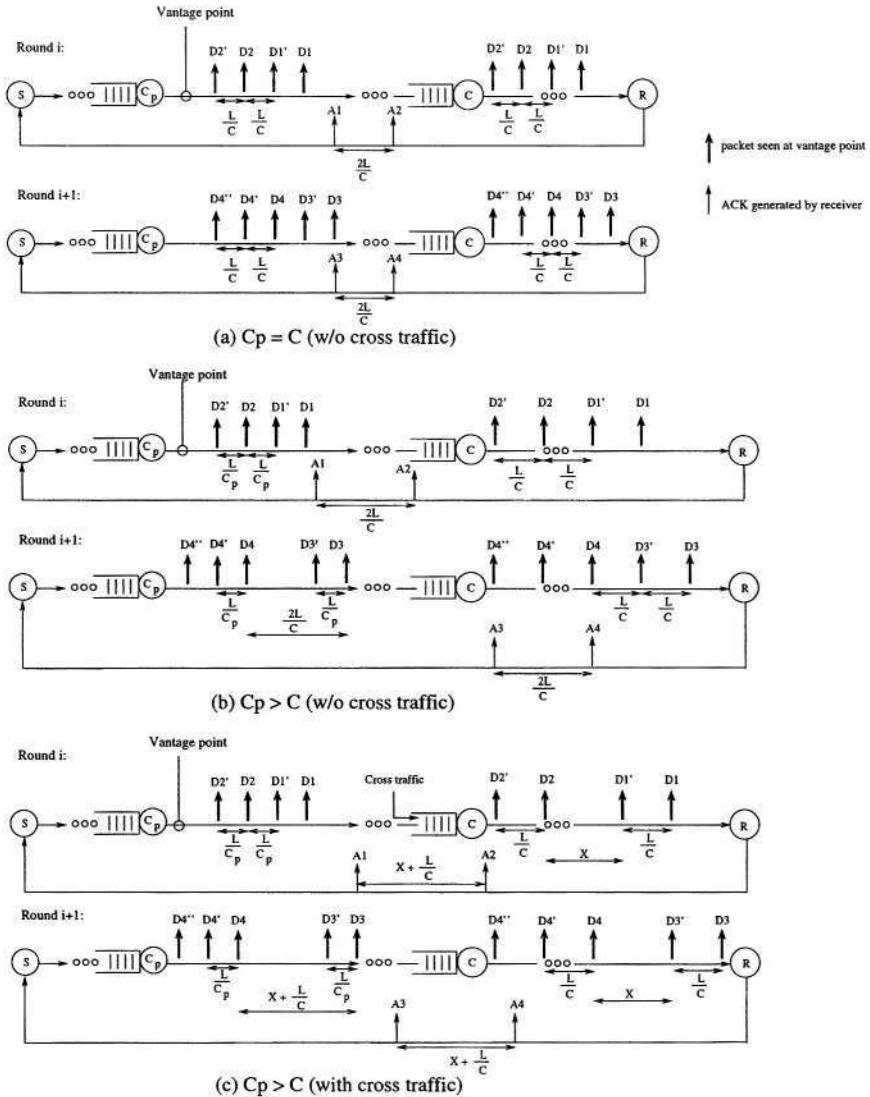


Fig. 1. TCP data and ACK dispersion sequences in three typical scenarios.

though a single mode with about 50% of the interarrivals is clearly visible, a second strong mode cannot be detected.

The capacity estimation technique is as follows. For a TCP flow f , let $P_f(i)$ be the size of the i 'th data packet, and $\Delta_f(i)$ be the dispersion between packets i and $i+1$. If packets i and $i+1$ have the same size, we compute a capacity sample $b_i = P_f(i)/\Delta_f(i)$. Note that packets with different sizes traverse the network with different per-hop transmission latencies, and so they should not be used by the packet pair technique [4]. As explained in the previous paragraph, we can assume

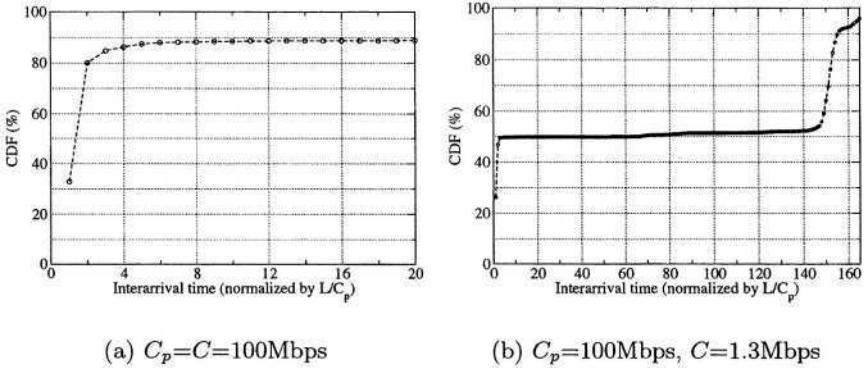


Fig. 2. Cumulative distribution of normalized packet interarrivals

that about 50% of the data packets are sent back-to-back due to the delayed-ACK algorithm, and so they can be used in capacity estimation. The rest of the packets may have been sent with a larger dispersion than L/C_p , and so they can underestimate C_p . Based on this insight, we sort the capacity samples of flow f and drop the lower 50% of them. To estimate C_p , we employ a histogram-based technique to identify the strongest mode among the remaining capacity samples. The center of the strongest mode gives the final capacity estimate \tilde{C}_f . The bin width that we use is $\omega = \frac{2(IRQ)}{K^{1/3}}$ (known as “Freedman-Diaconis rule”), where IRQ and K is the interquartile range and number, respectively, of the capacity samples.

The algorithm does not produce estimates for interactive flows, ACK flows, and flows with just a few data packets. For such flows, the number of packet pairs can be small and the detection of a capacity mode is quite prone to statistical errors.

3 Validation

We have validated the previous passive estimation technique with active measurements. Specifically, we send a file of size Y with *scp* from various sources around the Internet to a destination at Georgia Tech. At the same time, we collect a trace of TCP data packets using *tcpdump* at the destination host. The trace is then used to estimate passively the capacity of the corresponding path. We also measure the capacity of the same path using *pathrate* [4]. Since the trace is collected at the end host, the pre-trace capacity is equal to the end-to-end capacity in these experiments. To show the effect of the flow size on the accuracy of the estimation technique, we repeat each experiment for three values of Y : 40KB, 110KB, and 750KB.

Table 1 shows the results of some validation experiments. The passive and capacity estimates are reasonably close, and they correspond to common capacity values such as 256Kbps (upstream DSL), 1.5Mbps (T1), 10Mbps (Ethernet), and

100Mbps (Fast Ethernet). Passive estimation with larger flows obviously helps, even though the results with the 40KB flows are not too inaccurate either.

Table 1.

Name	Location	Passive estimate (Mbps)			Pathrate (Mbps)
		750KB	110KB	40KB	
lulea.ron.lcs.mit.edu	Sweden	97	96-97	94-96	99-101
mazul.ron.lcs.mit.edu	MIT	1.4-1.5	1.4-1.5	1.5-1.7	1.5
magrathea.caida.org	UCSD	97-98	96-99	93-95	94-96
diple.acad.ece.udel.edu	U-Delaware	98	97-98	97-99	94-97
aros.ron.lcs.mit.edu	U-Utah	9.5-9.7	9.2-9.7	11.1-11.2	11.9-12.3
thalis.cs.unipi.gr	Greece	9.1-9.2	8.3-8.5	6.4	9.7-9.8
dsl-64-192-141-41.telocty.com	Atlanta	0.225-0.226	0.226	0.225	0.226

4 Capacity Distributions

We performed capacity estimation on several packet traces from edge and backbone network links with a wide range of average rates. The three traces that we use in this paper are publicly available at the NLANR-MOAT site [8], and they are described in Table 2. Note that the capacity estimation technique can provide an estimate for a small fraction of flows (about 4-13%, depending on the trace), but for a large fraction of bytes (about 80%).

Table 2.

Trace	Link type	Date	Local Time	Rate (Mbps)	TCP flows	Estimate C_p % flows	%bytes
MRA-if-1	OC-12	2002/08/07	20:12:00-20:13:30	180.4	71357	3.8	82.7
MRA-if-2	OC-12	2002/08/07	20:12:00-20:13:30	157.3	118786	8.2	83.4
Auck-1-if-0	OC-3	2001/04/02	14:27:00-14:30:00	2.8	9657	7.8	85.5
Auck-2-if-0	OC-3	2001/06/11	08:56:00-08:59:00	4.8	14017	12.1	81.5

Figure 3(a) shows the distribution of TCP flow capacity estimates for the two interfaces of the MRA-1028765523 OC-12 trace. The cumulative distribution is plotted in terms of TCP bytes, rather than TCP flows. Note that most bytes are generated from flows with capacities 1.5Mbps, 10Mbps, 45Mbps, and 100Mbps. These values correspond to some commonly used links (T1, Ethernet, T3, and Fast Ethernet, respectively). Figure 3(b) shows the distribution of TCP flow capacity estimates for two segments of the Auckland OC-3 trace [8]. Note that the

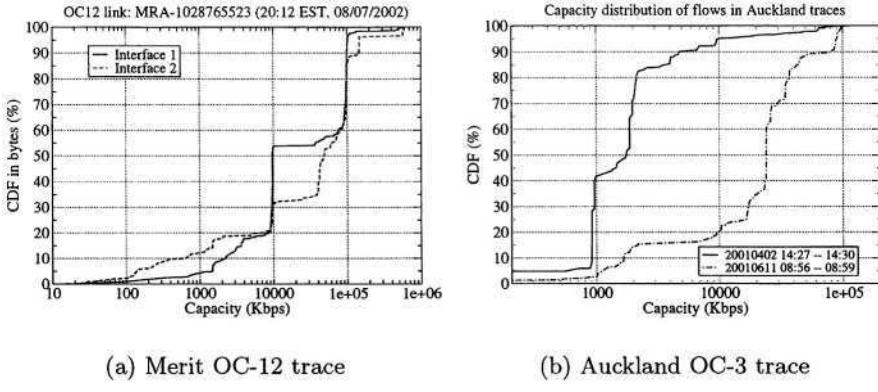


Fig. 3. Capacity distribution in terms of bytes

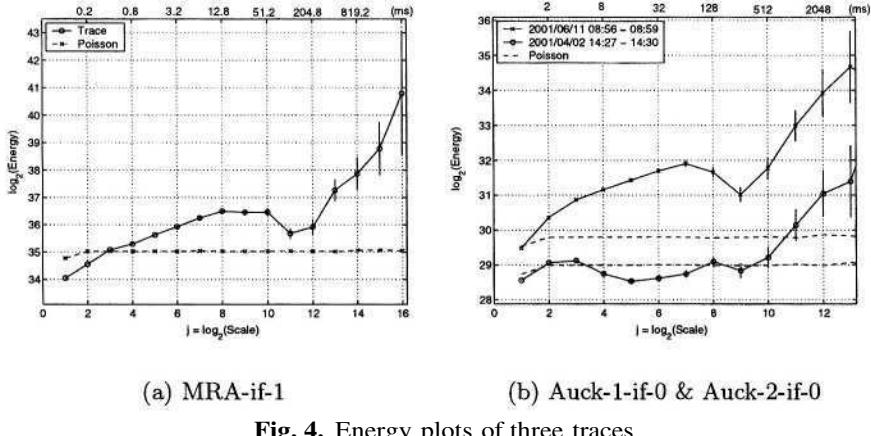
two distributions are quite different. A major difference is that the 2001/06/11 trace carried traffic from TCP flows with significantly higher capacities. Specifically, about 80% of the bytes in that trace were generated from TCP flows with a capacity of more than 10Mbps. On the other hand, more than 80% of the bytes in the 2001/04/02 trace were carried by TCP flows with a capacity of less than 3Mbps.

We have also investigated the correlation between the capacity of a flow and the flow's average throughput and maximum window size. Due to space constraints we do not report the details of that analysis here. The main result, however, is that both correlation coefficients are close to zero, implying that the previous two flow characteristics are independent of the pre-trace capacity, and probably independent of the end-to-end capacity as well. The reason may be that the throughput and window size of bulk TCP transfer are often limited by the receiver's advertised window. The correlation coefficient between C_p and the flow size is also close to zero.

5 Capacity and Traffic Burstiness

We employ wavelet-based energy plots to analyze the correlation structure and burstiness of traffic in a range of short time scales [9,10]. Since the Poisson stream (i.e., independent exponential interarrivals) is traditionally viewed as benign while traffic with stronger variability is viewed as bursty, we use the Poisson process as a reference point in the following analysis. The energy plot of a Poisson process with rate λ is a horizontal line at $\log_2(\lambda T_0)$, where T_0 is the minimum time scale of the energy plot. If the energy of a traffic process X_j at scale $T_j = 2^j T_0$ is higher than the energy of a Poisson process that has the same average rate with X_j , then we say that X_j is bursty at scale T_j . Otherwise, we say that X_j is smooth at scale T_j .

Figure 4(a) shows the energy plot of a highly aggregated backbone trace, which carries thousands of flows at any point in time. We focus in time scales up to 100msec ($j \leq 10$). The correlation structure and burstiness of the trace in longer

**Fig. 4.** Energy plots of three traces

scales is determined by Long-Range Dependency effects that have been studied in depth in earlier work [11]. The trace is clearly bursty, compared to Poisson traffic, in all time scales between 1-100 msec. This may be surprising from the perspective of the theory of point processes, because that theory predicts that the superposition of many independent flows tends to a Poisson process [12]. There is no actual contradiction however. The previous superposition result applies to flows with rate R/N , where N is the number of aggregated flows, i.e., it assumes that the flow interarrivals become “sparser” as the degree of aggregation increases. That is not the case, however, in typical packet multiplexers; flows are aggregated in higher capacity links without artificially increasing the interarrivals of each flow.

Figure 3 shows that a major part of the previous trace (about 40% of the bytes) are generated from 100Mbps flows, i.e., flows with comparable capacity to the 622Mbps capacity of the monitored OC-12 link. These high-capacity flows are not small relative to the aggregate, neither in terms of size (not shown here), nor in terms of rate. Consequently, we should expect that their correlation structure and burstiness can significantly affect the burstiness of the aggregate traffic.

To elaborate on the previous point, we examine the energy plot of the two Auckland traces from Figure 3(b). As previously shown, the 2001/06/11 trace carries traffic from significantly higher capacity TCP flows than the 2001/04/02 trace. Figure 4(b) shows the corresponding energy plots. The 2001/06/11 trace is clearly bursty, while the 2001/04/02 trace remains below the Poisson energy level. We note that the two traces are similar in other aspects, including flow RTTs, number of active flows, flow size distribution, and average utilization.

To further examine the link between short scale burstiness and flow capacities, we separate the TCP flows of the trace MRA-if-1 for which we have a capacity estimate in two subsets: S_{H+} and S_{H-} . S_{H+} consists of all flows with capacity larger than a threshold H (in Mbps), while S_{H-} includes the remaining flows that have a lower capacity. The average rate of S_{H+} and S_{H-} are 119Mbps and 30Mbps, respectively. The energy plots of S_{H+} and S_{H-} are shown in Fi-

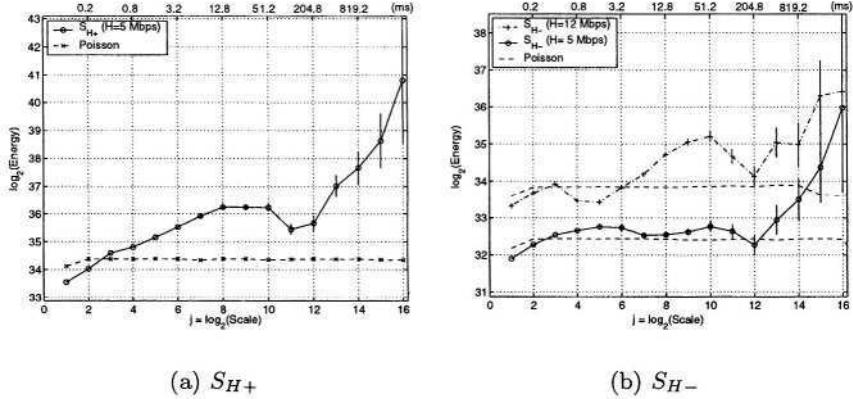


Fig. 5. Energy plots of S_{H+} and S_{H-}

figure 5. If the threshold H is between 1-10Mbps, the resulting energy plots are not so sensitive to the exact value of H , and so we set $H=5$ Mbps. Notice that the energy plot of S_{H-} is about at the same level with that of the corresponding Poisson process, meaning that the lower capacity flows do not generate significant burstiness. On the other hand, S_{H+} has much higher energy than the corresponding Poisson process, as shown in Figure 5(a), confirming our earlier conjecture that high capacity flows cause significant burstiness in short scales. Note that if we set $H>10$ Mbps, then both S_{H+} and S_{H-} will be characterized as bursty. Finally, it should be mentioned that the file size distributions of S_{H+} and S_{H-} are similar. S_{H+} includes 1937 flows, with 86% of them being larger than 10KB. S_{H-} includes 773 flows with 90% of them being larger than 10KB. Consequently, the difference in the burstiness of the two subsets cannot be attributed to their flow size distribution.

6 More Recent Results

In more recent work [13], we have further investigated the connection between flow capacities and short time scale burstiness. The main result of that work is to explain the origin of such burstiness based on TCP self-clocking. Specifically, we have shown that, under a certain condition on the flow's window size and bandwidth-delay product (that is proportional to the flow capacity), TCP self-clocking can generate a two-level ON/OFF interarrival structure. That structure results in considerable traffic burstiness and strong correlations in sub-RTT time scales.

Acknowledgment. This work would not be possible without the traces collected from the NLANR Passive Measurement and Analysis (PMA) project. The NLANR PMA project is supported by the National Science Foundation Cooperative agreements ANI-9807479 and ANI-0129677 and by the National Laboratory for Applied Network Research.

References

1. Zhang, Z.L., Ribeiro, V., Moon, S., Diot, C.: Small-Time Scaling behaviors of Internet backbone traffic: An Empirical Study. In: Proceedings of IEEE INFOCOM. (2003)
2. Prasad, R.S., Murray, M., Dovrolis, C., Claffy, K.: Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. IEEE Network (2003)
3. Carter, R.L., Crovella, M.E.: Measuring Bottleneck Link Speed in Packet-Switched Networks. Performance Evaluation **27,28** (1996) 297–318
4. Dovrolis, C., Ramanathan, P., Moore, D.: Packet Dispersion Techniques and Capacity Estimation. Technical report, Georgia Tech (2004) To appear in the IEEE/ACM Transactions on Networking.
5. Pasztor, A., Veitch, D.: Active Probing using Packet Quartets. In: Proceedings Internet Measurement Workshop (IMW). (2002)
6. Paxson, V.: End-to-End Internet Packet Dynamics. IEEE/ACM Transaction on Networking **7** (1999) 277–292
7. Lai, K., M.Baker: Measuring Bandwidth. In: Proceedings of IEEE INFOCOM. (1999) 235–245
8. NLANR MOAT: Passive Measurement and Analysis. <http://pma.nlanr.net/PMA/> (2003)
9. Abry, P., Veitch, D.: Wavelet Analysis of Long-Range Dependent Traffic. IEEE Transactions on Information Theory **44** (1998) 2–15
10. Veitch, D.: Code for the Estimation of Scaling Exponents.
http://www.cubinlab.ee.mu.oz.au/darryl/secondorder_code.html (2001)
11. Park, K., W. Willinger (editors): Self-Similar Network Traffic and Performance Evaluation. John Wiley (2000)
12. Cox, D.R., Isham, V.: Point Processes. Chapman and Hall, London (1980)
13. Jiang, H., Dovrolis, C.: The Origin of TCP Traffic Burstiness in Short Time Scales. Technical report, Georgia Tech (2004)

Micro-time-scale Network Measurements and Harmonic Effects

Mark Carson and Darrin Santay

National Institute of Standards and Technology (NIST)*

{carson, santay}@nist.gov

Abstract. As network transmission speeds increase, packet streams increasingly uncover fine details of the interior behavior of router hardware and software. This behavior reveals itself as a set of harmonic effects, as interior clocks periodically interrupt packet forwarding, and interior queues periodically empty and fill. We examine this behavior with a Linux-based router employing a variety of gigabit Ethernet interfaces, with a view toward two goals: the creation of harmonic models of router forwarding performance which are accurate and yet mathematically simple and fast; and the analysis of the potential for an undesirable succession of positively reinforcing router “reverberations.”

1 Introduction

It has been observed that network behavior differs significantly at a fine (millisecond level) time scale from that seen at coarser (one second or larger) time scales. [1] [2] Now, with transmission speeds reaching the gigabits/second range and beyond, network behavior at a micro scale (microsecond level) becomes relevant.

In the course of performing a high-speed calibration analysis of the NIST Net network emulator [3], we encountered a number of interesting phenomena (independent of NIST Net) in the underlying Linux-based router, which are only apparent at these very short time scales.

In effect, the high-speed uniform packet streams used to perform the analysis act as a sort of “network ultrasound probe,” exposing underlying characteristics of the forwarding node. When packets arrive in patterns which correlate strongly with periodic functions such as timer ticks, queue flushing or internal status updating, they may experience unusually low or high latencies, compared with those arriving randomly.

As an analytical tool, then, we can employ a range of such packet probes to uncover these fundamental node “frequencies.” Once isolated, these frequencies

* Certain software, equipment, instruments, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose. Product names mentioned in this paper are trademarks of their respective manufacturers

can be used to create computationally simple approximate models of node behavior. In this paper, we subject our Linux-based router to such probing and uncover three distinct fundamental frequencies of this sort; we then present analyses and generative mathematical models for these behaviors.

It should be noted that beyond their use as an artificially-created analytical tool, there is evidence that such relatively uniform high-speed packet streams occur naturally. At sufficiently high data rates, not only do interpacket gaps in packet trains [4] shrink, but their variance tends to decrease as well. This is due to a variety of factors: increased use of constant-rate UDP-based packet streams for audio and video transmission and other such uses [5] [6], increased use of persistent TCP connections [7] [8] which decrease the variability due to TCP's slow start, new hardware characteristics, such as *interrupt coalescing*, [9] featured by high-speed network interfaces, and other possible statistical reasons. [10] [11]

Given this situation, there is the possibility that such streams may inadvertently reveal intermediate node frequencies, resulting in increased jitter (latency variance). In the worst case, successive nodes may reinforce this effect, leading to an undesirable network "reverberation." Hence, understanding fine-grained individual node behavior should provide better insight into overall network dynamics.

2 Methods

The data described here were generated using a SmartBits 6000B Performance Analysis System [12]. This device was used to drive uniform loads of up to 1 Gbit/sec through a Linux-based router (Figure 1). Individual packet latencies were then measured with the 10 MHz (0.1 μ sec precision) SmartBits timer.

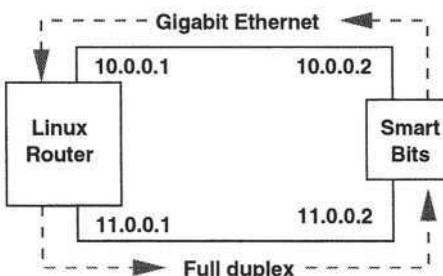


Fig. 1. Router testbed configuration

Given the influence of system bus design on gigabit Ethernet performance, [13] we conducted all tests using 64 bit/66 MHz PCI-based cards in 64 bit/66 MHz slots. The Linux system under test had a 1666 MHz Athlon processor and

1 GB RAM. It was based on Red Hat 7.3, with a stock 2.4.18 kernel installed. A variety of gigabit Ethernet interfaces were tested:

- SysKonnect SK-9844, one of the original 1000Base SX (fiber) interfaces with an Xaqli XMAC II controller.
- NetGear GA621, another 1000Base SX (fiber) interface with a National Semiconductor DP83820 controller.
- NetGear GA622T, also based on a National Semiconductor DP83820 controller, but with a 1000Base TX (copper) interface.

The range of tests covered offered loads from 100 kbps to 1 Gbps, with packet sizes from 128 to 1500 bytes. The complete set of results is available at the NIST Net web site. [14] In this paper, we will consider only a representative subset of tests, with loads from 100 Mbps to 900 Mbps, and packet sizes of 1024 bytes.

For all tests, we brought the load up to the specified level for ten seconds, then captured and recorded the latencies of approximately 200,000 packets. (At lower loads, some of the tests did not run long enough to generate that many packets, but in all cases at least 125,000 were captured.) For the analyses below, a representative slice of 30,000 packet latencies was taken out of the middle of the capture range; this was done to avoid any possible sampling “edge” effects.

3 Results

Figure 2 shows sample results, for 100, 300, 500 and 700 Mb/sec loads with NetGear GA621 adapters. Here, and in all the charts, latencies (y-axis) are expressed in units of 0.1 μ sec (100 nanosec).

As may be seen, packet latency behavior changes dramatically as the load increases. Looking at the corresponding latency distributions (Figure 3), we can identify three distinct performance regimes: At low loads, latencies are fairly uniformly distributed over a relatively narrow range, with an overlay of horizontal banding (seen as peaks in the distribution plots). As the load increases, the banding becomes increasingly pronounced, until the latency trace takes on a “quantum” appearance, with certain latency levels “allowed” and common, with other intermediate levels being “forbidden” and nearly absent. Finally, when the load reaches sufficiently high levels, the pattern changes again; the latency traces take on a vertical, peak and valley appearance, rather than a horizontal, banded one, and the latency distribution begins to resemble a normal curve (in the statistical sense).

The other Ethernet adapters exhibit similar behavior, but with some notable differences at higher offered loads. In the case of the NetGear Ga622T, the newest and, presumably, highest-performing adapter of the group, the transition from the horizontal quantum regime to the vertical normal regime happens at a higher load level; at 70% offered load (700 Mb/sec), it is in a transitional state between the two regimes (Figure 4).

For the SysKonnect SK-9844, the high-load latency traces take on a distinctive appearance. While still quasi-normal in terms of their distribution curves, they add a new, long-term periodicity to the trace curves (Figure 5).

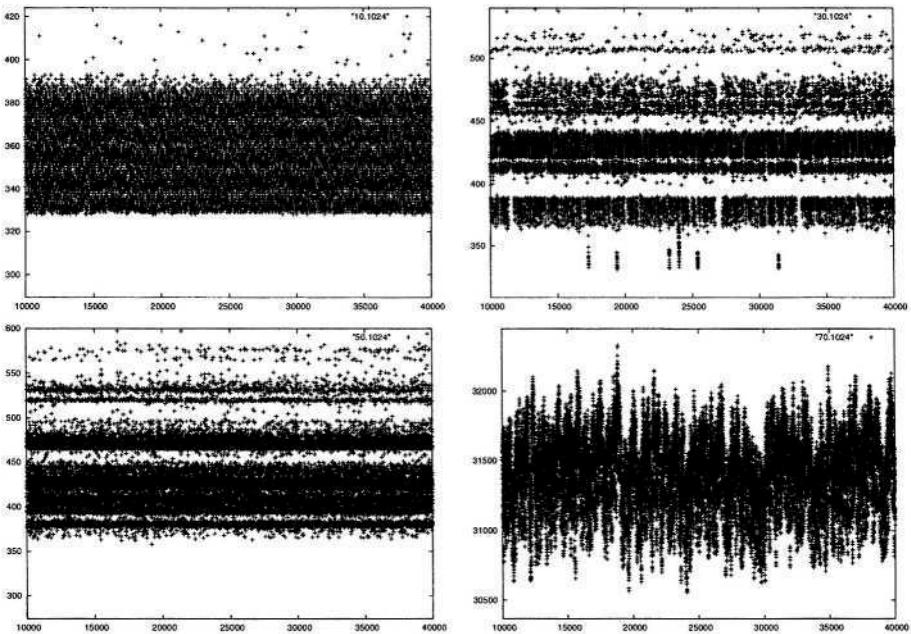


Fig. 2. GA621 latency traces for 100, 300, 500 and 700Mb/sec loads

4 Analysis

How can we explain the latency behavior exhibited by the various gigabit interfaces? The horizontal banding seen at low-to-mid load levels is a typical indication of an underlying clock during some point of processing; only when a clock tick occurs can packet processing advance. For example, latency traces when NIST Net-imposed delays are added (available at the NIST Net site [14]) show clear indications of the 8192Hz (122 μ sec) NIST Net clock.

In these traces, banding seems to occur at multiples of approximately 1.4 μ sec, or a bit over 700kHz. The likeliest explanation is found in the interrupt coalescing used by the gigabit interfaces to reduce the interrupt load on the CPU; [9] this acts in effect like a clock, bunching packets in certain latency ranges.

Banding would then be expected to be less distinct at lower loads, because such coalescing would occur only on input; once a packet is received, it can be handled all the way through the kernel forwarding code through output to the egress interface with no further delays, and hence no further coordination imposed. In this situation, the natural variation in processing time in the relatively long forwarding path would tend to smear the banding out.

By contrast, at medium loads, coordination is also required on output. With a packet size of 1024 bytes, when the load reaches 300 Mbps, interpacket arrival intervals are approximately 27 μ sec, which is less than the minimum latency for

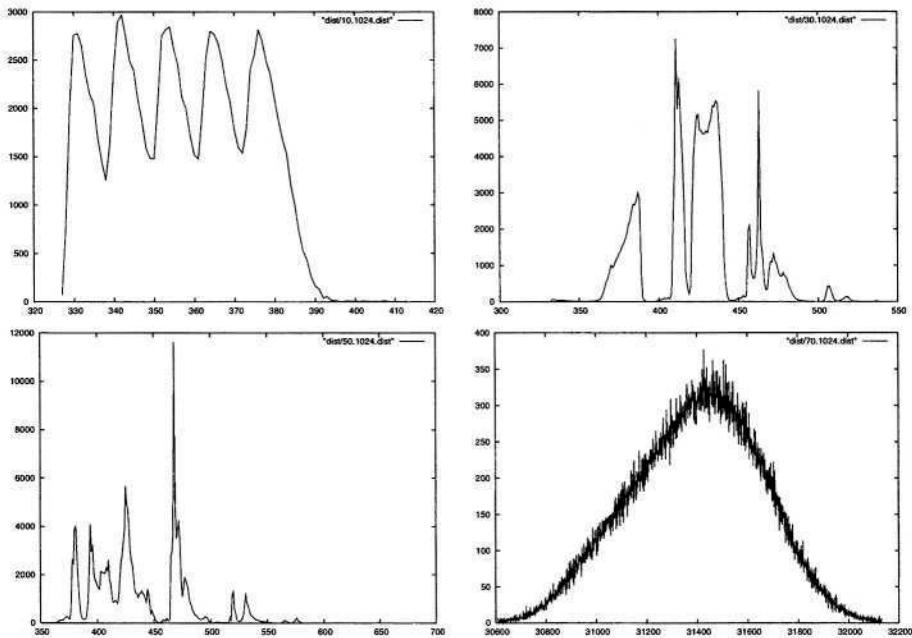


Fig. 3. GA621 latency distributions

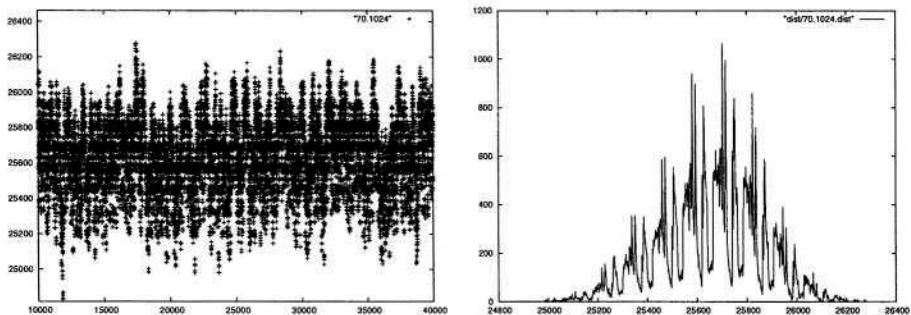


Fig. 4. GA622T latency trace and distribution

all the gigabit adapters (around 33 μ sec for 1024 byte packets). Hence, input processing can begin to interfere with output processing, forcing coordination between the two. This results in the quantized distinct bands seen in traces in this range.

At the very highest loads, queuing effects become evident. When packets arrive at rates higher than they can be forwarded, they tend to queue up until a critical point is reached, whereupon the queue is flushed, all packets are either

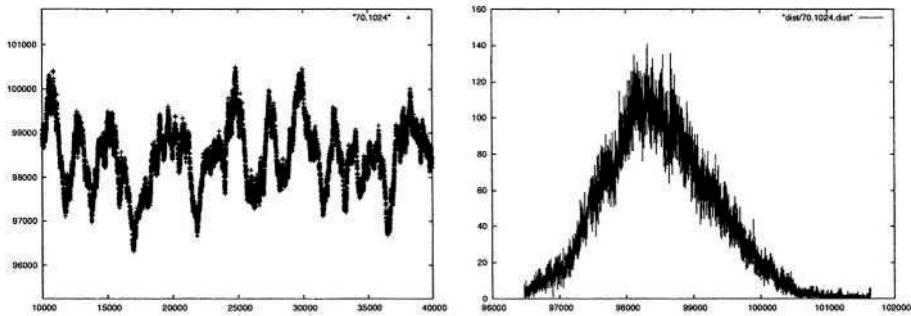


Fig. 5. sk9844 latency trace and distribution

forwarded or lost, and the cycle repeats. This pattern gives a ladder-like, vertical structure to the latency traces.

Looking at more detailed traces for 700Mb/sec and 900Mb/sec loads (Figure 6), it appears this cycle occurs around every 37 packets at 700Mb/sec and every 50 packets at 900Mb/sec, or in both cases approximately every 450 μ sec (2.2kHz). During this cycle, latency increases approximately 70 μ sec. A gap of approximately 50 μ sec is then required to flush out all packets; during this time several packets are lost (around 4-5 at 700Mb/sec and 6-7 at 900Mb/sec).

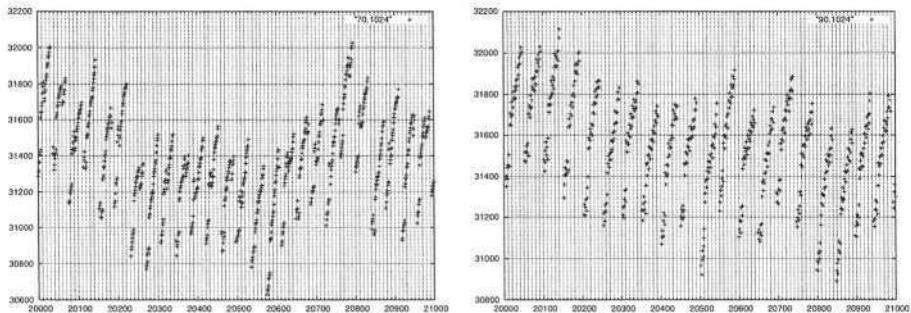


Fig. 6. GA621 detail traces for 700 and 900Mb/sec loads

The behavior of the SK-9844 card under high load differs in several respects. While its average latency is higher (985 μ sec at 700Mb/sec vs. 314 μ sec for the GA621), its loss rate is correspondingly lower (5% vs. 16%). Presumably the differences are due to the much larger buffering provided by the SK-9844 (enough for nearly 1000 packets). This allows the queuing behavior to play out over a much larger range.

5 Modeling

The relatively simple, repetitive structures exhibited by the latency traces facilitates the development of relatively simple, fast computational models of this behavior. The harmonic nature of the latency data encourages the notion of using Fourier transform-based models.

Our basic approach is to use a truncated Fourier representation; that is, given a representation of the latency data $l(t)$ as a Fourier series

$$l(t) = \sum_i^N c_i f_i(t)$$

we simplify it to

$$\bar{l}(t) = \sum_{i \in \max(N)} c_i f_i(t)$$

where $\max(N)$ is the subset of (indices of) Fourier coefficients c_i of greatest absolute value. For the models presented here, we chose $N = 30000$ (that is, began with a 30,000-point Fourier representation), and then truncated this to the largest 500 or 200 values.

Such modeling works quite well in unmodified form for higher data rates, where the queuing behavior overwhelms all other structure. The left side of Figure 7 shows the results of a truncated 500-coefficient Fourier representation of the GA621 data trace. The SK9844 behavior, shown on the right, is even simpler, being perfectly well modeled by a 200-coefficient Fourier representation.

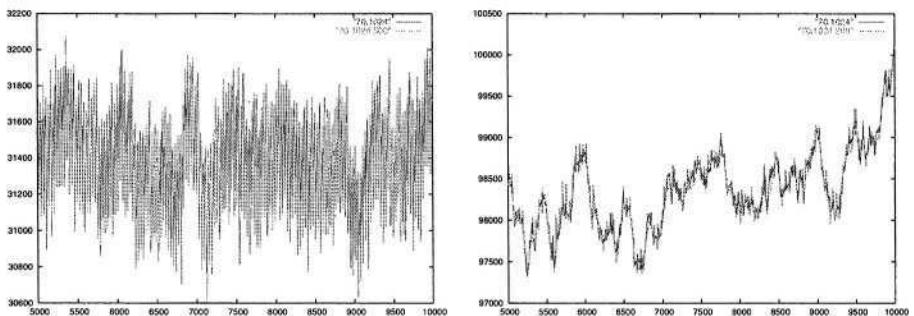


Fig. 7. GA621 and SK9844 modeled 700Mb/sec traces

At lower data rates, some provision must be made for the “quantum” nature of the latency distributions. The most straightforward method of doing so is to generate points using a simpler distribution (in this, case, the nearly-normal

distribution produced by a truncated Fourier representation), and then “nudge” them into the desired distribution as described in, for example [15] and [16]. Mathematically, if $F(x)$ and $G(x)$ are the cumulative distribution functions of the current and desired distributions, we replace a generated value x by

$$y = G^{-1}(F(x)).$$

The nudging process need not be terribly precise; in this case, we found that 100-point tabular approximations to the cumulative distribution functions are more than adequate. Applying these to a 500-coefficient truncated Fourier representations yields the modeled traces shown in 8.

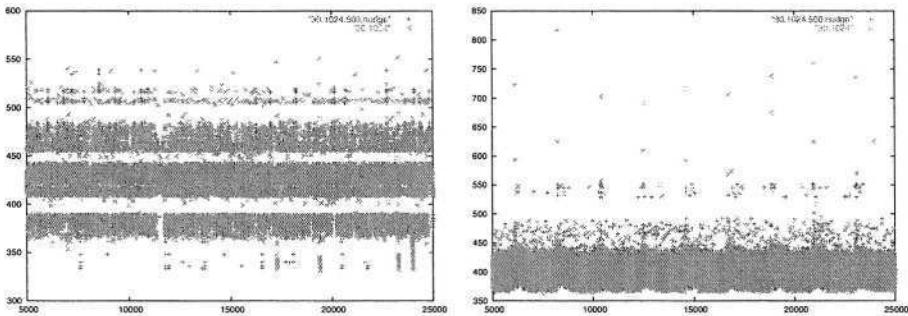


Fig. 8. GA621 and SK9844 modeled 300Mb/sec traces

Overall, then, all the observed latency distributions can be well-modeled with relatively compact representations, which allows for very fast latency value generation. In our first, totally unoptimized attempts, all representations require at most 700 data values (500 Fourier coefficients and two sets of 100-point distribution approximations); total analysis and regeneration time for 30,000 values was under 0.1 sec (over 300,000 values analyzed and regenerated per second). This can undoubtedly be improved greatly; in particular, it does not seem necessary to generate all 30,000 coefficients, since the desired few hundred all tend to be found within the first thousand or so.

When analysis has already been completed, values can be generated at rates well in excess of 1 million/second.

6 Conclusions

In this paper, we have presented empirical descriptions at the sub-microsecond level of the latency behavior of a variety of gigabit Ethernet interfaces in a Linux-based router. Based on these observations, we have posited some tentative

analyses, and then created simple Fourier transform-based generative models capable of recreating the observed behaviors.

The phenomena observed here suggest several further avenues of exploration. As one example, the quantization and queuing effects are clearly sources of increased jitter (latency variance). When multiple routers of similar characteristics are connected, these effects may tend to reinforce each other, or tend to cancel each other out, depending on the details of their interactions. In a simulation of two interconnected routers passing traffic at 300Mb/sec, we found jitter could vary more than 20%, with no real change in overall average latency, depending on the exact alignment of the two router's clocks. Interestingly, in this case overall jitter can be reduced, at the expense of a slight increase in average latency, by having the first router *add* a small amount of jitter to its output timings.

The data and analytical results described in this paper are all in the public domain and are available through the NIST Net web site [14].

Acknowledgments. The work described here was supported in part by the Defense Advanced Research Project Agency's Network Measurement and Simulation (NMS) project. We would also like to thank our colleagues at NIST, in particular Doug Montgomery, John Lu and Kevin Mills, for their valuable advice throughout the course of this project and during the preparation of this manuscript.

References

1. Agrawala, A., Shankar, U.: Fine-time-scale measurement of the Internet (1993–2000) <http://www.cs.umd.edu/projects/sdag/netcalliper/>.
2. Sanghi, D., Gudmundsson, O., Agrawala, A., Jain, B.: Experimental assessment of end-to-end behavior on Internet. Technical Report CS-TR 2909, University of Maryland (1992) <ftp://ftp.cs.umd.edu/pub/netdyn/infocom93.ps>.
3. Carson, M., Santay, D.: NIST Net: a Linux-based network emulation tool. SIGCOMM Comput. Commun. Rev. **33** (2003) 111–126
4. Jain, R., Routhier, S.A.: Packet trains: measurements and a new model for computer network traffic. IEEE Journal on Selected Areas in Communications **4** (1986) 986–995
5. Schulzrinne, H.: RTP profile for audio and video conferences with minimal control. RFC 1890 (1996) <http://www.ietf.org/rfc/rfc1890.txt>.
6. Schulzrinne, H., Rao, A., Lanphier, R.: Real time streaming protocol (RTSP). RFC 2326 (1998) <http://www.ietf.org/rfc/rfc2326.txt>.
7. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T.: Hypertext transfer protocol – HTTP/1.1. RFC 2068 (1997) <http://www.ietf.org/rfc/rfc2068.txt>.
8. Cohen, E., Kaplan, H., Oldham, J.: Managing TCP connections under persistent HTTP. Computer Networks (Amsterdam, Netherlands: 1999) **31** (1999) 1709–1723
9. Jin, G., Tierney, B.L.: System capability effects on algorithms for network bandwidth measurement. In: Internet Measurement Conference 2003 (IMC 2003). (2003)
10. Grover, W.: Self-organizing broad-band transport networks. In: Special Issue on Communications in the 21st Century. Volume 85. (1997) 1582–1611

11. Partridge, C. In: Gigabit Networking. Addison-Wesley (1994) 109, 231, 270
12. Spirent Communications: SmartBits 6000B Performance Analysis System (1996) <http://www.spirentcom.com/>.
13. Gray, P., Betz, A.: Performance evaluation of copper-based gigabit ethernet interfaces. In: Proceedings of the 27th Annual IEEE Conference on Local Computer Networks, IEEE Computer Society (2002) 679–690 <http://www.cs.uni.edu/~gray/gig-over-copper/hsln-lcn.ps>.
14. Santay, D., Carson, M.: NIST Net web site; calibration data (2003–2004) <http://www.antd.nist.gov/nistnet/calibration/>.
15. Chen, H., Asau, Y.: On generating random variates from an empirical distribution. AIEE Transactions **6** (1974) 163–166
16. Bratley, P., Fox, B.L., Schrage, L.E. In: A Guide To Simulation. Springer Verlag (1987) 149

Their Share: Diversity and Disparity in IP Traffic

Andre Broido¹, Young Hyun¹, Ruomei Gao², and kc claffy¹

¹ Cooperative Association for Internet Data Analysis
SDSC, University of California, San Diego

{broido,youngh,kc}@caida.org

² Georgia Institute of Technology
gaorm@cc.gatech.edu

Abstract. The need to service populations of high diversity in the face of high disparity affects all aspects of network operation: planning, routing, engineering, security, and accounting. We analyze diversity/disparity from the perspective of selecting a boundary between mice and elephants in IP traffic aggregated by route, e.g., destination AS. Our goal is to find a concise quantifier of size disparity for IP addresses, prefixes, policy atoms and ASes, similar to the oft-quoted 80/20 split (e.g., 80% of volume in 20% of sources). We define *crossover* as the fraction c of total volume contributed by a complementary fraction $1 - c$ of large objects. Studying sources and sinks at two Tier 1 backbones and one university, we find that splits of 90/10 and 95/5 are common for IP traffic. We compare the crossover diversity to common analytic models for size distributions such as Pareto/Zipf. We find that AS traffic volumes (by byte) are top-heavy and can only be approximated by Pareto with $\alpha = 0.5$, and that empirical distributions are often close to Weibull with shape parameter 0.2–0.3. We also find that less than 20 ASes send or receive 50% of all traffic in both backbones’ samples, a disparity that can simplify traffic engineering. Our results are useful for developers of traffic models, generators and simulators, for router testers and operators of high-speed networks.¹

1 Introduction

The lack of a predictive relation between *number* (cardinality) and the combined *size* (volume) of a collection of objects is a recurrent problem in Internet data analysis. The volume of a natural category, such as users, instants, and networks, can be unevenly distributed among individual objects. For example, while the Internet architecture does not have any *single* point of failure, 80-90% of routes use the top 20 providers. Similarly, 80 source ASes (among several thousand observed) can contribute 95% of the traffic on a link.

¹ Support for this work is provided by DARPA NMS (N66001-01-1-8909), and by NSF ANI-0221172, with support from the DHS/NCS.

In this paper we study the mismatch between number and size in terms of *diversity* and *disparity*. *Diversity* is the presence of a large number of distinct objects (e.g., many users sharing a link). Many objects have a natural size measure such as bytes per transfer, customers per provider, and visits per website. *Disparity* is concentration of a size measure in a small subset of objects. For example, a bursty flow may accumulate most of its duration in lulls. In extreme cases of disparity, a *giant cluster* forms, in which the aggregate size is comparable with total volume. We see this with TCP in the IP protocol space, with a popular operating system, and recently, with P2P applications in some networks.

Mathematically, diversity/disparity is present when the counting measure (such as number of addresses) and the size measure (e.g., traffic per address) are close to mutually disjoint (singular), i.e., supported by non-intersecting sets. Many Internet measures of interest are disjoint (e.g., in the case of lulls and bursts, most bytes are transferred in negligible total time). The ubiquity of disjoint measures renders comparison of IP objects challenging.

Neither diversity nor disparity are good or bad per se; the impact depends on the situation. Motivations to study disparity include offsetting its negative impacts (such as lack of resilience) and developing ways to manage total volume via control of a few contributors [1]. Data reduction is another motivation, e.g. frequent objects are assigned shorter bit strings (as in Huffman encoding). Indeed, a valid motto of Internet science is: *Find Disparity in Diversity*.

Size disparity is often called the “mice-elephants” phenomenon. It was observed early in Internet history that Internet traffic displays favoritism at any given aggregation, i.e., many small contributors and a few large ones². Researchers have described duration ‘elephants’ (long-lasting flows) [3] and bitrate or burst ‘elephants’ [4]. We focus on volume elephants, due to ISPs’ need for a metric for use in pricing as well as due to bitrate limitations of many links.

Comparing diversity/disparity across multiple datasets, directions, measures, percentile levels and source/destination granularities can easily result in an explosion of numbers. We propose a concise characteristic of disparity that we call *crossover*. We define it as the fraction $1 - c$ of volume accumulated in fraction c of top objects. We justify this metric in Section 3. Object size at the crossover x_c serves as a cutoff between the mice and elephant classes (cf. [5]). We study crossovers both empirically and analytically. We think that crossovers are potentially as useful as while being more descriptive than the 95th percentile currently used in many MIBs and autogenerated reporting software.

The paper is structured as follows. Section 2 discusses motivations, and in Section 3 we compute crossovers for uniform, exponential and Pareto distributions. We assess their validity as models for aggregated traffic by comparing their crossovers with observed values. In particular we find that uniform or exponential distributions have crossovers under 70/30. The Pareto density $Cx^{-\alpha-1}$, $1 \leq x \leq N$ approaches the observed splits of 90/10 for $\alpha = -1$ (Zipf

² “1% of those networks were responsible for 70% of the traffic [on NSFNET] for the month of December 1992” [2].

distribution) and $N = 10^{10}$, whereas 95/5 splits with realistic values of N can only be obtained for α in $0 < \alpha < 1$ range.

Section 4 presents our empirical analysis of diversity and disparity of IP addresses, prefixes, policy atoms and ASes (measured by bytes and packets) for the longest traces in our Backbone Traffic Data Kit. We find that for most combinations of IP categories (from addresses to ASes), measures (bytes or packets) and datasets, crossover ratios are above 90/10, with many above 95/5. We discuss these results in Section 5 and outline future work and conclusions in Section 6.

2 Preliminaries

Motivation. The advent of Dag monitors [6] [7] has facilitated the capture of packet headers for over a terabyte of IP traffic (Table 1). However, humans cannot use such a volume of data in raw form. One needs to reduce 12 orders of magnitude (10^{12} bytes) to one order (a ten-line report) before it can be handled by a person[8]. Fortunately, the size disparity in network data makes reporting “heavy-hitters” possible and useful. We introduce the notion of *crossover* in order to concisely quantify disparity of large data sets. Crossover also allows for a rough estimate for the number of objects taken into account by a routing/traffic engineering optimizer.

The economic motivation for studying disparity of aggregated traffic can be explicit (e.g., for price differentiation [9]) or implicit, e.g., for security [10], QoS or traffic engineering [1] [5].

Routing-based aggregation. Lots of meaningful aggregations exist between an individual bit and all traffic observed in an experiment. We use 5 categories based on routing (including ports that route data through end systems): flow (source/destination address and port, protocol, 64s-timeout [11]), IP addresses, prefixes, policy atoms [12] and Autonomous Systems (ASes).³ Routing-based aggregation of IP traffic makes sense because it follows ISPs’ income flow.

Measures. The most commonly used measure is the counting one; it assigns 1 to each object. Data aggregation maps one category to another, e.g., bytes to packets, or prefixes to origin ASes. The counting measure is then collapsed to a “marginal” that counts the number of elements in an aggregated object, e.g., bytes per packet or addresses per prefix. It is notable that many measures used in practice map to node degree in graphs. An n -level aggregation hierarchy will produce $n(n-1)/2$ marginal measures, unmanageable even for small n . The multitude of available measures can explain why researchers sometimes derive inconsistent answers to the same question. Disjoint measures (Section 1) can calibrate the concepts of rare, typical and prevalent in divergent, incompatible ways. To avoid the perils of ambiguity, we only use two measures, bytes and packets, for each type of routed object.

Crossover. Knowing the boundary between mice and elephants is a requirement of many traffic engineering schemes [5] [14]. However, there is no natural

³ We separate source and destination rather than working with a matrix (cf.[13]).

boundary in the size spectrum. In fact, there are cases where most volume comes from midrange (“mules” [15]) contributions.

To address these concerns we studied the dependence of the cutoff on the proportion of traffic in the elephant class and aggregation level, and compared the object count median to the size median. We found however that the mice-elephant cutoff is best placed at the crossover point. We are now ready to discuss the virtues of this statistic in detail.

3 Theory

Mice-elephant boundary. Labeling an object as mouse (contributing to numbers) or elephant (contributing to mass) can be cast as hypothesis testing [16]. Let $f(x)$ be the density of objects of size x (the number of objects of size x divided by the number of observed objects). The mass density at x is $xf(x)/\bar{x}$ where $\bar{x} = \int_0^\infty xf(x)dx$ is the average object size. The null hypothesis H_0 , “the object is a mouse,” has likelihood function $f(x)$, whereas $xf(x)/\bar{x}$ is the likelihood for competing hypothesis H_1 , “the object is an elephant.” The maximum likelihood decision would amount to a cutoff at \bar{x} : elephants are objects whose size is above the average. This is the point where two densities intersect, ($f(x) = xf(x)/\bar{x}$ at $x = \bar{x}$.) Figure 1(a) shows an example for flow and byte measures’ densities. The 10 kb intersection agrees with the average in Table 2 (Section 4). We leave this approach for future work.

Another option is to equalize the type I error (fraction of objects over the cutoff) with the type II error (fraction of mass under the cutoff). This is a natural choice when the cost of each error is unknown, and it follows a statistical tradition of taking confidence intervals with equal significance at either side. We call s the crossover threshold if the share of objects above s and share of mass below s equals c . The proportion $1 - c : c$ is the *crossover split*. Figure 1(b) presents an example of a cdf and ccdf intersection for prefix volumes in D04 (see Section 4.)

An equivalent definition for the crossover s is the point where the cdf of objects crosses the volume ccdf, or where the sum of two cdfs or two ccdfs equals 1. Since the cdf sum monotonically increases from 0 to 2, the crossover always exists. We take the size for which the volume share of top objects and their counts’ share add up closest to 100%, which locates the crossover between two medians, for object count and for volume, which serve as upper and lower bounds on the elephant cutoff (Fig.1 (b)).

We can now compute examples of crossovers for some standard distributions. Note that linear transforms $x \mapsto kx$ do not change the crossover split (although shifts $x \mapsto x + a$ do), so we can take any scale factor in the formulas.

For a continuous density $f(x)$, crossover size s satisfies an equation

$$\int_0^s f(x)dx + \int_0^s xf(x)dx/\bar{x} = 1. \quad (*)$$

We give explicit values for several special cases below.

All sizes equal: object counts equal object sizes, resulting in a 50/50 split.

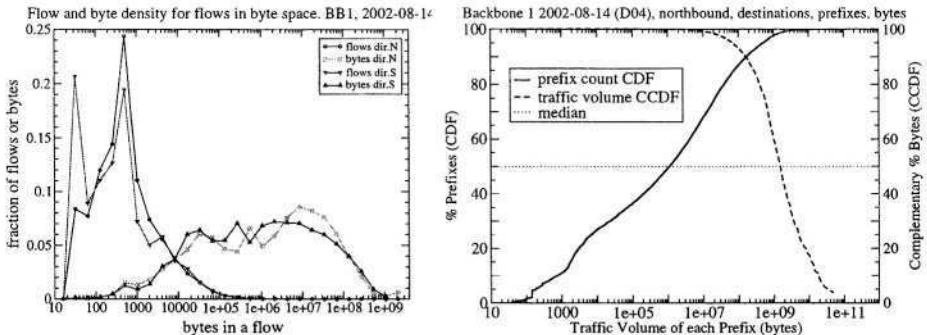


Fig. 1. a) Disjointness of flow and byte measures on 5-tuple flows, D04 b) Crossover for destination prefix byte counts, D04N

Uniform distribution. For $0 \leq s \leq 1$, the equation $s + s^2 = 1$ is satisfied by the golden section[16]⁴ $\frac{\sqrt{5}-1}{2}$. The split is 61.8/38.2, about 60/40.

Exponential: $P(X < s) = 1 - e^{-s}$. Equation (*) reduces to $1 - e^{-s} - se^{-s} + 1 - e^{-s} = 1$, i.e. $2 + s = e^s$; $s = 1.146$ and the split is 68.2/31.8, about 70/30.

Pareto distribution is usually truncated at some N , which specifies the maximum object size. On the interval $[1, N]$, the cdf $F(s) = P(X \leq s) = \frac{1-s^{-\alpha}}{1-N^{-\alpha}}$, $\alpha > 0$; mass fraction is $M(s) = \frac{1-s^{-\alpha+1}}{1-N^{-\alpha+1}}$, $\alpha \neq 1$ and Equation (*) becomes

$$\frac{1-s^{-\alpha}}{1-N^{-\alpha}} + \frac{1-s^{-\alpha+1}}{1-N^{-\alpha+1}} = 1. \quad (**)$$

For $\alpha = 1$ (Zipf distribution) $M(s) = \ln s / \ln N$.

We derive an approximate solution to $F(s) + M(s) = 1$, or $F(s) = 1 - f$, $M(s) = f$, by dropping $N^{-\alpha}$ from F' s denominator. Zipf's 80/20 split is at $N \approx 3000$ (e^8), $s \approx 5$ ($e^{1.6}$) and 90/10 at $N = 10^{10}$, $s = 10$. The 80/20 rule may thus have its origin in the Zipf distribution with a ratio of 3000 from highest to lowest value. It may well resemble the bitrate consumption disparity found in most ISPs; only a few of them offer a wider range of bitrates (for example, 3.5 orders covers DSL at 512 Kbps to OC-48 at 2.5 Gbps).

When $\alpha > 1$, Equation (**) reduces to $s^\alpha = s + 1$ for large N ; for $\alpha = 2$ this again produces a golden section split of 62/38. In general, as α grows, the split evens out since large sizes become less likely.

More extreme splits than 90/10 only occur for Zipf when N exceeds the current typical traffic range (e.g., 95/5 requires $N = 10^{26}$), but Pareto with α outside the conventional interval $\alpha \geq 1$ ([18]) can produce these splits for moderate sizes of N . For $\alpha = 0.5$, for example, Equation (**) holds whenever $s = \sqrt{N}$, resulting in a 90/10 split at $N \approx 6600(e^{8.8})$, $s \approx 90$, and 95/5 at $N \approx 133,000(e^{11.8})$, $s \approx 365$. Section 4 confirms these results.

⁴ Also known as *golden ratio* or *divine proportion* [17].

Table 1. Bulk sizes of OC-48 and OC-12 datasets

Set	Bb	Date	Day	Start	Dur	Dir	Src.IP	Dst.IP	Flows	Packets	Bytes
D04N	1	2002-08-14	Wed	09:00	8 h	Nbd (0)	2124 K	4074 K	106.6 M	2144 M	1269 G
D04S	1	2002-08-14	Wed	09:00	8 h	Sbd (1)	1122 K	12661 K	193.8 M	3308 M	2140 G
D05I	U	2002-08-14	Wed	08:22	13 h	Inbd (1)	961 K	11183 K	37.6 M	538 M	326 G
D05O	U	2002-08-14	Wed	08:20	16 h	Obd (0)	25.6 K	1412 K	22.0 M	549 M	249 G
D08N	1	2003-05-07	Wed	00:00	48 h	Nbd (0)	3902 K	8035 K	275.5 M	4241 M	2295 G
D09N	2	2003-05-07	Wed	10:00	2 h	Nbd (1)	904 K	2992 K	56.7 M	930.4 M	603 G
D09S	2	2003-05-07	Wed	10:00	2 h	Sbd (0)	466 K	2527 K	47.3 M	624.2 M	340 G

Table 2. Rates and size ratios for OC-48/OC-12 datasets. New sources, destinations and flows per second are bulk averages over the whole trace (in units of 1000/sec).

Trace	Sr/s	Ds/s	Fl/s	kpps	Mbps	Ut.%	Fl/Sr	Fl/Ds	Pk/Fl	Pk/Ds	Bt/Fl	Bt/Pk
D04N	74	141	3700	74	352	14.2	50	26	20	526	11906	592
D04S	39	440	6730	115	594	23.9	173	15	17	261	11041	647
D05I	21	245	821	12	57	9.2	39	3	14	48	8668	605
D05O	0.44	25	381	10	35	5.6	857	16	25	389	11347	454
D08N	23	47	1594	25	106	4.3	71	34	15	528	8331	541
D09N	126	415	7881	129	671	26.9	63	19	16	311	10635	649
D09S	65	351	6566	87	378	15.2	101	19	13	247	7193	545

4 Diversity and Disparity in High-Speed Traffic

Data sources. We use four longest datasets from our Backbone Traffic Data Kit (Tab. 1): D04, D05, D08 and D09.

The data in D04, D08, D09 was collected by OC-48 monitors using DAG 4 cards from U.Waikato/Endace [7] on Linux platform. D04 contains 8 hours of OC-48 traffic at up to 28.5% utilization (over 1 sec intervals)) taken at Tier 1 Backbone 1 (BB1) in August 2002. D08 and D09 were captured in May 2003. D08 covers 48 hours of Backbone 1 traffic from the same link as D04, albeit at lower utilization. (Only the northbound direction was captured.) D09 contains 2 hours (overlapping with D08) at up to 30.6% OC-48 utilization taken in Tier 1 Backbone 2 (BB2). Backbone links connect San Jose in the south to Seattle in the north. BB1 and BB2 use Packet over Sonet (POS). BB2 also prepends 80% packets with 4-byte MPLS headers (fewer than 50 distinct labels used on each direction; always one label in a stack.) Both encapsulations result in a small reduction of available Sonet payload.⁵

D05 was collected on an OC-12 (622 Mbps) ATM link that carries traffic between a university and the Internet. We label link directions by prevalence of inbound (I) and outbound (O) traffic, although due to multihoming each direction carries both. D05 is taken on the same day as D04. All traces are

⁵ The reduction depends on average packet size and the extent of HDLC byte stuffing [19]. Knowing these factors is essential for precise bitrate estimates. Utilizations here and in Table 2 give IP packet volume divided by Sonet raw bitrate of 2488.32 Mbps.

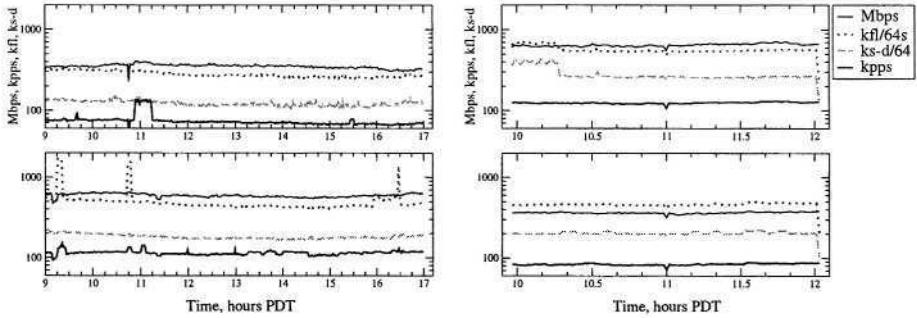


Fig. 2. Traffic rates (64 sec intervals) D04N (left), D09N (right). Byte and packet counts are expressed in Mbps, kpps respectively as per-second averages.

captured in the middle of the week around noon. We used CoralReef [13] and other CAIDA programs for data processing.

The raw diversity of our data is high. Traces differ by utilization, traffic symmetry, and temporal dependencies. In Figure 2 each panel shows the number of bytes, flows, source-destination pairs and packets. The remarkable stationarity of the traces with respect to baselines (major bursts are rare) means that volume distributions of these traces are convincing from a stochastic viewpoint.

An interesting property of traces D04 and D09 is equality between orders of magnitude for bitrate (Mbps) and the number of flows active per (64-sec) interval (a flow over 64 sec translates to 1 kbps of average bitrate). The maximum value of flows/second is 48702 for D04S and 20778 for (similarly utilized) D09N (the discrepancy is due to attacks in D04, see below). The maximum number of source-destination pairs is 24310 for D04S (no major attack at this second) and 15531 for D09N. For 64s intervals the maximum numbers of flows is 1.71M for D04S (peak of 220K IP pairs) and 719K flows (peak of 435K IP pairs) for D09N. In addition, D08 and D05 have diurnal variation with a factor of 2-6X.

Another interesting property of our datasets is the almost constant average bytes/flow. Table 2 consistently shows it at around 10 kbytes. Packets/flow, packets/destination, and bytes/packet are also of the same order of magnitude for most traces. The only exception is the inbound university trace D05I, skewed by backscatter [20], scans and other traffic debris attracted to a large address block.⁶ We plan to report in future whether and to what extent these ratios are invariant in high-speed IP traffic.

Prefix/AS diversity. The diversity of prefixes and ASes in our data is also high. Taken together, sources and destinations on both directions of each link cover 30-55% of Route Views (semiglobal [22]) prefixes and 42-62% ASes. However, disparity of coverage between directions can be high, e.g., in D09 the northern side of the link has only data from/to 1.5-4% of prefixes and ASes.

⁶ Our full analysis [21] skips all traffic directed to that block.

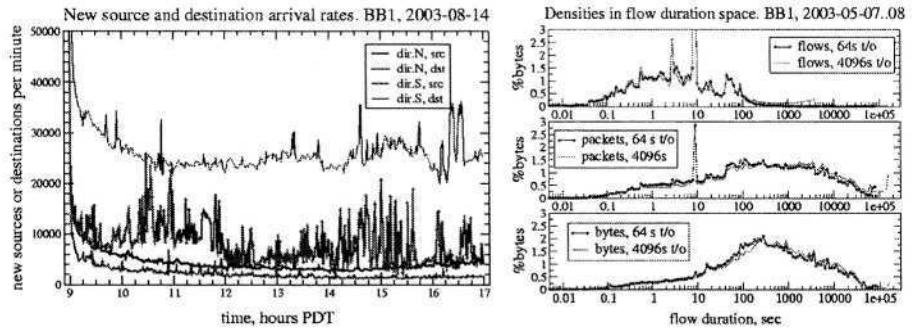


Fig. 3. a) Bursts in destination arrival rates, D04 b) Flows, packet and byte densities in duration space for two timeouts

Another interesting property is the symmetry in coverage disparity. The number of sources on one direction of a link is of the same order of magnitude as the number of destinations on the other, even though these sources and destinations do not necessarily match each other.

Extreme disparity. Packet floods, DDoS attacks and IP address and port scan are usually viewed as traffic anomalies. However, each trace in our study (indeed, almost any wide-area Internet trace) contains examples of all these phenomena. This observation renders ‘anomalies’ normal in highly multiplexed traffic (cf. similar observations for backscatter data [20]). We view them as cases of extreme size disparity at particular aggregation levels.

In particular, *floods* are typically aimed at overwhelming the capabilities of the receiving machine at the other end, e.g. OS interrupt processing. To reach that goal, lots of small packets are sent. As a result, *packet rate* increases without proportional growth in utilization; cf. curve in the upper left plate of Figure 2.

DDoS attacks represent a particular type of flood; two such attacks appear present on southbound D04 (Figure 2 lower left). The source addresses sweep the whole /16 address blocks of an academic network in Asia; the destination addresses point to hosts on consumer networks in US and Turkey. Note that the excursions at the flow aggregation level are missing on the level of source-destination pairs (because of the restriction to /16s), while more aggressive address spoofing would transpire to that level as well. These attacks also change packet rates (bottom curve), but to a much smaller extent.

Another type of disparity derives from IP address and port *scans* done (among others) by viruses and hackers. This activity appears as a large number of source-destination pairs (address scans) or flows (port and address scans). Figure 3 shows multiple bursts in new source-destination pairs per minute due to repetitive scans going north. Scans may be present in the D09, Fig.2 (right).

Figure 5 (upper left) indicates a large number of destination IP addresses with small (40–200 bytes) traffic volume, concentrated at a few small packet sizes, 40, 40+40, 3*48 bytes, many of which reflect SYN probes from scanning tools. Half of

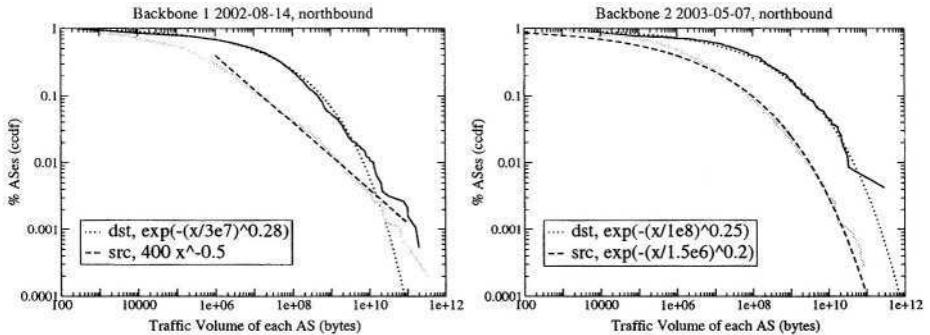


Fig. 4. Ccdf for source and destination bytes per AS. D04N (left), D09N (right).

dir.0's destinations and 2/3 of dir.1's destinations favor these packet size modes. Neither source IP cdf's (Fig. 5 left, dotted lines) nor AS distributions (upper right) have this property. Scans also impact packet counts per IP addresses, resulting in large (20-40% for D04 and D09) fractions of destination IPs with only one packet; the corresponding source counts are at most 20%. The impact of scans on traffic volumes per prefix (Figure 5, middle panels) and AS (Figure 5, right panels) is negligible since they affect only one prefix at a time.

The measures of Fig.3(b) (top: flow density; middle: packets; bottom: bytes) reveal the presence of scans in D08 differently. While demonstrating disparity in flow duration (by 7 orders of magnitudes) and the prevalence of *mules* [15], they also have several strong spikes. Many scans send 2 or 3 SYN packets to the same address, retransmitting after standard timeouts of 3 and 6 seconds. As a consequence, many flows (almost 12.4%) have duration around 9 sec (8.9-10.0) and about 2.6% at 3 sec (2.81-3.16). Packet density has a spike at 9 sec, while for byte density it is a barely visible bump.

Long tails. As shown in Section 3, the dominance of the contribution of the largest sizes is a consequence of the heavy tail. Internet loads cover many orders of magnitude, and the number of objects decays slowly.

We find that traffic distribution per AS in traces D04 and D09 is closer to Weibull (0.2-0.3) than to Pareto distribution (Figure 4), and even when its tail is close to Pareto, α can be as small as 0.5 (Fig.4, left panel, source ASes)) which results in stronger bias towards elephants a Zipf distribution with the same range would predict. In future work we plan to compare this result to the Pareto approximation with $\alpha \sim 1$ for 5-minute prefix byte volumes in [5].

Diversity at fixed percentiles. 95% of bytes in our data are sourced by fewer than 8% and destined to fewer than 20% of IPs, prefixes, and ASes. We find that the fraction of IP addresses comprising any traffic percentile is always smaller than the fraction of networks (prefixes, atoms, ASes); the fractions of networks are usually close to each other. More details of the analysis are at [21].

Crossovers. Table 3 quantifies the diversity/disparity of our data in terms of crossovers. We find that crossover splits are far more extreme than 80/20;

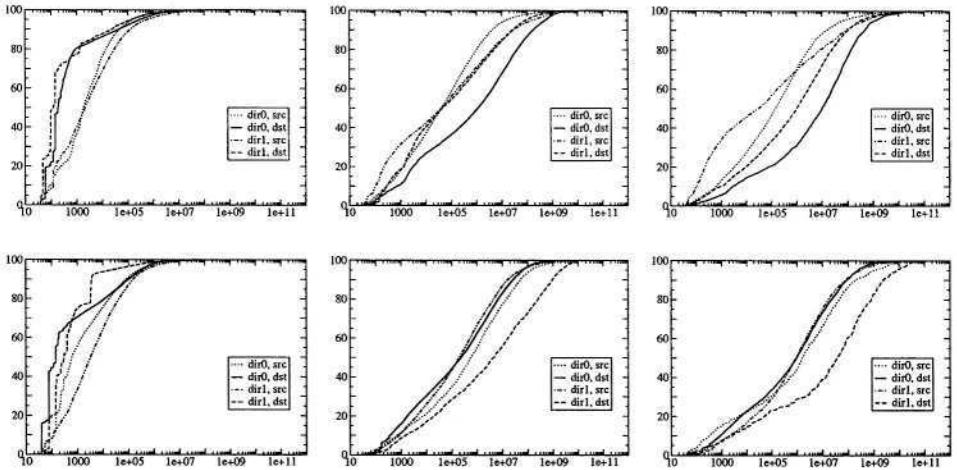


Fig. 5. IP addresses (left), prefixes (center) and ASes (right) as traffic sources (dotted lines) and sinks (solid and dashed lines). Object count cdfs over traffic volume (bytes per object). D04 (first row) and D09 (second).

in fact, almost all of them are in 90/10 range, and many exceed 95/5. This is in particular true for IP addresses, for which the disparity is highest. Packets disparity is usually higher than bytes' for IP addresses, but it is comparable for prefixes, atoms and ASes. Another property that also holds true for fixed percentiles is the position of atoms close to (often between) prefixes and ASes. The analysis in [21] also shows small numbers of atoms and ASes responsible for 50% of all traffic. In particular, the number of ASes responsible for half the traffic is always under 20 for backbone traces D04, D08, D09. These properties make atoms and ASes good candidates for use in traffic engineering [14].

5 Discussion

The phenomenon of size diversity/disparity has multiple causes. One cause strongly suggested by our data is the variable amount of aggregation present in the objects of the same taxonomic level. For example, traffic from an IP address may be generated by a single person, but it can also come from a network behind a NAT, potentially with thousands of addresses. Traffic toward an IP address may be destined to one user or to a popular news server; in the latter case the IP flow incorporates millions of individual contributions transferred as a single network news feed. A large AS with a large set of connected customer ASes may have inherited many of them via mergers and acquisitions. The same applies to other measures of wealth. The nature of the process shaping size distributions is a matter of debate dating back to Yule's 1924 paper [23] (cf. [24].) Indeed, no single model is likely to fit all cases of size disparity, even if limited

Table 3. Crossover for D04 (left) and D09 (right). Bytes (top), packets (bottom).

	N,src	N,dst	S,src	S,dst		S,src	S,dst	N,src	N,dst
IP	97.5/2.5	96.2/3.8	97.4/2.6	96.4/3.6	IP	97.0/3.0	93.1/6.9	95.6/4.4	97.9/2.1
Pf	97.2/2.8	89.9/10	96.6/3.4	93.7/6.3	Pf	93.5/6.4	89.5/10.5	93.6/6.4	87.6/12.5
At	97.1/2.9	92.1/7.8	97.0/3.0	94.5/5.5	At	93.4/6.6	89.0/11.0	93.2/6.8	89.4/10.5
AS	97.4/2.6	92.3/7.7	97.1/2.9	95.1/4.9	AS	94.1/5.7	91.1/8.9	93.9/6.1	90.8/8.9
	N,src	N,dst	S,src	S,dst		S,src	S,dst	N,src	N,dst
IP	93.8/6.2	95.7/4.3	94.7/5.3	95.8/4.2	IP	94.4/5.6	92.6/7.4	91.1/8.9	97.2/2.8
Pf	94.7/5.3	90.2/9.8	94.9/5.1	93.7/6.3	Pf	90.8/9.2	89.7/10.3	90.1/9.9	87.0/13.0
At	95.3/4.7	92.0/8.0	95.6/4.4	94.3/5.7	At	91.3/8.7	89.2/10.8	90.6/9.4	88.9/11.3
AS	96.0/4.0	92.2/7.8	95.8/4.2	94.9/5.1	AS	92.8/7.2	91.3/8.7	91.9/8.0	90.6/9.4

to long-tailed size distributions. However, our prior work [25] presents evidence that some long-tailed distributions can arise from multiplicative coalescence, a process in which the probability of joining two objects is proportional to a power of their sizes' product.

6 Concluding Remarks

The rich hierarchy of categories used in IP traffic analysis yields many aggregated measures that can serve as foundations for differentiating typical from rare and extreme. Many of these measures are mutually exclusive, which can affect research conclusions unless the disjointness, in particular diversity/disparity and similar phenomena, are explicitly considered.

In this paper we suggested size disparity as a unifying paradigm shared by seemingly unrelated phenomena: burstiness, scans, floods, flow lifetimes and volume elephants. We pointed out that in general an aggregated measure has a meaning of node degree in some graph. We then discussed concentration properties of byte and packet measures aggregated by IP address, prefix, policy atom and AS. We found that an attempt to faithfully quantify diversity/disparity in Tier 1 backbone data leads to combinatorial explosion of the parametric space. To reduce the description complexity, we introduced a mice-elephant boundary called *crossover*. We showed that many IP traffic aggregation categories have crossovers above the proverbial 80/20 split, mostly around 95/5. We also found that the Pareto models, previously used for file/connection/transfer sizes [18] and short-term prefix traffic volumes [5], require a significant bent ($\alpha \sim 0.5$) to account for the size disparity of aggregated and accumulated backbone traffic. On the other hand, a Weibull distribution with shape parameter 0.2-0.3 can serve as an alternative model for the tails of AS volume data.

Due to space limitations we could not include all analyses done for this study. Our results, including geotraffic volumes, diversity of objects that contribute over 1% of traffic, consumers of fixed (50, 90, 95, 99) traffic percentile volumes,

crossover fractions and cutoffs, volume of mice and distribution plots (all for bytes and packets) are available at [21].

We intend to extend this study of ‘volume elephants’ to the ‘elephants of stability’, i.e. flows with low variation that are important for traffic engineering [14] [5]. We plan to investigate disparity measures based on Shannon’s entropy notion, and to find which natural category of traffic aggregates can fill the (two orders of magnitude by packet or byte volume) gap between IP addresses and prefixes. Finally we hope that crossovers, introduced here for the first time, will serve as a bridge between academic and operational parts of networking community, combining a mathematically precise value on one side with a familiar concept on the other.

Acknowledgements. Thanks to Dan Andersen and Ken Keys for keeping the Backbone Trace Data Kit and analysis software in excellent shape, to Patrick Verkaik for MPLS data, and to Khushboo Shah for discussions of scans.

References

1. A.Shaikh, J.Rexford, Shin, K.G.: Load-sensitive routing of long-lived IP flows. In: SIGCOMM. (1999)
2. kc claffy: Internet traffic characterization (1994) Ph.D. thesis, UCSD.
3. N.Brownlee, kc claffy: Understanding Internet traffic streams: Dragonflies and Tortoises. In: IEEE Communications. (2002)
4. K.Lan, J.Heidemann: On the correlation of Internet flow characteristics (2003) Report ISI-TR-574, www.isi.edu/trpublic/pubs/au-kclan.html.
5. Papagiannaki, K., Taft, N., Diot, C.: Impact of flow dynamics of traffic engineering principles. In: INFOCOM. (2004)
6. I.Graham, M.Pearson, J.Martens, S.Donnelly: Dag - a cell capture board for ATM measurement systems (1997) wand.cs.waikato.ac.nz.
7. Endace: Measurement Systems (2004) www.endace.com.
8. C.Estan, S.Savage, G.Varghese: Automatically inferring patterns of resource consumption in network traffic. In: SIGCOMM. (2003)
9. Odlyzko, A.M.: Privacy, economics, and price discrimination on the Internet. In: ICEC ACM. (2003)
10. R.Mahajan, S.M.Bellovin, Floyd, S., J.Ioannidis, V.Paxson, S.Shenker: Controlling High Bandwidth Aggregates in the Network. In: CCR. (2002)
11. Claffy, K., Braun, H.W., Polyzos, G.: A Parametrizable methodology for Internet traffic flow profiling. In: IEEE JSAC. (1995)
12. Broido, A., kc claffy: Analysis of Route Views BGP data: policy atoms (2001) Proceedings of NRDM, Santa Barbara.
13. Moore, D., Keys, K., Koga, R., Lagache, E., kc claffy: CoralReef software suite as a tool for system and network administrators. In: Usenix LISA. (2001)
14. N.Feamster, J.Borkenhagen, J.Rexford: Guidelines for interdomain traffic engineering. In: CCR. (2003)
15. Broido, A., E.Nemeth, kc claffy: Spectroscopy of private DNS update sources. In: WIAPP03. (2003)
16. Korn, G.A., Korn, T.M.: Mathematical handbook for scientists (1968)

17. R.Knott: Fibonacci numbers and the golden section (2003)
www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fib.html.
18. Crovella, M.E., Bestavros, A.: Self-similarity in World Wide Web traffic. Evidence and possible causes. In: IEEE/ACM Transactions on Networking. (1997)
19. W.Simpson: PPP in HDLC-like Framing. In: RFC1662. (1994)
20. D.Moore, Voelker, G., S.Savage: Inferring Internet Denial-of-Service Activity. In: USENIX Security Symposium. (2001)
21. Supplement: (2004) www.caida.org/analysis/workload/diversity.
22. Broido, A., E.Nemeth, kc claffy: Internet expansion, refinement and churn (2002) ETT 13.
23. G.U.Yule: A mathematical theory of evolution. In: Phil. Trans. Roy. Soc. London Ser. B, 213:21–87. (1924)
24. Mitzenmacher, M.: A brief history of generative models for power law and lognormal distributions. In: Internet Mathematics. (2003)
25. A.Broido, kc claffy: Analysis of routing and topology data (2001)
www.caida.org/outreach/presentations/routingtopology2001/.

Origins of Microcongestion in an Access Router

Konstantina Papagiannaki¹, Darryl Veitch², and Nicolas Hohn^{2*}

¹ Intel Corporation

² University of Melbourne

Abstract. Using an authoritative data set from a fully instrumented router at the edge of a core network, packet delays through an access link are studied in detail. Three different root causes of delay are identified and discussed, related to: unequal link bandwidth; multiplexing across different input links; and traffic burstiness. A methodology is developed and metrics are defined to measure the relative impacts of these separate, though inter-related, factors. Conclusions are given regarding the dominant causes for our representative data set.

1 Introduction/Motivation

Recent studies have shown that backbone networks are highly over-provisioned, and so inflict very little loss or delay on packets traversing them. For example, despite core routers with output buffers capable of holding on the order of 1 second of data, delays rarely exceed millisecond levels [1]. When examined on fine time-scales however, during localised periods of congestion, or ‘microcongestion episodes’, delays can still reach levels which are of concern to core network providers bound by Service Level Agreements (SLAs).

Typically backbone networks are structured in a hierarchy, where link bandwidths decrease as one moves from the long haul links connecting different Points of Presence (PoPs) (currently OC-192), through those interconnecting core routers within a PoP (OC-48 to OC-192), down to access links connecting customers to access routers (OC-3, OC-12 or gigabit Ethernet). The access links, being closer to the edge of the network, are more interesting to study from the delay perspective for two reasons. First, the list of potential causes of delays in a network widens as we move toward the edge. Second, an access link is typically managed by the customer. SLAs therefore do not apply and the link may be run at higher load levels to lower costs, again increasing the potential for congestion.

The aim of this work is to examine in detail the causes of microcongestion episodes in an access router leading away from the core, with a particular emphasis on delays. Although a full separation is not possible, there are nonetheless different generic ‘causes’ or mechanisms of congestion in general, and delay in particular, which can be identified. Briefly, these are related to: i) Reduction in

* This work was done when K. Papagiannaki, D. Veitch and N. Hohn were with the Sprint Advanced Technology Laboratories, in Burlingame, CA, USA.

dina.papagiannaki@intel.com, dveitch@sprintlabs.com, n.hohn@ee.mu.oz.au

link bandwidth from core to access, ii) Multiplexing of multiple input streams, iii) Degree and nature of burstiness of input traffic stream(s). To our knowledge a taxonomy of congestion on an access link (or indeed any link) along these lines has not been studied previously. In particular we seek to answer the question, “What is the dominant mechanism responsible for delays?”, in such a context. More generally, a knowledge of the relative importance of different causes of higher than usual delay, and their interactions, gives insight into how delays may evolve in the future, and not only for the access router we study here. Accordingly, one of our main contributions is a methodology and a set of metrics which can be used more generally to this end.

We first flesh out the mechanisms above in more detail in the next section. We then give a description of our data and experimental setup in section 3. We describe our results in section 4 and summarise in section 5.

2 Congestion Mechanisms

Fundamentally, all congestion is due to one thing – too much traffic. The different mechanisms above relate to different ways in which traffic can be built up or concentrated, resulting in a temporary shortage of resources in the router. To explain the mechanisms precisely, we must have a model of router operation, as it is the router which will multiplex traffic arriving from different high speed links, and deliver it (in this case) to the lower speed output link.

In recent work [2] we looked at the modelling question in fine detail, using the comprehensive data set described in the next section. More specifically, we studied the through-router delays suffered by packets destined for a given output interface in a modern store and forward router. A model was developed which consists of two parts: a fixed minimum delay $\Delta(L)$ dependent upon packet size L which models the front end of the router and the transmission across the switch fabric, and a FIFO queue which models the output buffer and serialisation. We showed that for a store and forward router where the output buffer is the bottleneck, predicted through-router delays follow the measured ones very precisely. We also showed that, as expected, the FIFO queue part of the model dominates the delay dynamics. We will use this model below both conceptually and to generate many of the actual results. We ignore option packets here, which can have much larger delays but which are very rare.

In the framework of the model, *microcongestion* can now be precisely understood as the statistics of delays suffered during *busy periods*, which are time intervals where the system is continuously busy, but idle to either side. Here by ‘system’ we mean a given output interface and the portion of the router, leading from the input interfaces, related to it. Note however that packets are deemed to arrive to the system only after they have *fully* arrived to one of the input interfaces involved. For an input link, we will use ‘busy period’ in a different but related sense, to refer to a train of back-to-back packets (corresponding to a busy period of the output link of the router upstream). We can now discuss the three mechanisms.

Bandwidth Reduction. Clearly, in terms of average rate, the input link of rate μ^i could potentially overwhelm the output link of rate $\mu^o < \mu^i$. This does not happen for our data over long time scales, but *locally* it can and does occur. The fundamental effect is that a packet of size p bytes, which has a width of p/μ^i seconds on the input wire, is stretched to p/μ^o seconds at the output. Thus, packets which are too close together at the input may be ‘pushed’ together and forced to queue at the output. In this way busy periods at the input can only worsen: individually they are all necessarily stretched, and they may also then meet and merge with others. Furthermore new busy periods (larger than just a single packet) can be created which did not exist before. This stretching effect also corresponds, clearly, to an increase in link utilisation, however it is the small scale effects, and the effect on delay, that we emphasise here. Depending on other factors, stretching can result in very little additional delay, or significant buildups.

Link Multiplexing. For a given output link, input traffic will typically arrive from different traffic streams over different input links. This is particularly the case given the Equal Cost MultiPath (ECMP) routing currently deployed by network providers for load balancing purposes. Whether these streams are correlated or not, the superposition of multiple streams increases the packet ‘density’ at all scales and thereby encourages both the creation of busy periods at the output, and the inflation of existing ones. To first order this is simply an additive increase in utilisation level. Again however, the effect on delays could either be very small or significant, depending on other factors.

Burstiness. It is well known that traffic is highly variable or bursty on many time-scales. The duration and amplitude (the highest degree of backlog reached) of busy periods will depend upon the details of the packet spacings at the input, which is another way of saying that it depends on the input burstiness. For example packets which are already highly clustered can more easily form busy periods via the bandwidth induced ‘stretching’ above. To put it in a different way, beyond the first order effect of utilisation level, effects at second order and above can have a huge impact on the delay process.

3 Experimental Setup

We made measurements on a fully instrumented access router inside the Sprint IP backbone network. Of its 6 links, 4 were destined to customers at OC-3 and OC-12 speeds, while the other 2 connected to 2 different backbone routers inside the same PoP at OC-48 rate.

The first 44 bytes of *every* packet seen on *every* link attached to the router were captured, together with a GPS synchronised timestamp accurate to at least $5\ \mu s$. Using the methodology proposed in [1], the packets common to any two links were identified, and from the timestamps the through-router delays (with minor exceptions) were determined for each packet.

With one exception, every link on the router had an average link utilisation below 50% and thus experienced low congestion, and in particular low delays

(99.26% were below 0.5ms). The exception, which we study in detail here, was an access link at OC-3 rate fed from the two OC-48 backbone links, where average utilisations measured over 5-minute intervals reached as high as 80%. Busy periods on this link lasted up to 15 ms, and resulted in maximum through-router delays as high as 5 ms.

In this work we study 13 hours of data comprising more than 129,000 busy periods. The completeness of this data set, and the analysis methodology, allows us to measure in fine detail the evolution of busy periods both on the input links and in the router itself. We can therefore empirically answer essentially any question regarding the formation and composition of busy periods, or on the utilisation and delay aspects of congestion, that we wish. In queueing terminology we have full access to the entire sample path of both the input and output processes, and the queueing system itself. An example of the sample path of the queue workload at the output covering over 3 busy periods is given in Figure 1.

4 Results

As mentioned above, we know from our previous work how to accurately model delays across the monitored router. We therefore use this model to run ‘semi-experiments’ (see [3] for details of this concept) as needed, where virtual scenarios are explored using real input traffic data fed to the physical router model, to help us quantify the contributions of the three mechanisms.

The experiments take the following form. First, a ‘total’ traffic stream S_T is selected. It is fed through the model with output rate μ , and the locations and characteristics of all the resulting busy periods are recorded. Note that even in the case when S_T is the full set of measured traffic, we still must use the model to determine when busy periods begin and end, as we can only measure the system when packets arrive or depart, whereas the model operates in continuous time.

For a given busy period we denote its starting time by t_s , its *duration* by D , its *amplitude*, that is the maximum of the workload function (the largest of the delays $\{d_j\}$ suffered by any packet), by A , and let t_A be the time when it occurred. When we need to emphasise the dependence on the stream or the link bandwidth, we write $A(S_T, \mu)$ and so on.

We next select a substream S_S of traffic according to some criteria. We wish to know the extent to which the substream contributes to the busy periods of the total stream. We evaluate this by feeding the substream into the model, since the detailed timing of packet arrivals is crucial to their impact on busy period shape and amplitude. The focus remains on the busy periods of the total stream even though the substream has its own busy period structure. Specifically, for each busy period of S_T we will look at the contribution from S_S appearing in the interval $[t_s, t_A]$ during which it was building up to its maximum A . Exactly how to measure the contribution will vary depending upon the context.

It is in fact not possible in general to fully separate the congestion mechanisms, as the busy period behaviour is a result of a detailed interaction between

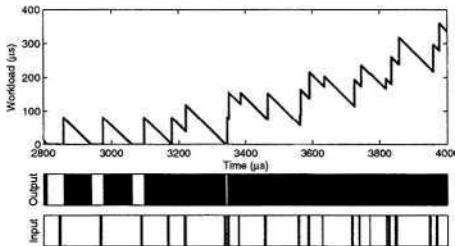


Fig. 1. The bandwidth reduction effect. Bottom Input/Output ‘bar’ plots: busy periods on the OC-48 input and OC-3 output links. Top: resulting queueing workload process. The output busy periods are longer and far fewer than at the input.

all three. The extent to which separation is feasible will become apparent as the results unfold.

4.1 Reduction in Bandwidth

To fix ideas, we illustrate the first mechanism in Figure 1. The two ‘bar plots’ visualise the locations of busy periods on the OC-48 input link (lower bar), and the resulting busy periods following transmission to the smaller OC-3 output link (upper bar). For the latter, we also graph the corresponding system workload induced by the packets arriving to the busy period, obtained using the model. We clearly see that the input busy periods - which consist typically of just one or two packets, have been stretched and merged into a much smaller number of much longer busy periods at the output.

In order to quantify the “stretching” effect we perform a virtual experiment where the total traffic stream S_T is just one of the main input OC-48 streams. By restricting to just a single input stream, we can study the effect of link bandwidth reduction without interference from multiplexing across links.

In this case our ‘sub-stream’ is the same as the total stream ($S_S = S_T$), but evaluated at a different link rate. We quantify “stretching and merging” using the normalised *amplification factor*

$$AF = \frac{A(S_T, \mu_o)}{\max_k A_k(S_T, \mu_i)} \frac{\mu_o}{\mu_i},$$

where $AF \geq 1$. The amplitude for the substream is evaluated across all k busy periods (or partial busy periods) that fall in $[t_s, t_A]$.

In simple cases where packets are well separated, so that all busy periods at both the input and output consist of just a single packet, then stretching is purely linear and $AF = 1$. If queueing occurs so that non-trivial busy periods form at the output, then $AF > 1$. The size of AF is an indication of the extent of the delay increase due to stretching. If the utilisation at the output exceeds 1 then theoretically it will grow without bound.

We present the cumulative distribution function for AF in Figure 2 for each of the main input streams separately. Less than 5% of the busy periods are

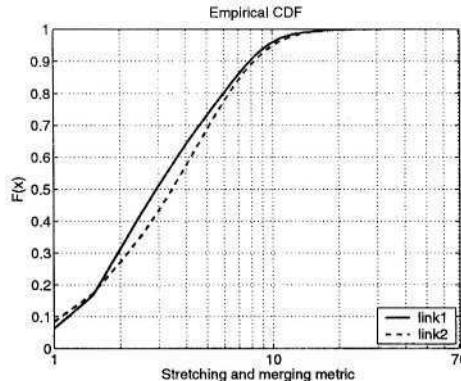


Fig. 2. Empirical distribution functions of AF for the OC-48 input streams.

in the ‘linear’ regime with minimal delay detected via $\text{AF} = 1$. The majority are significantly amplified by the non-linear merging of input busy periods into larger output busy periods. If instead we had found that in most cases that AF was close to 1, it would have been an indication that most of the input traffic on that link was shaped at OC-3 rate upstream.

To get a feeling for the size of the values reported in Figure 2, note that a realistic upper bound is given by $\text{AF} = 240000$, corresponding roughly to a 500ms buffer being filled (in a single busy period) by 40 byte packets well separated at the input, that would induce a maximum workload of $129 \mu\text{s}$ when served at OC-48 rate. A meaningful value worthy of concern is $\text{AF} = 1030$, corresponding to delays of 20ms built up from 375 byte packets, the average packet size in our data.

4.2 Link Multiplexing

To examine the impact of multiplexing across different input links, we let the total stream S_T be the full set of measured traffic. The rampup period, $[t_s, t_A]$, for two busy periods of S_T are shown as the topmost curves in Figures 3 and 4. We select our substreams to be the traffic from the two OC-48 backbone links, S_1 and S_2 . By looking at them separately, we again succeed in isolating multiplexing from the other two mechanisms in some sense. However, the actual impact of multiplexing is still intimately dependent on the ‘stretch transformed’ burstiness structure on the separate links. What will occur cannot be predicted without the aid of detailed traffic modelling. Instead, we will consider how to measure what *does* occur, and see what we find for our data.

Figures 3 and 4 show the delay behaviour (consisting of multiple busy periods) due to the separate substreams over the rampup period. The nonlinearity is striking: the workload function is much larger than the simple sum of the workload functions of the two input substreams, although they comprise virtually all of the total traffic. For example in Figure 3 the individual links each contribute

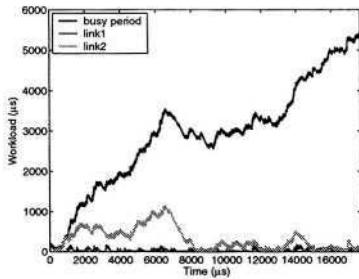


Fig. 3. Effect of multiplexing on the formation of busy periods (from t_s to t_A).

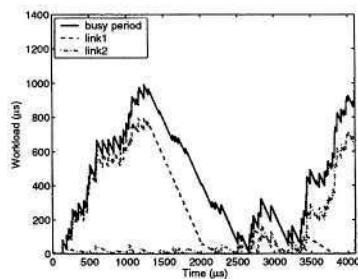


Fig. 4. A ‘bimodal’ busy period, assessing the contribution to A is ambiguous.

less than 1ms of workload at worst. Nonetheless, the multiplexed traffic leads to a significantly longer ramp-up period that reaches more than 5ms of maximum workload at t_A on the right of the plot.

To quantify this effect we define the “link multiplexing” ratio

$$\text{LM} = \frac{\max_k A_k(S_i, \mu_o)}{A(S_T, \mu_o)},$$

which obeys $\text{LM} \in [0, 1]$. Values close to zero indicate that the link has a negligible individual contribution. Therefore if all substreams have small values, the non-linear effect of multiplexing is very strong. In contrast, if $\text{LM} \approx 1$ for some link then it is largely generating the observed delays itself, and multiplexing *may* not be playing a major role. Large values of LM are in fact subject to ambiguity, as illustrated in Figure 4, where the ratio is large for both links. The busy period has a bimodal structure. The first mode is dominated by link 1, however its influence has died off at time t_A , and so is not significantly responsible for the size of A .

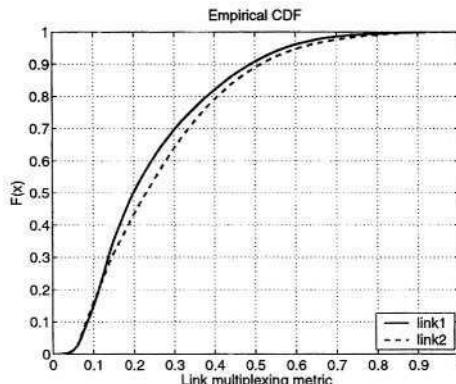


Fig. 5. Empirical distribution functions of LM for the OC-48 input streams.

The results for the data are presented in Figure 5. In more than 95% of the busy periods traffic from each individual link contributes to less than 60% of the actual busy period amplitude. Therefore, it appears that multiplexing is an important factor overall for the delays experienced over the access link.

4.3 Flow Burstiness

There are many definitions of burstiness. It is not possible to fully address this issue without entering into details which would require traffic models, which is beyond the scope of this paper. We therefore focus on burstiness related to 5-tuple flows, to investigate the impact that individual flows, or groups of flows, have on overall delays. We begin by letting the total traffic S_T be that from a single link, to avoid the complications induced by multiplexing.

In order to obtain insight into the impact of flow burstiness we first select as a substream the ‘worst’ individual flow in S_T in the simple sense of having the largest number of packets in the rampup period $[t_s, t_A]$. Two extreme examples of what can occur in the rampup period are given in Figures 6 and 7. In each case the busy period amplitude is large, however the flow contribution varies from minimal in Figure 6, to clearly dominant in Figure 7.

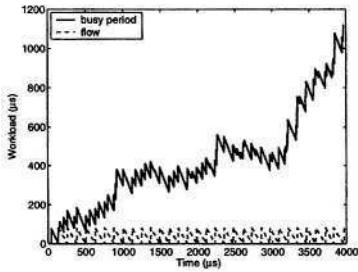


Fig. 6. Flow with multiple packets and no significant impact on the queue buildup.

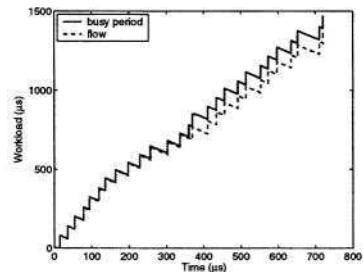


Fig. 7. Flow with multiple packets that dominates the busy period.

To refine the definition of worst flow and to quantify its impact we proceed as follows. For each busy period in the total stream we classify traffic into 5-tuple flows. We then use each individual flow S_j as a substream and measure the respective $A(S_j, \mu_o)$. The worst or “top” flow is the one with the largest individual contribution. We define “flow burstiness” as

$$FB = \max_j \frac{\max_k A_k(S_j, \mu_o)}{A(S_T, \mu_o)},$$

where as before the inner maximum is over all busy periods (or partial busy periods) of the relevant substream falling in $[t_s, t_A]$. The top flow may or may not be the one with the greatest number of packets.

Our top flow metric takes values $FB \in (0, 1]$. If FB is close to zero then we know that **all** flows have individually small contributions. Alternatively if FB is

large then, similarly to LM, there is some ambiguity. We certainly know that the top flow contributes significant delay but in case of bimodality this flow may not actually be responsible for the peak defining the amplitude. In addition, knowledge about the top flow can say nothing about the other flows.

We present the cumulative distribution function for FB in Figure 8 for each of the OC-48 links. For more than 90% of the busy periods the contribution of the top flow was less than 50%. In addition for 20% of the busy periods the contribution of the top flow was *minimal* (for example as in Figure 6), that is it was the smallest possible, corresponding to the system time of a single packet with size equal to the largest appearing in the flow.

If the top flow has little impact, it is natural to ask if perhaps a small number of top flows together could dominate. One approach would be to form a stream of the n largest flows in the sense of FB. However, as the choice of n is arbitrary, and it is computationally intensive to look over many values of n , we first change our definition to select a more appropriate substream. We define a flow to be *bursty* if its substream generates a packet delay which exceeds the minimal delay (as defined above) during the rampup period. Note that only very tame flows are not bursty by this definition! We denote by S_b the substream of S_T that corresponds to **all** bursty flows, and compute the new flow burstiness metric:

$$\text{FB}' = \frac{\max_k A_k(S_b, \mu_o)}{A(S_T, \mu_o)}.$$

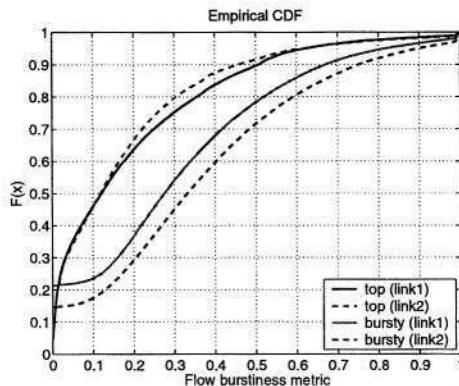


Fig. 8. Empirical distribution functions of FB for the OC-48 input streams.

As before, $\text{FB}' \in [0, 1]$, and its value can be interpreted in an analogous way to before. The difference is that, as the substream is much larger in general, a small value is now extremely strong evidence that individual flows do not dominate. Note that it is possible that no flow is bursty, in which case $\text{FB}' = 0$, and therefore that the top flow is not necessarily bursty. This has the advantage of avoiding the classification of a flow as dominant, thereby giving the impression

that it is bursty in some sense, simply because a trivial flow dominates a trivial busy period.

Our results are presented in Figure 8. As expected, the contribution of S_b to the busy period is more significant: in 20% of cases it exceeds 60%. On the other hand, 20% of the busy periods had FB' equal or close to zero, indicating that they had no bursty flows. Indeed, we found that only 7.7% of all flows in our dataset were classified as “bursty” according to our definition. Only in a very small number of cases does the top or the subset of bursty flows account for the majority of the workload (for example as in Figure 7). Consequently, it seems that in today’s network flow dynamics have little impact on the delays experienced by packets in core networks.

5 Summary

We have studied in detail the origins of packet delays flowing toward an access link, and clarified the role of three different mechanisms, related to: unequal link bandwidth; multiplexing across different input links; and traffic burstiness.

Our first contribution was methodological. We showed how a router model can be used to investigate the role of the different mechanisms, and defined metrics to help assess their impact. The possible values of the metrics, and how they can be interpreted, was discussed.

Our second contribution was to investigate the actual contributions in todays access networks, via a comprehensive and representative data set. We focused on an OC-3 access link fed mainly by two OC-48 links carrying roughly 50% of the traffic each. The link was not highly congested (no packet drops over 13 hours and packet delays all under 6ms), however it was much more congested than typical core links. We found that the link bandwidth reduction factor of 16 (from OC-48 to OC-3) played a significant role in delay buildups (non-trivial amplification factor AF), indicating that traffic is bursty, and not significantly shaped at OC-3 rates upstream. Multiplexing was also found to be significant (small multiplexing fraction LM), as in most cases the traffic on the individual links could not individually induce delays which were a large fraction of the observed delays. Finally the effect of individual 5-tuple flows, and sets of ‘bursty’ flows, was found to be small in most cases (small flow burstiness ratio FB), leading to the noteworthy conclusion that 5-tuple flow dynamics are not responsible for excessive packet delay in today’s core networks. These conclusions are strongly traffic dependent. The methodology and metrics we define can be used to monitor traffic evolution, and are especially effective when one wishes to confirm a hypothesis that a given effect (such as individual flow impact) is negligible.

References

1. Papagiannaki, K., Moon, S., Fraleigh, C., Thiran, P., Tobagi, F., Diot, C.: Analysis of measured single-hop delay from an operational backbone network. In: IEEE Infocom, New York (2002)

2. Hohn, N., Veitch, D., Papagiannaki, K., Diot, C.: Bridging router performance and queuing theory. In: Proceeding of ACM Sigmetrics Conference on the Measurement and Modeling of Computer Systems, New York, USA (2004)
3. Hohn, N., Veitch, D., Abry, P.: Cluster processes, a natural language for network traffic. IEEE Transactions on Signal Processing, special issue on “Signal Processing in Networking” **51** (2003)

Performance Measurement and Analysis of H.323 Traffic*

Prasad Calyam¹, Mukundan Sridharan², Weiping Mandrawa¹, and Paul Schopis¹

¹ OARnet, 1224 Kinnear Road, Columbus, Ohio 43212.

{pcalyam, wmandraw, pschopis}@oar.net

² Department of Computer and Information Science,
The Ohio State University, Columbus, OH 43210.

sridhara@cis.ohio-state.edu

Abstract. The popularity of H.323 applications has been demonstrated by the billions of minutes of audio and video traffic seen on the Internet every month. Our objective in this paper is to obtain Good, Acceptable and Poor performance bounds for network metrics such as delay, jitter and loss for H.323 applications based on objective and subjective quality assessment of various audio and video streams. To obtain the necessary data for our analysis we utilize the H.323 Beacon tool we have developed and a set of Videoconferencing tasks performed in a LAN and also with end-points located across multiple continents, connected via disparate network paths on the Internet.

1 Introduction

H.323 [1] is an umbrella standard that defines how real-time multimedia communications, such as audio and video-conferencing, can be exchanged on packet-switched networks (Internet). With the rapid increase in the number of individuals in industry and academia using H.323 audio and video-conferencing systems extensively, the expectation levels for better audio and video performance have risen significantly. This has led to the need to understand the behavior of audio and video traffic as it affects end user perceived quality of the H.323 applications over the Internet. Several studies have been conducted [2,3,4] and many approaches [5,6,7] have been proposed to determine the performance quality measures of H.323 applications. Many of the previous studies used pre-recorded audio and video streams and aimed at obtaining quality measures either based solely on network variations or on various audiovisual quality assessment methods.

Our primary focus in this paper is to understand how the various levels of network health, characterized by measuring delay, jitter and loss, can affect end user perception of audiovisual quality. By systematically emulating various network health scenarios and using a set of Videoconferencing ‘Tasks’ we determine performance bounds for delay, jitter and loss. The obtained performance bounds

* This work was supported in part by The Ohio Board of Regents and Internet2

are mapped to end-users perceptions of the overall audiovisual quality and are then categorized into Grades such as ‘Good’, ‘Acceptable’ and ‘Poor’. We show that end-users are more sensitive to variations in jitter than variations in delay or loss. The results of this paper could provide ISPs and Videoconferencing Operators a better understanding of their end-user’s experience of audiovisual quality for any given network health diagnostics.

To obtain the necessary data to support the various conclusions in this paper, we utilized the H.323 Beacon tool [8] we have developed and a set of Videoconferencing tasks. Over 500 one-on-one subjective quality assessments from Videoconferencing Users and the corresponding H.323 traffic traces were collected during our testing, which featured numerous network health scenarios in an isolated LAN environment and on the Internet. The collected traces provided objective quality assessments. The Internet testing involved 26 Videoconferencing end-points; each performing 12 Videoconferencing tasks and located across multiple continents, connected via disparate network paths that included research networks, commodity networks, cable modem connections, DSL modem connections and Satellite networks.

The rest of the paper is organized as follows: Section 2 provides a background pertaining to this paper, Section 3 describes our testing methodology, Section 4 discusses our analysis of the performance bounds for delay, jitter and loss and Section 5 concludes the paper.

2 Background

2.1 H.323 System Architecture

There are numerous factors that affect the performance assessments of H.323 applications. These factors can be subdivided into 3 categories: 1. Human factors 2. Device factors 3. Network factors. First, Human factors refer to the perception of quality of the audio and video streams and also the human error due to negligence or lack of training which results in performance bottlenecks which then affect the performance assessments. Secondly, essential devices such as H.323 terminals, Multipoint Control Units (MCUs), gatekeepers, firewalls and Network Address Translators (NATs) frequently contribute towards performance degradations in H.323 systems. Thirdly, the network dynamics caused by route changes, competing traffic and congestion cause performance bottlenecks that affect performance assessments. In this paper we are interested in studying aspects of the Human factors, which deal with end-user perception of audiovisual quality and the Network factors, which contribute to any network’s health. The reader is referred to [9] for details related to Device factors.

2.2 Audiovisual Quality Assessment Metrics

There are two popular methods to assess audiovisual quality: Subjective quality assessment and Objective quality assessment. Subjective quality assessment

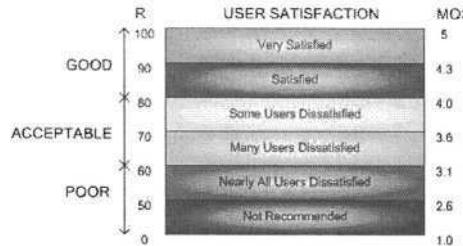


Fig. 1. Voice Quality Classes

involves playing a sample audiovisual clip to a number of participants. Their judgment of the quality of the clip is collected and used as a quality metric. Objective quality assessment does not rely on human judgment and involves automated procedures such as signal-to-noise ratio (SNR) measurements of original and reconstructed signals and other sophisticated algorithms such as Mean Square Error (MSE) distortion, Frequency weighted MSE, Segmented SNR, Perceptual Analysis Measurement System (PAMS) [10], Perceptual Evaluation of Speech Quality (PESQ) [11], and Emodel [5], to determine quality metrics. The problem with subjective quality assessment techniques is that human perception of quality is based on individual perception, which can vary significantly between a given set of individuals. The problem with objective quality assessment techniques is that they may not necessarily reflect the actual end-user experience. There have been studies [12] that show that when objective and subjective quality assessment are performed simultaneously, the results are comparable.

In our study, we employ both the subjective and objective quality assessment methods to determine end-user perception of audiovisual quality for various network health scenarios. To obtain subjective quality assessment scores from the participants, we extended the slider methodology presented in [7] and developed our own slider that was integrated into our H.323 Beacon tool. Participants ranked the audiovisual quality on a scale of 1 to 5 for various Videoconferencing tasks using what is basically the Mean Opinion Score (MOS) ranking technique. To obtain objective quality assessment scores we utilized the Telchemy VQMon tool [12] that implements the Emodel and uses traffic traces obtained for the various Videoconferencing tasks as an input for its analysis. The Emodel is a well established computational model that uses transmission parameters to predict the subjective quality. It uses a psycho-acoustic R-scale whose values range from 0 to 100 and can be mapped to MOS rankings and User Satisfaction as shown in Fig. 1. Though the Emodel fundamentally addresses objective quality assessment of voice, our collected data shows reasonable correlation of the subjective quality assessment scores for audiovisual quality provided by the participants and the objective quality assessment scores provided by VQMon. The reader is referred to [2,5,12] for more details relating to Emodel components.

2.3 Network Performance Metrics

The variables that affect the MOS rankings the most in H.323 system deployments are the dynamic network changes caused by route fluctuations, competing traffic and congestion. The network dynamics can be characterized by 3 network metrics viz. delay, jitter and loss as specified in [13].

Delay is defined as the amount of time that a packet takes to travel from the sender's application to the receiver's destination application. The components that contribute to the end-to-end delay include: (a) compression and transmission delay at the sender (b) propagation, processing and queuing delay in the network and (c) buffering and decompression delay at the receiver. The value of one-way delay needs to be stringent for H.323 audio and video traffic to sustain good interaction between the sender and receiver ends. It is recommended by [14] that delay bounds for the various grades of perceived performance in terms of human interaction can be defined as: Good (0ms-150ms), Acceptable (150ms-300ms), Poor ($> 300ms$).

Jitter is defined as the variation in the delay of the packets arriving at the receiving end. It is caused due to congestion at various points in the network, varying packet sizes that result in irregular processing times of packets, out of order packet delivery, and other such factors. Excessive jitter may cause packet discards or loss in the playback buffer at the receiving end. The playback buffer is used to deal with the variations in delay and facilitate smooth playback of the audio and video streams. There have been some indications in [15] about jitter bounds, which have been verified to be approximately correct in our earlier work [9] and are also supported by the studies conducted in this paper. However, there have not been well defined rules of thumb to suggest the accurate jitter bounds in terms of the various grades of H.323 application performance. Our studies suggest the following jitter values to be reasonably reliable estimates to determine the grade of perceived performance: Good (0ms-20ms), Acceptable (20ms-50ms), Poor ($> 50ms$).

Lastly, loss is defined as the percentage of transmitted packets that never reach the intended destination due to deliberately discarded packets (RED, TTL=0) or non-deliberately by intermediate links (layer-1), nodes (layer-3) and end-systems (discards due to late arrivals at the application). Though popular experience suggests loss levels greater than 1% can severely affect audio-visual quality, there have not been well defined loss bounds in terms of the various grades of H.323 application performance. Our studies in this paper suggest the following loss values to be reasonably reliable estimates to determine the grade of perceived performance: Good (0%-0.5%), Acceptable (0.5%-1.5%), Poor ($> 1.5\%$).

3 Test Methodology

3.1 Design of Experiments

Our approach in determining the performance bounds for delay, jitter and loss in terms of Good, Acceptable and Poor grades of performance, is to view the

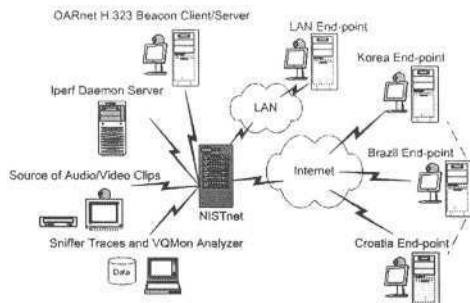
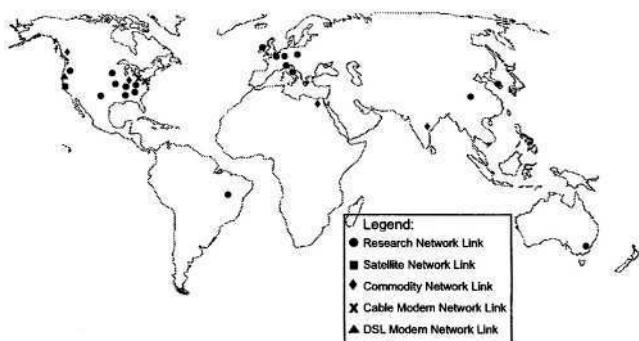
network health as an outcome of the combined interaction of delay, jitter and loss. Our reasoning is that all three network parameters co-exist for every path in the Internet at any given point of time; regulating any one of these parameters affects the other parameters and ultimately the Quality of Service (QoS) perceived by the end-user in terms of H.323 application performance. A real-world example is illustrated in [16] where, resolving a loss problem in an Intercampus DS3 led to a decrease in the observed loss but unexpectedly led to an increase in the overall jitter levels in the network. This shows that network health needs to be sustained in a stable state where the network delay, jitter and loss are always within the Good performance bounds. In our design of experiments, we employ a full factorial design for the 3 factors, i.e. we emulate 27 scenarios that cover every possible permutation involving the various delay, jitter and loss levels.

We performed extensive LAN tests that covered all of the 27 possibilities and selected 9 scenarios shown in Table 1 of Section 4 for Internet measurements whose results in essence reflected the results of the 27 scenarios. The reason to use the 9 scenarios is that it is impractical and non-scalable to have each participant judging quality 27 times in a single subjective assessment test session.

For each of the 9 Internet test scenarios a Videoconferencing task was assigned. A Videoconferencing task could be any activity that takes place in a routine Videoconference. A casual conversation, an intense discussion, or a class lecture would qualify as a Videoconferencing task. There is significant literature recommending strategies for tasks that could be part of Subjective and Objective assessments of audiovisual quality [6,7]. All of them recommend that in addition to passive viewing for assessments of audiovisual quality, the participants must be presented with realistic scenarios. Key guidelines proposed in the above literature were followed in task creation, participant training for scoring the audiovisual quality, task ordering and overall environment setup for the assessment. A subset of the Videoconferencing tasks performed by the test participants involved the audio and video loop back feature of the H.323 Beacon tool. The loopback feature enables local playback of remote H.323 Beacon client audio or video recorded at a remote H.323 Beacon server. The reader is referred to [8] for more details relating to the H.323 Beacon.

3.2 Test Setup

To obtain the performance bounds for delay, jitter and loss and to affirm our conclusions, we chose a two phase approach. In the first phase we performed extensive testing by emulating all the 27 scenarios in a LAN environment and obtained the performance bounds for delay, jitter and loss as stated in Section 2.3. In the second phase, we used the 9 scenarios described in Section 3.1 and performed the Internet tests. In both the phases of testing, we conducted one-on-one testing with participants at each of the LAN/Internet sites and collected traffic traces and objective and subjective quality assessments. Fig. 2 shows the overall test setup and Fig. 3 shows the participating sites in the testing and their last mile network connections. NISTnet [17] network emulator was used to create the various network health scenarios by introducing delay, jitter and loss

**Fig. 2.** Overall Test Setup**Fig. 3.** World Map Showing the Test Sites Involved

in each path under test. Spirent SmartBits [18] technology was used to qualify whether NISTnet accurately introduced the delay, jitter and loss settings.

The difference between the LAN and Internet tests in terms of the NISTnet settings was that in the LAN environment the end-to-end delay, jitter and loss were completely controlled by the NISTnet settings; whereas, in the Internet the network paths already had inherent values of delay, jitter and loss. Therefore a network path characteristics pre-determination step was required for each test site before configuring additional end-to-end delay, jitter and loss on NISTnet. To accurately obtain the inherent network path characteristics, we used data from OARnet H.323 Beacon, NLANR Iperf and appareNet developed by Apparent Networks. The end-to-end delay, jitter and loss values configured on the NISTnet for the Internet tests were the values obtained by deducting the inherent path characteristics values from the LAN NISTnet settings.

4 Analysis of Performance Bounds

Sites with research network connections traversing Abilene and GEANT backbones had more consistent network path characteristics and overall results. This is in accord with the popular opinion about the superior performance levels of these backbones owing to the fact that there is limited cross traffic and low utilization levels on these networks. In contrast, sites that had commodity Internet connections, cable modem and DSL modem last mile connections and also academic sites such as Brazil and Korea had significant variations in network path characteristics and contributed the most to the variance in the overall results. Fig. 4 - 6 show the subjective and objective MOS rankings obtained during the testing for delay, jitter and loss values used in different scenarios. The variance observed in the results is contained well within the performance bounds values of delay, loss and jitter stated in Section 2.3. This clearly demonstrates that our LAN results are scalable to the Internet. Also the MOS values plotted in Fig. 4 - 6 include the values obtained for the tasks involving the H.323 Beacon. This proves the handy utility of the H.323 Beacon in determining user perceived audiovisual quality in H.323 production systems without remote end-user intervention.

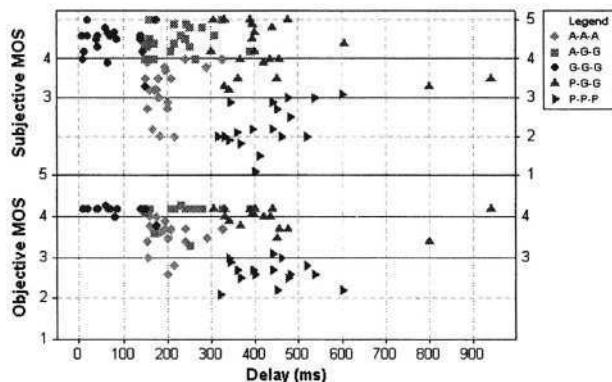


Fig. 4. Subjective and Objective MOS Vs Delay

The correlation between objective and subjective scores was observed to be in the above average to strong range. The Pearson correlation for delay, jitter and loss were 0.827, 0.737, and 0.712 respectively. The reason for all the correlations not being strong can be due to the following reasons: we used the Emode1 objective scores that are modeled after only audio traffic streams; participants at the end-points involved in the testing were from very diverse backgrounds that included demographics of Videoconferencing Coordinators, Network Engineers, Graduate Students, Instructors and IT Managers; and various types of

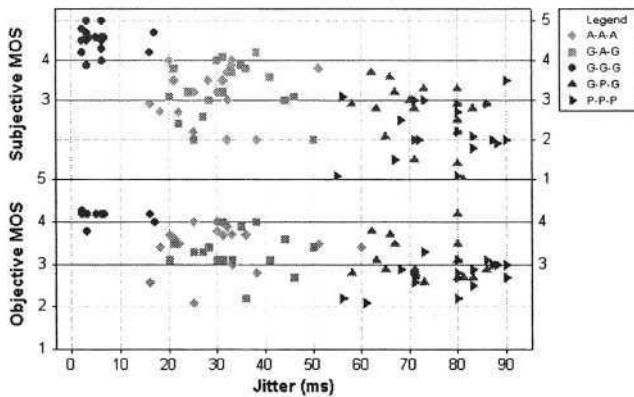


Fig. 5. Subjective and Objective MOSS Vs Jitter

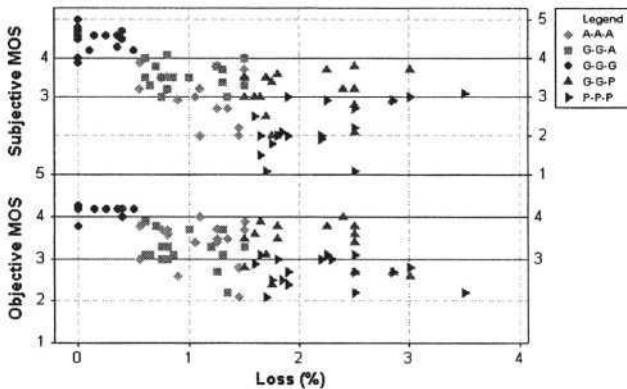


Fig. 6. Subjective and Objective MOS Vs Loss

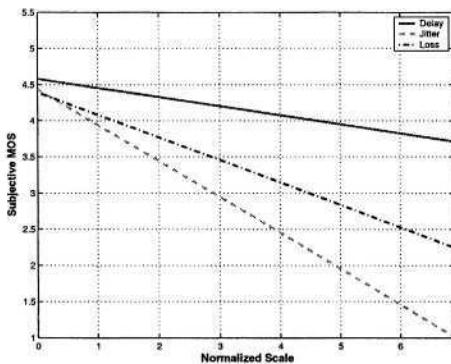
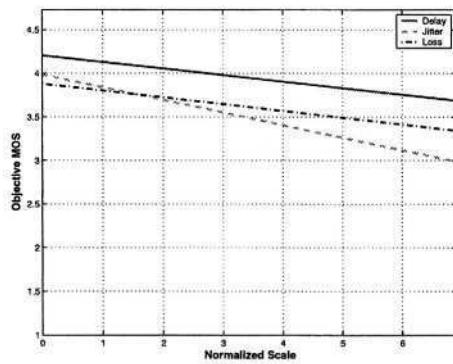
hardware codec/software codec H.323 end-points were used in the testing. The various audio codecs observed at the endpoints were GSM, G.711 and G.722 and the various video codecs observed were H.261, H.262 and H.263.

Table 1 summarizes all the Subjective and Objective Quality Grade Assessments for the LAN/Internet tests. A '*' next to a paired-grade in the Result column indicates that an end-user will more often perceive that Grade of quality rather than the Grade without the '*' in that particular scenario. Values in the Result column that have more than one Grade and no '*' imply that there is an equal chance of an end-user perceiving either of the Grades of quality for that scenario.

In the pursuit to identify the most dominating factor amongst delay, jitter and loss that affects end-user perception of audiovisual quality, we normalized the scales of these factors and plotted them against both the Subjective and Objective MOS assessments as shown in Fig. 7 and 8. Each unit in the nor-

Table 1. Results of Quality Grade Assessments for LAN/Internet Tests

	S1	S2	S3	S4	S5	S6	S7	S8	S9
Delay	G	A	P	G	G	P	G	G	A
Jitter	G	A	P	G	P	G	G	A	G
Loss	G	A	P	P	G	G	A	G	G
Result	G	A/P	P	A/P	A/P	G*/A	A	A*/P	G

Legend: $G \rightarrow \text{Good}$, $A \rightarrow \text{Acceptable}$, $P \rightarrow \text{Poor}$, $(S1 - S9) \rightarrow \text{Scenarios 1-9}$ **Fig. 7.** Effects of Normalized Delay, Jitter and Loss variations on Subjective MOS**Fig. 8.** Effects of Normalized Delay, Jitter and Loss variations on Objective MOS

Normalized scale corresponds to a delay of 150ms, jitter of 20ms and loss of 0.5%. We can observe that end-user perception of audiovisual quality is more sensitive to changes in jitter than to changes in delay and loss. In fact, the changes in delay have low impact on the end-user's perception of the audiovisual quality; although delay values more than 3 units on the normalized scale are deemed to be unsuitable for interactive communications [14].

5 Conclusion and Future Work

In this paper, we determined the performance bounds for network metrics such as delay, jitter and loss. We use these bounds to determine the impact of network health on end-user perception of audiovisual quality of H.323 applications. By emulating various network health scenarios both in the LAN and on the Internet and by using realistic Videoconferencing tasks, we show that end-user perception of audiovisual quality is more sensitive to the variations in end-to-end jitter than to variations in delay or loss. In the Internet tests, by considering almost every

possible last-mile connection, we demonstrated that the results we obtained in the LAN tests scaled consistently to the Internet.

With the valuable network traces obtained from our one-on-one testing with various sites across the world we are currently studying the effects of jitter buffer sizes on packet discards under various network conditions using many popular audio and video codecs. We are also investigating effects of various packet sizes on the end-user perception of audiovisual quality of H.323 applications. The results of our studies could serve as trouble-shooting information during periods of suspected network trouble affecting H.323 audio and video-conferences. They can also foster broader understanding of the behavior of audio and video traffic over the Internet which can then lead to better designed networks in the future.

References

1. ITU-T Recommendation H.323, Infrastructure of audiovisual services- Systems and terminal equipment for audiovisual services, Series H: Audiovisual and multimedia systems. (1999)
2. Markopoulou, A., Tobagi, F., Karam, M.: Assessment of VoIP quality over Internet backbones. In: Proceedings of IEEE Infocom. (2002)
3. Marsh, I., Li, F.: Wide area measurements of Voice Over IP Quality. SICS Technical Report T2003:08 (2003)
4. Eurescom reports of Europe-wide experimentation of multimedia services (1999)
5. ITU-T Recommendation G.107, The Emodel, a computational model for use in transmission planning. (1998)
6. ITU-T Recommendation P.911, Subjective audiovisual quality assessment methods for multimedia applications. (1998)
7. Mullin, J., Smallwood, L., Watson, A., Wilson, G.: New techniques for assessing audio and video quality in real-time interactive communications. IHM-HCI Tutorial (2001)
8. OARnet H.323 Beacon: a tool to troubleshoot end-to-end h.323 application performance problems. (<http://www.itec ohio.org/beacon>)
9. Schopis, P., Calyam, P.: H.323 traffic characterization study. OARnet Technical Report submitted to American Distance Education Consortium (2001)
10. ITU-T Recommendation P.800, Methods for subjective determination of transmission quality. (1996)
11. ITU-T Recommendation P.862, An objective method for end-to-end speech quality assessment of narrowband telephone networks and speech codecs. (2001)
12. Clark, A.: Modeling the effects of burst packet loss and recency on subjective voice quality. (2001)
13. Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V.: RTP: A transport protocol for real-time applications. RFC 1889 (1996)
14. ITU-T Recommendation G.114, One Way Transmission Time. (1996)
15. Kelly, B.: Quality of service in internet protocol (ip) networks (2002)
16. Indraji, A., Pearson, D.: The network: Internet2 commons site coordinator training (2003)
17. NISTnet network emulation package. (<http://snad.ncsl.nist.gov/itg/nistnet/>)
18. Spirent SmartBits network measurement suite. (<http://www.spirentcom.com>)

Measurements and Laboratory Simulations of the Upper DNS Hierarchy

Duane Wessels¹, Marina Fomenkov², Nevil Brownlee², and kc claffy²

¹ The Measurement Factory, Inc.

wessels@measurement-factory.com

² CAIDA, San Diego Supercomputer Center, University of California, San Diego
{marina, nevil, kc}@caida.org

Abstract. Given that the global DNS system, especially at the higher root and top-levels, experiences significant query loads, we seek to answer the following questions: (1) How does the choice of DNS caching software for local resolvers affect query load at the higher levels? (2) How do DNS caching implementations spread the query load among a set of higher level DNS servers? To answer these questions we did case studies of workday DNS traffic at the University of California San Diego (USA), the University of Auckland (New Zealand), and the University of Colorado at Boulder (USA). We also tested various DNS caching implementations in fully controlled laboratory experiments. This paper presents the results of our analysis of real and simulated DNS traffic. We make recommendations to network administrators and software developers aimed at improving the overall DNS system.

1 Background

The Domain Name System (DNS) is a fundamental component of the modern Internet [1], providing a critical link between human users and Internet routing infrastructure by mapping host names to IP addresses. The DNS hierarchical name space is divided into zones and coded in the widespread “dots” structure. For example, *.com* is the parent zone for *microsoft.com*, *cnn.com*, and approximately 20 million other zones.

The DNS hierarchy’s root zone is served by 13 nameservers, known collectively as the DNS root servers. However, the root server operators use various techniques to distribute the load among more than 13 servers. Perhaps the oldest technique is a load-balancing switch for servers at the same site. The most recent technique is IP anycast. We estimate that, by now, there are actually close to 100 physical DNS root servers, even though there are only 13 root server addresses [2].

Just under the root are servers for the Internet’s 260-or-so top-level domains (TLDs), such as *.com*, *.nz*, and *.us*. TLD servers are similar to the roots in their requirements, architecture and traffic levels. In fact, many of them are significantly busier. The TLD zones contain referrals to second-level domains (SLDs), and so on.

Why does the Internet need so many root and TLD servers? Is this simply a fact of life, or have we inadvertently created a monster? One of the reasons that we need so many is to carry the normal request load, including a high proportion of unanswerable requests from poorly configured or buggy DNS resolvers. Another is to provide resilience against increasingly common distributed denial of service attacks targeting the system. As the

top of the DNS hierarchy becomes more distributed, it is harder for attackers to affect the service.

In this paper, we focus on the behavior of various DNS caching implementations. The behavior of DNS caches is important because they generate almost all of the queries to authoritative (root, TLD, SLD, etc) servers on the Internet. Authoritative servers do not query each other. Stub resolvers should always send their queries to caching nameservers, rather than talk directly to the authoritative servers.

2 Measurements of Live DNS Traffic

For a case study of workday DNS load in academic networks we used the NeTraMet tool [3] to capture samples of DNS traffic (Table 1) at the University of California San Diego (UCSD), USA, the University of Colorado (UCol), USA and the University of Auckland (UA), New Zealand. We configured NeTraMet to select only DNS packets to/from the root and gTLD¹ nameservers; these form a small set of high-level nameservers with known IP addresses. Each DNS sample provides the following information: time (in ms) when a query was sent or a response was received, which root or gTLD server it was sent to, the source host IP address, and a query ID. The UA and UCol samples (but not UCSD) include query types. The most recent samples collected in December 2003 also captured top level and second level domains for each query.

Table 1. Samples of DNS traffic to root and gTLD servers

Samples	UCSD Feb 03	U. of Auckland Sep 03	U. of Auckland Dec 03	U. Colorado Dec 03
Query rates to roots, per min	214	29	10	9
Query rates to gTLDs, per min	525	67	76	70
Number of query sources	147	19	42	1
Sample duration, hrs	48	64.5	157.3	157.2
Median response times (from roots)	5–200	60–340	5–290	27–180
% due to 3 highest users	58	85	74	100
% of A queries	n/a	23	70	81

The number of source hosts addressing the roots and/or gTLDs in the UCSD and UA data is larger than we expected. However, the distribution of queries among sources is highly skewed. Typically, the bottom half of sources produce < 1 % of the overall query load while a small number (3–5) of top sources is responsible for 80% or more of the total traffic. Network administrators need to exercise proper control of those few *high level sources* that dominate external DNS traffic and to optimize the software those source hosts use. At the same time, end user hosts should be configured to send DNS requests to (internal) local nameservers, rather than to root and TLD servers directly.

¹ The gTLDs are: *com*, *net*, *org*, *edu*, *mil*, *int*, and *arpa*.

Typically, the DNS traffic is dominated by A queries (hostname-to-address lookups) as in our December samples. This observation is in general agreement with other measurements (cf. [4], [5]). The first sample of UA DNS traffic was unusual because two of the three top users generated only PTR queries (address-to-hostname lookups). With the exception of these two sources, A queries constitute 58% of the remaining traffic.

2.1 Response Time and Server Selection

We found that in all samples the root server response time distributions has a long tail, except for G root, which is bimodal. Table 1 shows the range of median root server response times experienced by the busiest query source in each data set. For all nameservers at the same location response time distributions are very similar since the response time correlates with the geographical distance from the point of measurements to a given root server [6]. Nowadays, due to proliferation of anycast root server instances, the exact geographical location of a given root server has become a moot point.² Response time actually correlates with the distance to the closest anycast node. For example, the UA nameservers have a median response time of only 5 ms from the anycast F root node in New Zealand.

As we show in the next section, different resolvers utilize different nameserver selection algorithms. For the live data, we studied the probability of selecting a given root server versus its median response time (Figure 1). For each query source, response times were normalized by the minimum response time to the closest root server. When possible, we verified the software being used by hosts at each site. UCol was running BIND8, while both UA nameservers shown were running BIND9. For the UCol and UA#2 sources the probability is approximately inversely proportional to the response time, as expected. However, the UA#1 sent its queries (predominantly PTR) to the root servers in nearly uniform manner ignoring the latency. The UCSD#1 nameserver was using some Windows software and we were not able to obtain any indicative information about UCSD#2. It appears that both UCSD hosts more or less ignore response times when choosing which root servers to use.

2.2 Query Rates, Duplication, and Loss of Connectivity

We analyzed our most detailed samples from December 2003 in order to estimate the contribution of repeated queries to the total traffic. Table 2 compares the behavior of A queries to root and gTLD servers sent by the UCol nameserver and by the busiest UA source. For both of them A queries make up about 80% of the total traffic they generate.³

For both sources, the bulk of A queries to gTLD servers receive positive answers and, on average, there are 3 to 4 queries per each unique SLD. However, only 2% of A queries sent to root servers by UCol host were answered positively, with an average of 6.7 queries per each unique TLD. This result is in agreement with findings of [7]

² [2] gives locations of anycast nodes for the root servers. Unfortunately there is no straightforward way to determine which server instance any particular DNS reply packet came from.

³ Next most frequent types of queries are: for the UA source – PTR, 11 %, for the UCol source – SOA, 9%.

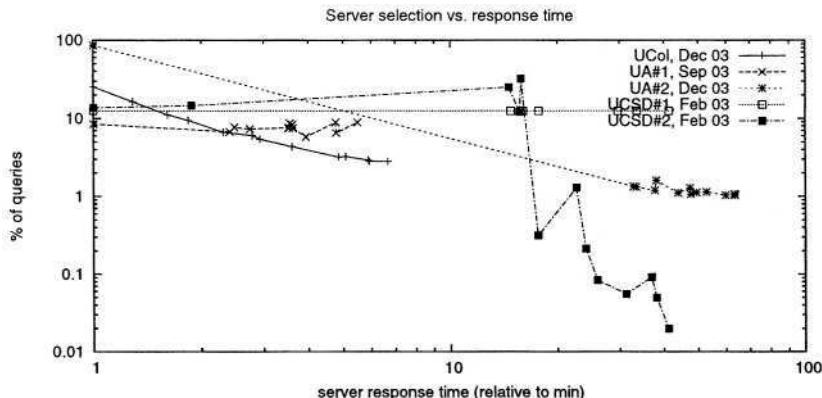


Fig. 1. Distributions of queries among root servers vs. response time. Each point on the plots shows percentage of queries sent to a given root server; the points are plotted in order of increasing response time.

which showed that only 2% of queries reaching root servers are legitimate. The average repetition rate corresponds to about one query per TLD per day which seems reasonable. At the same time, queries for invalid domains seem to be repeated unnecessarily.

In contrast, the busiest UA nameserver received 77.7% of positive answers from the roots. The average number of queries per TLD is 125.4. For example, there were 5380 queries for *.net* or approximately one query every two minutes. Each one received a positive answer and yet the server continued asking them.

Table 2. Statistics of A queries

Samples, 6.5 days long		U. Auckland		U. Colorado	
Queries to		roots	gTLDs	roots	gTLDs
positive answers, %	77.7%		94.7%	2.1%	96.3%
av. # of q. per domain	125.4		3.0	6.7	4.4
most frequent	<i>.net</i> , 5380	<i>iwaynetworks.com</i> , 2220		<i>.mil</i> , 293	<i>.needsomeadealz.com</i> , 6014
negative answers, %	20.6%		1.9%	97.1%	3.4%
most frequent	<i>.cgs</i> , 325	<i>.colframe.com</i> , 134		<i>.dnv</i> , 13660	<i>jclnt.com</i> , 7694

Our data indicate that repeated queries constitute a considerable fraction of the overall load and that both positive and negative caching can be improved. Using the actual query load observed, we simulated an ideal case when each query would be asked only once: (a) it is not repeated impatiently before the answer comes back, and (b) the answer, whether negative or positive, is cached and the same query is not repeated until its TTL expires. The results (Table 3) indicate that proper timing and caching of queries possibly can reduce the load by a factor of 3 or more. However, this simulation obviously is too optimistic since some repetition of queries to root/TLD servers is unavoidable, for

Table 3. Simulated query rates

Samples	U. Auckland		U. Colorado	
Queries to	roots	gTLDs	roots	gTLDs
real traffic	96,759	722,265	86,229	658,784
TTL = 3 h	20,833	306,566	30,243	308,002
TTL = 24 h	10,140	211,914	26,165	229,080

example in a case when queries for *aaa.foo.com* and *bbb.foo.com* immediately follow each other and neither *.com* nor *foo.com* are cached.

Loss of connectivity is another cause of repeated queries. In all our measurements, the loss of packets was very low, typically < 0.5%. However, there was a one hour period of connectivity loss in the University of Auckland data on 6 Dec 03, during which their nameservers sent queries but no answers came back. The query rate increased more than 10-fold, but quickly subsided back to normal levels as soon as connectivity resumed (Figure 2).

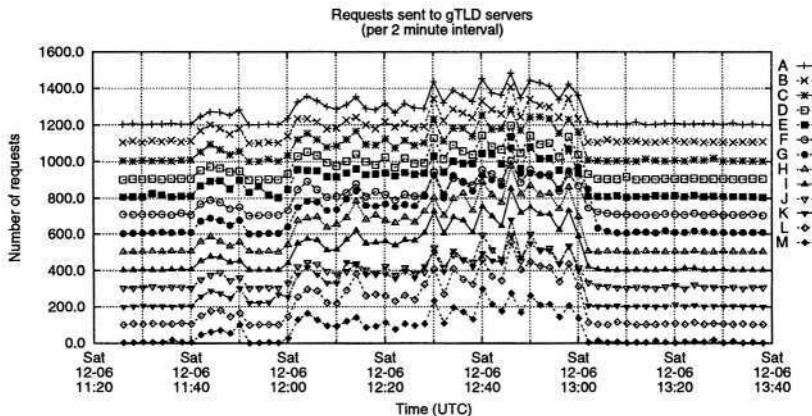


Fig.2. Query rates to gTLD servers. Each subsequent plot is shifted up by 100. There were two periods of connectivity loss when no answers were received: from 11:40 till 11:50 and from 12:00 till 13:00. During these periods query rates increased considerably.

3 Laboratory Simulations

3.1 Experimental Setup

We set up a mini-Internet in our laboratory to study the behavior of DNS caches. Do they distribute the load as they should? What happens when the cache cannot talk to any root servers? In our setup (Figure 3), three systems on top mimic the authoritative root, TLD, and SLD servers. These systems all run BIND8. Another system, “Cache” in the

middle, runs the various DNS caches that we test, one at a time. The bottom system, “User”, generates the test load of end-user queries.

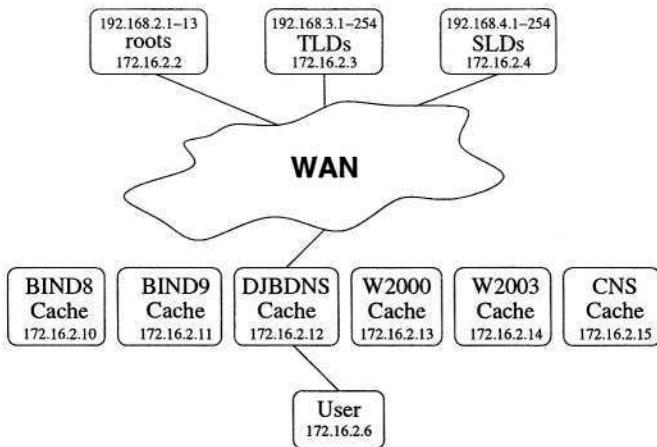


Fig. 3. Network setup for DNS tests. The “User” system sends DNS queries to only one of the caches in each test. The cache recursively resolves queries for the user by talking to the authoritative root, TLD, and SLD servers. For some tests we introduce wide-area packet loss and transmission delays with FreeBSD’s dumynet.

Networking and Addressing. It is somewhat of a stretch to replicate the massive DNS system with only five computers. We use a few tricks to make this a little more realistic. We gave the root, TLD, and SLD authoritative servers multiple IP addresses (13, 254, and 254, correspondingly), bound to the loopback interface. Thus, each server’s external interface acts like a router—a way to reach the loopback addresses. The caching nameserver, and the User system have one address each. All systems are attached to a single 100baseTX Ethernet segment. We use FreeBSD’s Dumynet feature to introduce simulated packet loss and delays for some tests.

Trace Data. To drive the simulation, we created a trace file of DNS requests by collecting hostnames from 12 hours of the IRCache HTTP proxies [8] data. The trace file contains 5,532,641 DNS requests for 107,777 unique hostnames. There are 70,365 unique second-level zones, and 431 top-level zones (many of them bogus). The trace contains many repeated queries as necessary to test the caching part of the DNS cache.

We also take timestamps from the proxy logs and replay the trace at the same rate, preserving as closely as possible the time between consecutive queries. Thus, each test run takes 12 hours to complete.

Zone Files. Our scripts generate BIND zone files based on the contents of the trace file. We do not generate zones for bogus TLDs. Thus, we can also test negative caching.

For the root and TLD zones, we try to mimic reality as closely as possible. For example, we use the same number of nameserver IP addresses and the same TTLs for NS and glue records.⁴ To get the real values, we issued the necessary queries out to the Internet while building the zone file data.

We also try to match our simulated SLD parameters to reality. There are too many SLD zones (more than 100,000) and querying all of them would take much too long. Instead, we obtained the necessary values (the number of A records per name and the TTLs for A, NS, and CNAME records) from a random sample of 5000 domains and generated similar random values for the SLD zones in the trace. Unlike reality, each simulated SLD zone has only two NS records. We also determined that about 35% of the names in the trace file actually point to CNAME records, and our simulated zone data mimics this as well.

Tested Configurations. We tested the following six different DNS caches using their default configuration parameters.

1. BIND 8.4.3
2. BIND 9.2.1
3. dnscache 1.05, aka DJBDNS with CACHESIZE set to 100,000,000
4. Microsoft Windows 2000 v5.0.49664
5. Microsoft Windows 2003 v5.2.3790.0
6. CNS 1.2.0.3

For each cache, we ran tests for six different network configurations:

1. No delays, no loss
2. 100 millisecond delays, no loss
3. linear delays, no loss
4. linear delays, 5% loss
5. linear delays, 25% loss
6. 100% loss

The no-delay/no-loss configuration is simple, but unrealistic. For most real users, the root/TLD/SLD servers are between 30 and 200 milliseconds away (cf. Table 1). The 100ms-delay/no-loss test uses a constant 100-millisecond delay to each root/TLD/SLD server, but with no packet loss. It is also somewhat unrealistic.

In the next three configurations, with so-called linear delays, a nameserver's latency (in ms) is proportional to its order n in the nameserver list: $\tau = 30 + 10n$. For zones such as the root and *com*, which have 13 nameservers, the last one is 160 milliseconds away from the cache. This arrangement provides some pseudo-realistic diversity and allows us to see how often a DNS cache actually selects a nameserver with the best response time. We believe the linear-delay/5%-loss test to be the most realistic among all six configurations.

The final configuration has 100% packet loss. This test mimics a situation when a DNS cache cannot communicate with any authoritative nameservers. Reasons for such non-communication include firewalls, packet filters, unroutable source addresses, and

⁴ Glue records are, essentially, A records for nameservers of delegated zones.

saturated network connections. Live measurements showed that when the cache’s queries reach an authoritative nameserver, but the replies do not make it back, the DNS traffic increases (cf. Figure 2).

3.2 Results

General statistics. Figure 4 shows how many queries each DNS cache sent in various tests. For example, the leftmost bar shows that in the no-delay/no-loss test, BIND8 sent a small number of queries to the roots, about 400,000 queries to the TLDs, and about 600,000 to the SLDs. Its total query count is just over 1,000,000.

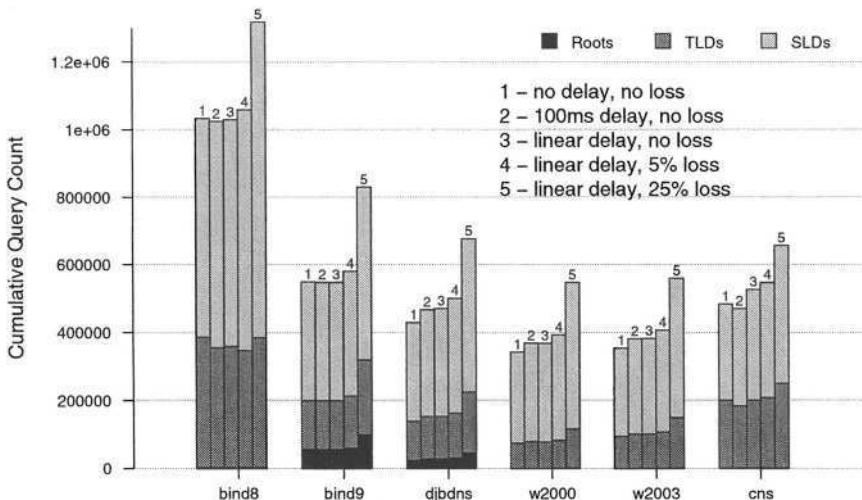


Fig. 4. Cumulative query counts for all caches.

BIND8 always sends more queries than any of the other caches, primarily because it sends three queries (A, A6, and AAAA) to the roots/TDLs/SLDs for each of the expired nameserver addresses for a given zone. Recall that every SLD zone in our model has two nameservers, while root and TLD zones usually have more. This result implies that the number of nameservers and their address record TTLs can have a significant impact on DNS traffic.

The BIND9 results show the largest percentage of root server queries. For the no-loss tests, BIND9 sends 55,000 queries to the roots, versus only 1800 for BIND8. The reason for this is because BIND9 always starts at the root when querying for expired nameserver addresses and sends an A and A6 query for each nameserver.

The DJBDNS data also shows a relatively high fraction of root server queries, which is about half of the BIND9 numbers. DJBDNS also starts at the root when refreshing expired nameserver addresses. However, it only sends a single A query for one of the nameservers, not all of them.

The two Windows results are very similar, with slightly more queries from W2003. These two send the fewest overall number of queries for the five test cases shown.

CNS fits in between DJBDNS and BIND9 in terms of total number of queries. Due to its low root server query count, we can conclude that it does not start at the root for expired nameserver addresses. Also note that, like BIND8, CNS sends slightly fewer queries for the 100-millisecond delay test (#2), than it does for the no-delay test (#1).

Root Server Query Counts. Table 4 shows counts of queries to the roots in each test. As discussed above, except for the 100% loss tests, BIND9 and DJBDNS hit the roots much harder than the others. The BIND8 numbers would probably be closer to Windows if it did not send out A6 and AAAA, in addition to A queries.

Table 4. Number of Messages sent to roots.

Delays Pkt Loss	none	100ms	linear	linear	linear	
	none	none	none	5%	25%	100%
BIND8	1,826	1,876	1,874	1,899	2,598	37,623,266
BIND9	55,329	55,260	55,256	59,222	99,422	2,564,646
DJBDNS	24,328	27,646	27,985	30,341	44,503	12,155,335
W2000	622	657	663	727	1,020	66,272,276
W2003	693	669	666	709	1,009	39,916,750
CNS	824	831	924	975	1,179	12,456,959

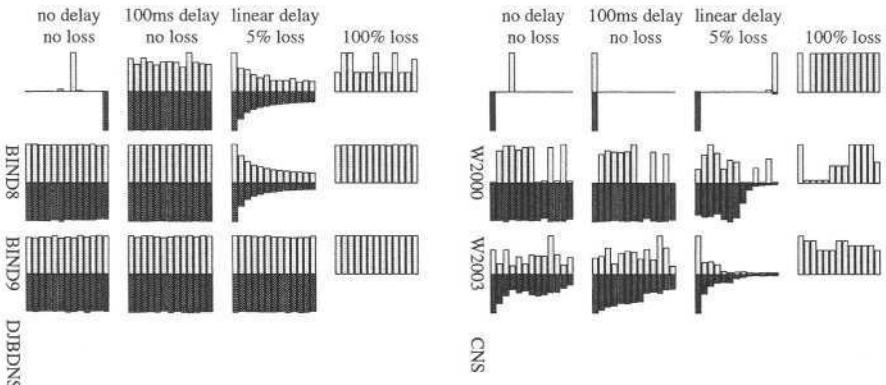
The 100% loss tests are very interesting. Our trace contains 5,500,000 hostnames and that is how many queries the fake user sends to the cache. Except for BIND9, all other caches become very aggressive when they do not get any root server responses. They send out more queries than they receive. On one hand this is understandable because there are 13 root nameservers to reach. However, it is appalling that an application increases its query rate when it can easily detect a communication failure. The worst offender, W2000, actually amplifies the query stream by more than order of magnitude.

BIND9 is the exception in the 100% loss tests. It actually attenuates the client's query stream, but only by about half. BIND9 is the only DNS caching software that has a nifty feature: it avoids repeat queries for pending answers. For example, if user sends two back-to-back queries (with different query-IDs) for `www.example.com`,⁵ most caching resolvers will forward both queries. BIND9, however, forwards only one query. When the authoritative answer comes back, it sends two answers to the user.

Root/COM Nameserver Distribution. Table 5 shows how the caches distribute their queries to the root (lighter bars) and to the *com* TLD server (darker bars) for four of the test cases. BIND8 almost uses a single server exclusively in the no-delay/no-loss test. Since that server always answers quickly, there is no need to try any of the others. In the 100-millisecond delay test, however, BIND8 hits all servers almost uniformly. The test with linear delays and 5% loss has an exponential-looking shape. The first nameservers

⁵ Of course the answer must not already be cached.

Table 5. Distribution of queries to root and *com* TLD nameservers, showing how the caching nameserver distributed its queries in each test. The upper, lighter bars show the histogram for root nameservers. The lower, darker bars show the histogram for the *com* TLD nameservers.



have the lowest delays, and, unsurprisingly, they receive the most queries. The 100% loss test is odd because, for some reason, five of the servers receive twice as many queries as the others.

The BIND9 graphs look very nice. The two no-loss tests, and the 100%-loss test, show a very uniform distribution. Also, the linear-delay/5%-loss test yields another exponential-looking curve, which is even smoother than the one for BIND8.

DJBDNS shows a uniform distribution for all tests. The software does not try to find the best server based on delays or packet loss. This is an intentional design feature.⁶

Windows 2000 has very poor server selection algorithms, as evidenced by the histograms for the 100ms-delay and linear-delay tests. It selects an apparently random server and continues using it. In the 100%-loss test it did query all roots, except the second one for some reason. The Windows 2003 DNS cache is somewhat better, but also shows strange patterns.

CNS also demonstrated odd server selections. To its credit, the linear-delay/5%-loss case looks reasonable, with most of the queries going to the closest servers. In the 100%-loss test, the selection is almost uniform, but not quite.

4 Conclusion

Our laboratory tests show that caching nameservers use very different approaches to distribute the query load to the upper levels. Both versions of BIND favor servers with lower round-trip times, DJBDNS always uses a uniform distribution, Windows 2000 locks on to a single, random nameserver, and Windows 2003 shows an odd, unbalanced distribution. BIND9 and DJBDNS hit the roots much harder than other caches to avoid certain cache poisoning scenarios. BIND8 and BIND9's optimism in looking for IPv6 addresses results in a significant amount of unanswerable queries.

DNS zone administrators should understand that their choice of TTLs affects the system as a whole, rather than their own nameservers. For example, a BIND9 cache

⁶ See <http://cr.yp.to/djbdns/notes.html>

sends two root server queries each time a nameserver address expires. Popular sites can help reduce global query load by using longer TTLs.

Both laboratory tests and live measurements show that caching resolvers become very aggressive when cut off from the DNS hierarchy. We believe that resolvers should implement exponential backoff algorithms when a nameserver stops responding. We feel strongly that more caching implementations should adopt BIND9's feature that avoids forwarding duplicate queries. Other implementations should employ DJBDNS's minimalist strategy of sending only a single A query for expired nameserver glue.

Acknowledgments. Support for this work is provided by WIDE project (<http://www.wide.ad.jp>) and by DARPA grant 66001-01-1-8909.

References

1. Albitz, P., Liu, C.: DNS and BIND. O'Reilly and Associates (1998)
2. RSSAC: Root servers locations (2004) <http://www.root-servers.org/>.
3. Brownlee, N.: NeTraMet - a Network Traffic Flow Measurement Tool (2002) <http://www.caida.org/tools/measurement/netramet/>.
4. Jung, J., Sit, E., Balakrishnan, H., Morris, R.: DNS Performance and the Effectiveness of Caching. In: ACM SIGCOMM Internet Measurement Workshop. (2001)
5. Keys, K.: Clients of dns root servers (private communication) (2002) <http://www.caida.org/~kkeys/dns/2002-08-28/>.
6. T. Lee, B. Huffaker, M. Fomenkov, kc claffy: On the problem of optimization of DNS root servers' placement. In: PAM. (2003)
7. D. Wessels, M. Fomenkov: Wow, That's a Lot of Packets. In: PAM. (2003)
8. IRCache: Information resource caching project (2004) Funded by NSF grants NCR-9616602 and NCR-9521745, <http://www.ircache.net/>.

A Robust Classifier for Passive TCP/IP Fingerprinting

Robert Beverly

MIT Computer Science and Artificial Intelligence Laboratory
rbeverly@csail.mit.edu

Abstract. Using probabilistic learning, we develop a naive Bayesian classifier to passively infer a host’s operating system from packet headers. We analyze traffic captured from an Internet exchange point and compare our classifier to rule-based inference tools. While the host operating system distribution is heavily skewed, we find operating systems that constitute a small fraction of the host count contribute a majority of total traffic. Finally as an application of our classifier, we count the number of hosts masquerading behind NAT devices and evaluate our results against prior techniques. We find a host count inflation factor due to NAT of approximately 9% in our traces.

1 Introduction

Previous measurement studies analyzing Internet packet header traces demonstrate the wealth of information, for instance traffic characteristics and host behavior, available to a passive monitor. In this work we use simple probabilistic learning methods to perform a maximum-likelihood inference of a host’s operating system from packet headers. Drawing upon earlier TCP/IP “fingerprinting” techniques [1] we exploit the subtle differences in network stack implementations that uniquely identify each operating system. Whereas previous tools rely on an exact match from an exhaustive list of TCP settings, we develop a naive Bayesian classifier that provides a continuous degree of identification confidence without deep-packet inspection.

Rule-based approaches fail to identify as many as 5% of the hosts in traces we collect from an Internet exchange point, likely due to users modifying their TCP parameters or employing stack “scrubbers” [2]. In contrast our classifier can intelligently disambiguate under uncertainty.

While fingerprinting is often regarded as a security attack, we argue that our work is motivated by a number of positive and practical applications. Passively determining operating systems is valuable in intrusion detection systems [3], serving operating system specific content, providing security against unauthorized access, compiling network inventories, building representative Internet models and measuring NAT (Network Address Translation) [4] deployment.

This paper presents results from using our classifier to measure the distribution of hosts, packets and bytes per operating system in our exchange point trace captures.

As an application of our classifier we improve upon previous approaches [5] to infer the frequency and behavior of Internet clients behind NAT devices. Understanding the prevalence of hosts masquerading behind NAT devices has important implications [6][7][8] to the Internet address registries, developers of end-to-end applications and designers of next generation protocols.

The remainder of the paper is organized as follows. We present related work in Section 2. Section 3 describes our inference algorithm and methods to train the classifier. In Section 4 we give our measurement results and distribution of different operating systems. Section 5 details approaches to, and results from, NAT inference. We conclude with suggestions for further research and a summary of major findings.

2 Related Work

Several TCP/IP fingerprinting tools exist employing both active and passive techniques. Active methods involve repeatedly probing a remote host with specially crafted packets to induce operating-system-unique responses. Passive methods rely on an ability to observe a remote host's traffic and identify the machine based on normal traffic flow. Passive fingerprinting is non-intrusive and undetectable, but requires an acceptable monitoring point. Conversely, active tools generate traffic, but can be run anywhere in the network.

The most widely used active tool is the freeware nmap [9] program. nmap identifies a remote host by finding open TCP and UDP ports, sending specially formed packets and matching the responses to a database of over 450 signatures.

Our research focuses on large-scale measurement using passive fingerprinting. The freeware p0f tool [1] provided the original impetus for this work. p0f is a rule-based tool containing approximately 200 signatures gathered by the Internet community. The p0f signatures contain the following fields: IP time to live (TTL), presence of the IP don't fragment (DF) bit, initial TCP window size, SYN packet size and TCP options. Each signature maps distinctive fields of the IP and TCP header to different operating systems. For each passively observed TCP SYN packet, p0f searches the signature space for a match. Unfortunately, we find that p0f fails to identify as many as 5% of the hosts in traces we collect from an Internet exchange point.

Bellovin presents a technique for counting hosts behind NAT devices by observing patterns in IP ID sequences [5]. We utilize many of these techniques when performing our NAT measurement, but find several difficulties which we discuss in Section 5.

Because of these difficulties, the only Internet-wide measurements of NAT usage are from a study that attracted clients to a game server [10]. This study found NAT in use among 17% to 25% of the game playing Internet population. The study did not attempt to determine the number of machines masquerading behind each NAT device.

3 Methodology

In this section we detail our inference algorithm, address several practical difficulties and explore different methods of training the classifier.

Passive fingerprinting leverages the fact that different operating systems implement differing TCP/IP stacks, each of which has a unique signature. Even between versions or patches of an operating system there exist subtle differences as developers include new features and optimize performance [11].

Our implementation makes decisions based solely on the TTL, DF, window size and SYN size information which are collectively distinct. The classifier examines the initial TCP SYN packets, but determines the probabilistic likelihood of each hypothesis, i.e. operating system, and selects the maximum-likelihood hypothesis. Thus we always determine a best guess regardless of whether there is an exact signature match.

Before discussing specifics of our implementation, we briefly explain our process of inferring the original TTL of a packet as sent from the source host and mitigate complicating factors with the initial window size.

The IP TTL is decremented by one at each router along the forwarding path. Because our monitoring point is inside the network at an arbitrary distance away from different hosts, a packet's originating TTL as it was sent from a host must be inferred from the observed TTL. We use the well-known heuristic of selecting the next highest power of 2 as the originating TTL. If the observed TTL is greater than 128 we infer an original TTL of 255 and if less than 32 we infer 32.

The initial TCP window size can be fixed in the case of simple stacks or may be a function of the MTU (Maximum Transmission Unit) or MSS (Maximum Segment Size). The MSS is defined as the payload size a packet can carry for the interface MTU. Determining the MSS is complicated in our implementation as we assume visibility into only the first 40 bytes of each packet¹, thereby excluding observation of TCP MSS options. In addition, we assume asymmetrical traffic such that we cannot determine a negotiated MSS. An empirical study of the data to find the most common largest packet sizes revealed five common MSS values: 1460, 1380, 1360, 796 and 536 bytes. The first four are the result of common MTUs and 536 bytes is the default MSS [12]. A further complication arises when the MSS is effectively reduced by the size of the TCP and IP options. We infer the size of the options by taking the size of the SYN packet minus 40 bytes. For each of these five common MSS values, and its equivalent with options, we check if there exists an integer multiple of the window size. If so, the conditional probability of that window size is used.

We wish to evaluate the probability of each operating system hypothesis H_i given the observed data $P(H_i|D)$. Our four observations are: $d_{TTL=ttl}$, $d_{MSS=wss}$, $d_{SYN=syn}$, $d_{DF=\{0,1\}}$. A prediction is made using all the hypotheses weighted by their probabilities. Bayes' rule provides a convenient mechanism to obtain the posterior probability of each hypothesis from causal data. We employ the common simplifying strategy of assuming that each piece of data $d_j \in D$ is statistically independent to obtain the naive Bayesian classifier:

¹ In fact our exchange point monitor captures only the first 40 bytes

$$P(H_i|D) = \frac{\prod_j P(d_j|H_i)P(H_i)}{P(D)} \quad (1)$$

Clearly the fields are not independent of each other, but analysis has shown naive classifiers can perform close to optimal in practice [13]. Note that for classification, the denominator $P(D)$ is the same for each hypothesis and therefore does not need to be explicitly computed for comparisons.

We experiment with two methods to train the classifier. In the first we use the p0f signature file and take all hypotheses prior probabilities $P(H_i)$ as uniform over the hypothesis space. The p0f signatures are the product of a community effort and are highly accurate.

In our second training method, we collect all TCP SYN packets into a web server. We correlate each SYN packet with an HTTP log entry that contains an explicitly stated browser and operating system. To eliminate bias toward any operating system, we gather logs from a non-technical web server.

Web logs provide an attractive automated method to continually train the classifier without maintaining and updating a database of signatures. However, there are two potential sources of error when training the classifier with web logs. The first source of error is due to hosts connecting through proxies. The web log entry contains the host's operating system, but the captured TCP SYN is from the proxy. Secondly, some browsers provide false information in an attempt to provide anonymity or ensure compatibility. With a large enough training set, we maintain that these errors are statistically insignificant. We present measurements comparing results from our classifier trained with signatures and web logs in the next section.

Our classifier has two main advantages. It gives appropriate probabilistic weight to the value of each piece of data based on the training set, making it robust to user-tuned TCP stacks. Secondly, the classifier produces a maximum-likelihood guess along with a degree of confidence even in the face of incomplete or conflicting data.

The major limitation of our classifier is evaluating its accuracy without a large packet trace of known hosts. Traditionally, the performance of a classifier is measured by taking a known data set, training on some fraction of it and measuring the accuracy of the classifier on the remainder of the data. Unfortunately no data set is available on which to measure the accuracy of our classifier. We have a public web page available that captures each visitor's incoming packets, makes an inference and allows the user to correct an incorrect inference. We plan to measure the accuracy of our classifier once we gather a sufficiently large set of results from our web site.

In lieu of formal verification of the classifier, we gather packet traces from machines of all major operating systems. In addition to testing the stock TCP settings, we modify the socket buffer size, turn on window scaling and enable selective acknowledgments where possible. In many cases the rule-based tool could not identify our machines with modified stacks. We ensure that our tool correctly infers all configurations of machines in our test set.

Table 1. Inferred Operating System Distribution (59595 Unique Hosts)

Operating System	Bayesian Percent	WT-Bayesian Percent	Rule-Based Percent
Windows:	92.6	94.8	92.9
Linux:	2.3	1.6	1.7
Mac:	1.0	2.1	1.0
BSD:	1.6	0.0	1.6
Solaris:	0.4	0.5	0.2
Other:	2.1	1.1	1.0
Unknown:			1.6

4 Measurement Results

This section gives the results of our tool on Internet traces. We use our classifier to analyze approximately 38M packets from an hour-long mid-week trace collected at a United States Internet exchange at 16:00 PST in 2003. Within the trace, we analyze only the TCP packets which account for approximately 30.7M packets. Understanding the prevalence of operating systems in the wild allows Internet models to incorporate a realistic distribution of TCP features and provides insight into potential homogeneity implications, e.g. genetic host diversity in the face of epidemic network attacks.

For brevity of presentation, we group operating systems into six broad classifications.² We measure the number of unique hosts of each operating system type and per-operating system packet and byte distributions. We compare results from our Bayesian classifier trained with the p0f signatures (denoted Bayesian in the following tables), the classifier trained with web logs (denoted WT-Bayesian) and p0f (denoted rule-based).

Recall that our tool classifies the first TCP SYN of the three way handshake, i.e. the source of the host initiating communication. Thus in contrast to measurements of server operating system popularity [14], our results are mostly representative of client activity. However we will capture particular server initiated traffic, for instance SMTP.

Table 1 displays the operating system distribution among roughly sixty-thousand unique hosts in the trace. Windows based operating systems as a group dominate among all hosts with a greater than 92% share. Although we do not present the results here as they are not generally applicable, we also classify traces gathered from our academic department’s border router. By comparison we find only 40% popularity among Windows operating systems in our department’s traffic.

We then measure the number of packets attributable to each operating system and again cluster the results into broad classes. For each packet, we determine the operating system of the source based on a prior SYN inference. Surprisingly,

² Apple’s OS-X TCP stack is indistinguishable from BSD, so the results for Mac are under represented, while the BSD results are over represented.

Table 2. Inferred Operating System Packet Distribution (30.7 MPackets)

Operating System	Bayesian Percent	WT-Bayesian Percent	Rule-Based Percent
Windows:	76.9	77.8	77.0
Linux:	19.1	18.7	18.8
Mac:	0.8	1.5	0.8
BSD:	0.8	0.1	1.6
Solaris:	0.7	1.3	0.5
Other:	1.7	0.6	0.2
Unknown:			1.3

Table 3. Inferred Operating System Byte Distribution (7.2 GBytes)

Operating System	Bayesian Percent	WT-Bayesian Percent	Rule-Based Percent
Windows:	44.6	45.2	44.7
Linux:	52.6	52.3	52.4
Mac:	0.5	0.9	0.5
BSD:	0.7	0.1	1.2
Solaris:	0.7	1.1	0.6
Other:	0.9	0.4	0.1
Unknown:			0.7

Table 2 shows that as a fraction of total packets, the Linux operating system contributes substantially: approximately 19% compared to Windows's 77%.

Table 3 depicts the per-operating system distribution of approximately 7.2G bytes. Unexpectedly, Linux operating systems are the largest traffic contributor, with an average of over 2MB of traffic per Linux host in the hour measurement interval. The proportion of byte traffic between Windows and Linux was roughly equal with other operating systems contributing negligibly.

To understand the large discrepancy between operating system counts and the traffic contribution due to each operating system, we process the traces to find the largest flows. The top ten flows contribute approximately 55% of the total byte traffic. Of these ten, we classify five as Linux, two as Windows, one as BSD. The remaining two flows are not classified, and hence do not contribute to the totals in Table 3, because the monitor did not observe an initial SYN from the source.

The largest traffic source is a software distribution mirror running Linux that is continually transferring data with clients and other mirrors. Four Linux machines continually crawling web pages at a rate of one query every 2 to 3 ms are among the top ten byte flows. We also find email SMTP servers, web caches and P2P applications in the ten largest flows. Thus we conclude that Linux is the largest byte traffic contributor, primarily due to server applications.

5 NAT Inference

Next we describe techniques, including using our tool, to detect the number of hosts masquerading behind a NAT-enabled device. To provide an unbiased trace to evaluate different techniques, we create synthetic NAT traces with a known fraction of masquerading hosts. We evaluate the performance and discrimination power of existing techniques, such as IP ID pattern matching, and our classifier in detecting IP masquerading. Finally we present the results of our NAT inference on the exchange point traces.

5.1 Techniques for Counting NAT Hosts

Previous research toward passively counting hosts behind NAT devices centers on two methods. The first technique [15] requires a monitor before the client’s first-hop router, an impossibility for analysis of traces from deep within the network. By assuming the ability to monitor packets at the client attachment point, e.g. a switch or aggregation device, the TTL of packets a monitor observes are not yet decremented by a router and are a standard power of two. For packets with a TTL exactly one less than expected, this method can guess a NAT device is present.

Bellovin provides a more sophisticated approach by observing patterns in IP IDs [5]. Bellovin’s algorithm relies on the IP IDs from individual hosts being implemented as sequential counters. By building sequences of IDs that match within reasonable gap and time bounds, one can infer the actual number of machines in a trace.

However as Bellovin notes, matching sequential IP IDs may fail. The purpose of the IP ID field is to allow fragmentation and reassembly by guaranteeing a sufficiently unique number for the lifetime of the packet. There is no defined semantic to the field and hence no reason why it should be a counter. For example, BSD-based operating systems now implement the IP ID as a pseudo-random number rather than a counter. In addition, if the don’t fragment bit is set, reassembly is not necessary and hence some NAT devices reset the IP ID to zero. Both issues render IP ID sequence matching ineffective.

A few subtleties remain even assuming non-zero, sequential IDs. Sequence matching actually relies on incorrect NAT behavior: a NAT should rewrite IDs to ensure uniqueness but often does not. Of most relevance to the data from our monitoring point, the potential for misidentification increases when monitoring deep in the network. As the amount of traffic increases, the number of IP stacks producing IDs starting from a randomly distributed point increases. Hence the number of ID collisions increases as a function of observation point depth.

5.2 Evaluation of Sequence Matching

In order to evaluate the performance of Bellovin’s method on our traces, we implement our own version of the algorithm. We maintain separate pools of normal and byte-swapped sequences since the IP ID counter may be in network or host byte order. All arithmetic is modulo 2^{16} as follows. The algorithm adds

each new packet to the best matching sequence among the two pools. An ideal match is a sequence whose most recent ID is one less than the packet's ID. A packet's ID and time must be within prescribed thresholds gaps in order to match a sequence. If the gap between a new packet's ID and the last ID of all sequences is greater than the threshold, or the time gap is too great, we create a new sequence. After constructing all of the sequences, we coalesce them. We repeatedly combine sequence pairs with the successively smallest gap up to a threshold. All of our thresholds are the recommended values based on Bellovin's empirical results. Finally, sequences with fewer than 50 packets are discarded. The total number of sequences remaining in the normal and byte-swapped pools estimate the number of hosts in the trace.

Because ID collisions overwhelm the algorithm and produce obviously incorrect results, we take Bellovin's suggestion and augment the IP ID technique to discriminate based on IP address. We build, coalesce and prune sequences separately for each unique address.

To evaluate the performance of our algorithms and their suitability for measuring NAT in our live traffic traces, we create synthetic NAT traffic. We capture anonymized traffic from the border router of our academic building where no NAT is present. We then process the trace to reduce the number of unique addresses by combining the traffic of n addresses into one, where n is the "NAT inflation factor."

Using the campus trace of approximately 2.4M packets, we create a synthetic NAT trace with a NAT inflation factor of 2 so that there is the same traffic as in the original trace, but half the number of unique hosts. Using per IP address sequence matching, we obtain a NAT inflation factor of 2.07. Our experimental result well approximates our control data set and gives merit to the approach.

Finally, we apply Bellovin's per IP address sequence matching to our Internet exchange trace. We find an inflation factor of 1.092, lower than our intuitive sense, but providing an intuition for the lower bound.

5.3 Evaluation of Fingerprint Matching

Next, we explore a second technique to count the number of hosts masquerading behind NAT devices. We use our classifier to detect and count multiple packet signatures originating from a single IP address. We assume affects due to DHCP [16] and dual-boot systems are negligible over the hour long span of our trace.

Again, we validate our scheme against the synthesized NAT trace. Using the classifier we obtain a NAT inflation factor of 1.22, significantly less than the correct factor of 2. The per host IP ID sequencing scheme performs closer to ideal than using the classifier as the discriminator on our control trace. Counting NAT hosts on the basis of the unique signatures from each IP address is likely to undercount because the classifier cannot distinguish between multiple machines running the same operating system.

Nonetheless, the classifier finds a 20% host count increase due to machines masquerading behind NAT in the synthetic trace, suggesting that it is a useful metric. Combining several discriminators and techniques, for instance IP address,

operating system and IP ID sequence matching, will likely provide even higher accuracy in measuring NAT host counts.

The inferred NAT inflation factor using our classifier to distinguish unique hosts belonging to a single IP address is 1.020. This result, combined with the 9% inflation factor found using ID sequence matching, provides a measurement-based lower bound to understanding NAT prevalence in the Internet.

6 Future Work

Our classifier currently implements simplistic learning based only on the first packet in the TCP handshake. A more interesting and challenging problem is passively identifying the TCP stack variant, e.g. tahoe, and properties of the connection. Methods similar to our classifier could be used to train and evaluate packet streams for more complex inferences.

While we have confidence in our results, we require a large known data set to evaluate and improve the accuracy of our classifier. Once we collect a sufficiently large set of responses from our public web page that gathers packets and user responses, we plan to measure error in our classifier.

Finally, passively counting the number of machines using NAT enabled devices to masquerade behind a single address remains an open problem. One approach we are currently pursuing as an additional data point is measuring the Gnutella peer-to-peer network. The packet header for Gnutella query hits contains a “push” bit so machines behind NAT devices can indicate to other peers that they are not directly reachable. Measuring the prevalence of push queries provides another method of approximating NAT penetration among one user group.

7 Conclusions

We presented a statistical learning technique for passive TCP/IP fingerprinting without deep-packet inspection. Whereas rule-based approaches fail to identify as many as 5% of the hosts in our traces, our classifier provides a continuous degree of identification confidence in the face of incomplete data. We evaluated several approaches to training the classifier, including an automated method with web logs. Analysis of packet traces from an Internet exchange revealed strong dominance of commercial operating systems among all hosts. Surprisingly, we found that the freeware Linux operating system contributes a significant fraction of the byte and packet traffic.

We experimented with a new technique for determining the number of hosts masquerading behind NAT devices by leveraging our classifier. While our technique found many of the additional hosts due to NAT in a synthetic control data set, existing techniques such as Bellovin’ IP ID sequence matching performed better. Among the hosts in our exchange point trace, we found a NAT inflation factor of approximately 9% by Bellovin’s method and 2% using our classifier. Our results provide a measurement-based lower bound to understanding NAT prevalence in the Internet.

Acknowledgments. We would like to thank Mike Afergan and Kevin Thompson for reviewing initial versions of the paper. Thanks also to the Advanced Network Architecture group and Karen Sollins for her support of this work.

References

1. Zalewski, M.: Passive OS fingerprinting tool (2003)
<http://lcamtuf.coredump.cx/p0f.shtml>.
2. Smart, M., Malan, G.R., Jahanian, F.: Defeating TCP/IP stack fingerprinting. In: Proc. of the 9th USENIX Security Symposium. (2000)
3. Taleck, G.: Ambiguity resolution via passive OS fingerprinting. In: Proc. 6th International Symposium Recent Advances in Intrusion Detection. (2003)
4. Egevang, K., Francis, P.: The IP network address translator (NAT). RFC 1631, Internet Engineering Task Force (1994)
5. Bellovin, S.: A technique for counting NATted hosts. In: Proc. Second Internet Measurement Workshop. (2002)
6. Hain, T.: Architectural implications of NAT. RFC 2993, Internet Engineering Task Force (2000)
7. Senie, D.: Network address translator (NAT)-friendly application design guidelines. RFC 3235, Internet Engineering Task Force (2002)
8. Holdrege, M., Srisuresh, P.: Protocol complications with the IP network address translator. RFC 3027, Internet Engineering Task Force (2001)
9. Fyodor: Remote OS detection via TCP/IP stack fingerprinting (1998)
<http://www.insecure.org/nmap>.
10. Armitage, G.J.: Inferring the extent of network address port translation at public/private internet boundaries. Technical Report 020712A, CAIA (2002)
11. Paxson, V.: Automated packet trace analysis of TCP implementations. In: SIGCOMM. (1997) 167–179
12. Braden, R.: Requirements for internet hosts – communication layers. RFC 1122, Internet Engineering Task Force (1989)
13. Langley, P., Iba, W., Thompson, K.: An analysis of bayesian classifiers. In: National Conference on Artificial Intelligence. (1992) 223–228
14. Netcraft: Web server survey (2004) <http://www.netcraft.com>.
15. Phaal, P.: Detecting NAT devices using sflow (2003)
<http://www.sflow.org/detectNAT>.
16. Droms, R.: Dynamic host configuration protocol. RFC 2131, Internet Engineering Task Force (1997)

NETI@home: A Distributed Approach to Collecting End-to-End Network Performance Measurements

Charles Robert Simpson, Jr. and George F. Riley

Georgia Institute of Technology (Georgia Tech), Atlanta Georgia, USA,
`{rsimpson, riley}@ece.gatech.edu`

Abstract. NETI@home is an open-source software package that collects network performance statistics from end-systems. It has been written for and tested on the Windows, Solaris, and Linux operating systems, with testing for other operating systems to be completed soon. NETI@home is designed to run on end-user machines and collects various statistics about Internet performance. These statistics are then sent to a server at the Georgia Institute of Technology, where they are collected and made publicly available. This tool gives researchers much needed data on the end-to-end performance of the Internet, as measured by end-users. Our basic approach is to sniff packets sent from and received by the host and infer performance metrics based on these observed packets. NETI@home users are able to select a privacy level that determines what types of data are gathered, and what is not reported. NETI@home is designed to be an unobtrusive software system that runs quietly in the background with little or no intervention by the user, and using few resources.

1 Introduction

The need for the passive collection of end-to-end Internet performance measurements has been noted by many [1][2]. Active measurements of Internet performance have their obvious drawbacks and some believe that their results are not accurate due to the amount of traffic they inject [3] [4]. Other attempts at the passive collection of Internet performance statistics have either focused on a few collection points [1] or have the need to deploy costly and restrictive collection nodes [1][5]. Some Internet service providers (ISPs) collect end-to-end performance measurements as well; however they are usually reluctant to share this data. Further, the collection methods used by each ISP can differ dramatically and are usually not as detailed as other methods [1]. NETI@home passively gathers these measurements and makes them available to researchers while at the same time respecting the rights and privacy of the end-users. This article serves two purposes, the first is to alert the research community to NETI@home, and the second is to help publicize NETI@home so that even more users will install it on their systems and thus, more data may be collected.

2 Software Description

NETI@home is written in the C++ programming language and is built on top of the open-source Ethereal [6] software package. Ethereal is a well-written network protocol analyzer (sniffer) that is becoming the de facto network sniffer standard. NETI@home uses Ethereal's ability to sniff packets in real-time with little impact on the user's system. Should we find Ethereal to be unsuitable, we can easily use another network protocol analyzer, such as tcpdump [7], or write our own using the pcap library [8]. The sniffed packets are then sent to NETI@home for analysis, where they are sorted into bidirectional flows, as defined in [9]. These packets are *not* collected with the network interface in promiscuous mode, therefore only packets sent from and received by the user's system are analyzed.

NETI@home users are able to specify a privacy level that determines which statistics are collected and reported. At the highest privacy setting, no IP addresses are recorded. At the lowest privacy setting, all IP addresses are recorded. While the absence of IP addresses may reduce potential research possibilities, we believe that the rights and privacy of the NETI@home users are of the utmost importance.

Once NETI@home analyzes a specified number of packets, the data is compressed and sent to our server at the Georgia Institute of Technology (Georgia Tech) via TCP. This server is carefully monitored and will be upgraded if necessary. A duplicate copy of the binary data sent is stored on the user's system in a log file. The user can observe what data is actually transmitted using the included NETILogParse program. The collected packets at Georgia Tech are placed into a publicly accessible database. Researchers are then able to sort the data via a web interface to look for specific data and/or trends. This provides the much-needed end-to-end passive measurements that researchers have been wanting.

One of the most important goals of the NETI@home project is to have a large installed user base. To encourage end users to run the NETI@home software, a program written in Java, when run, maps each remote IP address contacted to a graphical display of the world. This plot allows users to visualize where each remote host with which they communicate is located geographically. To resolve IP addresses into latitude/longitude coordinates, the NetGeo [10] database from CAIDA is used.

3 Network Statistics

NETI@home collects many different statistics from end-users, in accordance with CAIDA specifications [9], with more statistics to be added in later releases. These statistics focus on four transport-layer protocols on the Internet: the Transmission Control Protocol (TCP), the User Datagram Protocol (UDP), the Internet Control Message Protocol (ICMP), and the Internet Group Management Protocol (IGMP), as well as their underlying network-layer protocols (such as the Internet Protocol, IP). The following is a brief list of some of the statistics gathered:

3.1 Transmission Control Protocol

- Source and Destination IP Addresses

Note: The IP addresses are only recorded if the user wishes, in accordance with their selected privacy setting.

- Source and Destination Ports
- Times the Connection was Established and Closed
- Number of Bad Checksums (IP and TCP)¹
- Number of Packets Sent and Received
- Number of Bytes Sent and Received
- Number of Acknowledgment Packets
- Number of Duplicate Acknowledgment Packets
- Number of Triple Duplicate Acknowledgment Packets
- Number of URG Flags
- Number of PUSH Flags
- Number of ECNECHO Flags
- Number of CWR Flags
- Sender and Receiver SACK Permitted
- Sender and Receiver Minimum Window Size
- Sender and Receiver Average Window Size
- Sender and Receiver Maximum Window Size
- Sender and Receiver Minimum TTL Values
- Sender and Receiver Maximum TTL Values
- Number of Packet *Retransmissions*

A *retransmission* is defined as a packet with a sequence number less than the highest sequence number that has been observed thus far.

- Number of Bytes Retransmitted
- Number of *Inactivity Periods*

An *inactivity period* is defined to be a period in which no packets are sent or received for at least 200 milliseconds. This statistic is a rough estimate of a TCP timeout.

- Minimum, Maximum, and Average *Round Trip Times*

Round trip time estimation is made by keeping a list of all packets for a flow. When an acknowledgment arrives, if its acknowledgment number is equal to one of the packets' sequence numbers plus that packet's length and that packet has not been retransmitted, the round trip time is estimated to be the difference between the time the acknowledgment arrives and when the original packet was sent. However, if the original packet had its SYN flag set, then the acknowledgment number should equal its sequence number plus one.

¹ This statistic does not work on systems that perform checksum offloading, such as Windows 2000

- Number of Packets Received *In-Order*

A packet is defined to be *in-order* if its sequence number is equal to the sum of the sequence number and length of the packet received immediately before it.

- Number of Packets Received *Out-of-Order*

A packet is defined to be *out-of-order* if it is not in-order.

- Number of Fragmented Packets
- Number of Don't Fragment Flags
- Sender and Receiver Maximum Segment Size (if specified)
- Number of *Rejected Connection Requests*

A connection is deemed *rejected* if the TCP three-way handshake procedure fails.

- Was the Flow *Idle Closed*?

A connection is defined to be *idle closed* if there has been no activity in the flow for 64 seconds, as per [9].

- Was the Flow *Reset Closed*, and by Whom?

A connection is defined to be *reset closed* if a packet is sent or received with the RST flag set.

- Was the Flow *FIN Closed*, and by Whom?

A connection is defined to be *FIN closed* if a packet is sent or received with the FIN flag set.

3.2 User Datagram Protocol

- Source and Destination IP Addresses

Note: The IP addresses are only recorded if the user wishes, in accordance with their selected privacy setting.

- Source and Destination Ports
- Number of Bad Checksums (IP and UDP)¹
- Number of Packets Sent and Received
- Average Packet Sizes
- Minimum Packet Sizes
- Maximum Packet Sizes
- Number of Fragmented Packets
- Number of Don't Fragment Flags
- Time of the First Packet
- Time of the Last Packet

3.3 Internet Control Message Protocol

- Source and Destination IP Addresses

Note: The IP addresses are only recorded if the user wishes, in accordance with their selected privacy setting.

- Number of Bad Checksums (IP and ICMP)¹
- Number of Fragmented Packets
- Number of Don't Fragment Flags
- ICMP Type
- ICMP Code

3.4 Internet Group Management Protocol

- Source and Destination IP Addresses
- Multicast IP Address

Note: The IP addresses are only recorded if the user wishes, in accordance with their selected privacy setting.

- Number of Bad Checksums (IP and IGMP)¹
- IGMP Version
- IGMP Type
- Number of Packets Sent and Received
- Number of Fragmented Packets
- Number of Don't Fragment Flags
- Maximum Response Time
- Time of the First Packet
- Time of the Last Packet

3.5 Miscellaneous

- NETI@home Version
- Operating System

The user's operating system is determined by either the GNU autoconf tool [11] or the appropriate functions in Microsoft Windows.

- *Send Time*

The *Send Time* is the time the data was sent to Georgia Tech, as recorded by the user's system.

- *Arrival Time*

The *Arrival Time* is the time the data arrived at Georgia Tech. By observing both the send and arrival times, major timing discrepancies between Georgia Tech and the user's system can be accommodated.

4 Distributing NETI@home

It is our goal to have NETI@home installed in many academic computer labs across the country as well as on many end-user systems in homes and offices. As the number of users increases, so does the value of our data. NETI@home is currently available for the Linux and Solaris operating systems, in RPM and tarball format, and for the Windows operating system in the form of a self-installing executable. All distributions are available from our website located at <http://neti.sourceforge.net/>. In addition to these distributions, we are in the process of testing distributions for other platforms such as the Macintosh operating system. To use NETI@home, one must also install the Ethereal network analyzer [6], which is available from the Ethereal website, <http://www.ethereal.com/>.

Since NETI@home is copyrighted (copylefted) under the GNU General Public License (GPL) [12], the source code is also available from our public website, <http://neti.sourceforge.net/>. Being an open-source project, NETI@home users do not have to worry about the possibility of so-called “spyware,” as they are free to see exactly how NETI@home works, and make suggestions if they wish.

5 Future Work

In the future, we will add many more features to NETI@home to aid researchers. For example, we would like to be able to collect the size of the TCP congestion window, which would allow us to monitor the TCP congestion control algorithm. However, to collect the size of the congestion window will require a lower-level application, such as a kernel module, or some other novel technique. Some other new features we are considering are: collecting the bandwidth available to the end user, adding additional transport-layer protocols to the current four that are studied, and collecting traceroutes from the user to selected destinations. Depending on the user’s privacy level, the traceroute will either not be recorded, will be trimmed, or it will be fully recorded. The traceroute was not initially included because it might tax users’ systems as well as the network itself. However, using the mapping program provided with NETI@home, traceroute functionality will be provided. When a user clicks a specific host on the map, another window will appear and a graphical traceroute will be performed. Since the traceroute is explicitly performed by the user, it should not be considering taxing to the user’s system or the network, and it truly represents a passive technique. This data will also be sent to the server at Georgia Tech to add to the amount of measurements that are collected. Finally, the NETI@home online database will receive many enhancements and analysis will be performed on the collected data.

6 Conclusions

NETI@home is a software package designed to run on end-user systems to collect various network performance statistics. NETI@home represents an excellent opportunity to further research into the performance of the Internet. We encourage

anyone and everyone to install the software on their machines to increase the amount of data with which researchers can work. NETI@home is an unobtrusive software package that runs quietly in the background and respects the privacy of the end-user. We further encourage researchers to use the data made publicly available by NETI@home to continue the advancement of the global Internet.

References

1. Murray, M., kc claffy: Measuring the immeasurable: Global internet measurement infrastructure. In: Proceedings of PAM2001 - A workshop on Passive and Active Measurements. (2001)
2. Hou, Y.T., Dong, Y., Zhang, Z.L.: Network performance measurement and analysis part 1: A server-based measurement infrastructure (1998) Fujitsu Laboratories of America, Technical Report FLA-NCRTM98-01.
3. Barford, P., Sommers, J.: A comparison of active and passive methods for measuring packet loss (2002) University of Wisconsin Technical Report.
4. Mochalski, K., Irmscher, K.: On the use of passive network measurements for modeling the internet. In: KiVS. (2003)
5. Fraleigh, C., Diot, C., Lyles, B., Moon, S., Owezarski, P., Papagiannaki, D., Tobagi, P.: Design and deployment of a passive monitoring infrastructure. Lecture Notes in Computer Science **2170** (2001)
6. Combs, G., et al.: Ethereal: - a network protocol analyzer. Software on-line: <http://www.ethereal.com> (2004)
7. Jacobson, V., Leres, C., McCanne, S.: tcpdump. Software on-line: <http://www.tcpdump.org> (1989) Lawrence Berkeley Laboratory.
8. Jacobson, V., Leres, C., McCane, S.: libpcap. Software on-line: <http://www.tcpdump.org> (1989) Lawrence Berkeley Laboratory.
9. CAIDA: Caida: Preliminary measurement specification for internet routers. <http://www.caida.org/tools/measurement/measurementspec/> (2004) The Cooperative Association for Internet Data Analysis - CAIDA.
10. Moore, D., Periakaruppan, R., Donohoe, J., kc claffy: Where in the world is net-geo.caida.org? In: INET 2000. (2000)
11. GNU: Gnu autoconf. Software on-line: <http://www.gnu.org/software/autoconf/> (1998)
12. GNU: Gnu general public license. <http://www.gnu.org/licenses/> (1991)

Comparative Analysis of Active Bandwidth Estimation Tools

Federico Montesino-Pouzols

Institute de Microelectrónica de Sevilla (IMSE-CNM) – CSIC
Avda. Reina Mercedes s/n. Edif. CIGA. E-41012 Seville, Spain
Federico.Montesino@imse.cnm.es

Abstract. A comparative analysis of state-of-the-art active probing tools for bandwidth estimation is outlined. Techniques and tools for capacity, available bandwidth and bulk transfer capacity estimation are simultaneously assessed. First, a generic framework for the design of active bandwidth estimation tools is proposed as a result of our analysis of the implementation and performance of a number of available techniques and tools. Then, we describe a first and comprehensive series of tests performed over a broad range of bandwidth estimation tools, scenarios and experimental conditions. These tests have been done with the aim to develop a set of practices, procedures and tools for the comparative analysis of active bandwidth estimation techniques and tools, which are presented. Finally, Overall conclusions are discussed and pointers to publicly available records and tools are given.

1 Introduction

Active bandwidth estimation tools can not only provide network operators with useful information on network characteristics and performance, but also can enable end users (and user applications) to perform independent network auditing, load balancing, and server selection tasks, among many others, without requiring access to network elements or administrative resources.

The research community is developing a set of metrics and techniques for active bandwidth measurement. Many of them [1] are well understood and can provide accurate estimates under certain conditions.

Some institutions have announced initiatives to deploy test platforms for active and passive bandwidth estimation as well as other related techniques, though no substantial results have been reported as for active bandwidth estimation. Also, some partial measurement and evaluation studies of bandwidth estimation tools have been published [2,3]. Nonetheless, we draw attention to the lack of publicly available comprehensive and comparative experimental results. We also note the lack of common and consistent practices and procedures to test available active bandwidth measurement tools.

Our aim is to develop a set of practices, procedures and tools for the comparative analysis of active bandwidth estimation techniques and tools. This way, we expect to fulfill two goals:

- Ease the deployment of platforms that take advantage of the huge amount of available experimental resources to provide a solid experimental basis for research on active bandwidth estimation.
- Assess to what extend current bandwidth estimation tools can be used as basis for providing a bandwidth estimation service for user applications as well as networks operation.

2 Analysis of Bandwidth Estimation Techniques

Our study covers measurement techniques as well as implementations. In this section, an study of measurement techniques for different metrics and techniques is outlined, ignoring implementation details of tools. The nomenclature and taxonomy of metrics, measurement techniques and tools given in [1] is considered as reference.

Our approach is to simultaneously assess techniques and tools for capacity, available bandwidth and bulk transfer capacity estimation, which allows us to abstract a common design model for these tools.

In this section, we present a generic framework for the design of active bandwidth estimation tools that will be referenced in following sections. Though this analysis can be generalized to network performance and dynamics analysis techniques, we restrict the following discussion to bandwidth estimation techniques.

A review of the source code of a number of active bandwidth estimation tools has been performed. As in early stages of development of bandwidth estimation tools there has been no clear taxonomy of techniques and no consensus on nomenclature and definition of bandwidth metrics, a great deal of code duplication among bandwidth estimation tools has been found.

The facts that `pathChirp` [4] has been implemented using the `NetDyn` [5] code as starting point, and that `cprobe` [6] and `pipechar` [7] provide estimates of the asymptotic dispersion rate [8,3] rather than the available bandwidth, are revealing. Thus, we have paid special attention to isolating reusable components in bandwidth estimation tools.

A simplified scheme that summarizes components and relations of a bandwidth estimation tool according to our analysis is depicted in Figure 1. Two main stages in these tools are identified: measurement and estimation. *Measurement* involves the generation of a probe packet pattern, its transmission through the network, and its reception and measurement. *Estimation* comprises statistical and heuristic processing of measurements according to some network model.

A major component is what we call probe packets generation system; both the measurement and the estimation stages depend on it. Isolation of this function as a component allows for the efficient implementation of tools for a set of measurement techniques, since facilities for common traffic patterns generation can be reused.

The probe packet generation component should provide generic services such as generation of packet trains, which currently can be found implemented as functions or methods in most tools based on packet trains. In case of packet

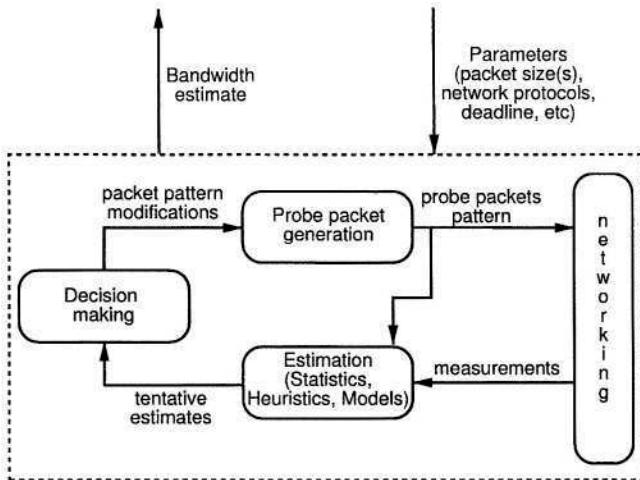


Fig. 1. Design scheme of an active bandwidth estimation tool

trains for techniques that follow the probe gap model [3], the packet gap can be fixed (as in IGI/PTR [2]), or a Poisson process (as in Spruce [3]. Similarly, in case of techniques that use the probe rate model [3], facilities can be defined for different patterns of packet trains as found in tools such as Pathload [9], PathChrip [4] and TOPP [10].

Also, within this design framework, traffic generation tools (such as Iperf [11]), often employed jointly with bandwidth estimation tools in order to perform tests, can be integrated as additional facilities provided by the probe packets generation component.

A feedback line from the estimation component to the probe packets generation component can improve performance by reducing network load and required estimation time. Within this feedback line, we consider a third component, *decision making*, in charge of conducting the probe packet generation process depending on the estimated accuracy and convergence of tentative estimates from the estimation component.

Keeping the probe packets generation system as a generic component separate from the decision making component makes it possible to implement tools that combine different probe packets patterns to estimate bandwidth, in a way similar to that of IGI/PTR. Some tools, such as early versions of pchar [12] lack the decision making module. In such cases, the probe packets pattern is fixed for each set of input parameters.

Note that the simplified scheme shown omits some reusable subsystems such as route change detection. In case of tools that run both at the receiver and the sender side, the scheme in Figure 1 corresponds to the end system that sends probe packets (usually the client). A more in depth description of this design can be found at [13]

3 Comparative Analysis of Active Bandwidth Estimation Tools

In this section, we outline the comparative analysis of active bandwidth estimation tools carried out. We describe the scenarios considered as well as overall practices and procedures followed to conduct experiments. General conclusions from our experiments are summarized.

We evaluate tools rather than techniques, focusing on design and implementation details that affect estimates accuracy, efficiency, speed and reliability. We put the focus on analyzing experimental results for developing and refining practices, procedures and tools for iterative tests performance.

Previous initiatives conducted by a number of institutions have been taken into account. In particular, we have partially based our work on studies performed on the old TEN-155 European research and education network [14] (which reports a detailed study, based on netperf [15], performed over a number of network paths and throughout long periods of time), a more recent analysis performed for GÉANT [16] (the current European research and education network), and the Internet Monitoring Program at Stanford Linear Accelerator Center [17].

These initiatives are aimed at studying the deployment of bandwidth estimation tools in production networks as the basis for an additional service available for end-user applications as well as network operators. However, no systematic evaluation of a significant subset of currently available active bandwidth estimation tools with respect to a complete set of criteria, across a wide variety of network paths and experimental conditions, using a well defined set of procedures, has been conducted.

Since we focus on experimental results, we do not perform tests through simulation but only using emulated and real scenarios.

Tools such as dummynet and NIST Net running on FreeBSD and GNU/Linux boxes are being employed for scenario emulation. Emulated scenarios make it possible for us to test the effects both of extreme network configurations and traffic shapers on estimation tools.

Real scenarios range from LAN environments and cable subscriber networks operated by private ISPs to the high performance Spanish NREN (national research and education network) backbone operated by RedIRIS, with connections to GÉANT as well as private ISPs. Tests are being conducted on network paths with variable length, ranging from single links to paths that traverse one or more administrative domains.

Most tests have been performed through paths between the following three end points:

- Cable modem end point (private ISP subscriber line).
- Several hosts at a LAN environment within a research institution connected to the Spanish NREN.
- Host near the RedIRIS backbone.

Considering the paths between these end points, we have applied a set of general practices. The most important ones are the following:

- For each path, whenever possible, tests are performed in both directions, distinguishing symmetric and asymmetric paths.
- Estimation tools are run through GNU time, recording its output together with the tool logs.
- Attach to test logs output from helper tools that provide basic information about network paths, such as DNS lookups, traceroute and ping.
- When testing each of the analyzed tools, perform tests with only one instance of the tool, and also with multiple instances of the same tool running at the same time.
- Perform tests both with overall idle systems and heavily loaded systems. For the latter case, we distinguish three types of system load:
 - cpu intensive tasks
 - input/output intensive tasks
 - both cpu and input/output intensive tasks
- For tools based on ICMP, perform tests with several levels of ICMP rate limits.
- In order to analyze the effects of simple traffic patterns, we use automated traffic generation tools such as Iperf (see <http://www.icir.org/models/trafficgenerators.html>).
- Consider path properties that imply known limitations for which there is no known solution, among which the effect of layer-2 store-and-forward devices on capacity estimation tools based on the variable packet size technique has been taken into account for our LAN experiments.

For simplicity, we are using just IPv4, as it is the network protocol common to the whole set of tools under test.attention has being paid to failure conditions and to the set of parameters accepted as user input by the tools being tested. We have performed small modifications to the source code as a way to apply some of our procedures.

Some tests have been performed using tools for estimating capacity , available bandwidth and bulk transfer capacity under the same network configuration and conditions. In such cases, for each path and each hop in the path, the theoretical relationships between these metrics provide a method to detect wrong estimates. These tests also reveal similar and disparate effects of practical conditions and scenarios on different estimation techniques.

We have completed a source code review for a comprehensive number of active bandwidth estimation tools available from the research community (see the lists of tools provided by Cooperative Association for Internet Data Analysis (CAIDA) and Stanford Linear Accelerator Center [17],among other organizations.

Tools and versions analyzed to date are enumerated in Table 1. A taxonomy as well as a brief discussion of many of them can be found in [1]. Previous

Table 1. Tools and versions analyzed

Tool	Version
Capacity	
bprobe	1.0
clink [18]	1.0
Nettimer [19]	2.3.8
pathrate	2.3.3
pchar	1.4 (+ Debian GNU/Linux package 1.4-4 patches)
sprobe [20]	0.3
Available Bandwidth	
cprobe	1.0
IGI/PTR	1.0
NetDyn	-
pathChrip	2.3.3
pathload	1.1.1
pipechar	Mar25-2K1
Spruce	0.2
Bulk Transfer Capacity	
Iperf [11]	1.7.0
netperf	2.2pl4
Treno	961001
nttcp	1.43

comparisons of some of the available bandwidth estimation tools are also published [2,3]. For a complete and updated enumeration of the tools analyzed refer to [13].

As we perform tests over these tools, comparative analysis procedures are being developed. Among these, we highlight a number of generic comparison criteria, though we note that not every criterion can be applied to each and every tool.

We have classified these criteria into two groups: a first set of criteria specified as simple numerical and scalar metrics that provide a partial comparison of tools performance, and a second set of criteria that provide a comparative evaluation of tools performance.

The first set includes metrics that can be useful to quickly choose or discard some techniques and tools according to simple constraints such as required time or allowed probe bandwidth, some of these metrics are the following:

- Total probe traffic generated.
- Maximum attainable accuracy.
- Total estimation time.

The second set of criteria, suited for comparative analysis between different techniques includes but is not limited to the following:

- Accuracy: maximum accuracy for a fixed time, maximum accuracy for a limited amount of probe traffic, dependency of accuracy on estimation time, and dependency of accuracy on probe traffic volume.
- Consistency of estimates (as discussed in [18]).
- Efficiency: required network load for a given accuracy, and required network load for a given estimation time.
- Estimation time: required time for a given accuracy.
- Dependency of accuracy, efficiency and estimation time on overall network load as well as overall machine (both sender and receiver) load.
- Dependency of accuracy, efficiency and estimation time on path properties such as number of hops and round trip time.

In addition to these numerical criteria for comparative analysis, we have analyzed the response of estimation tools to the following factors:

- Effects of path asymmetry.
- Effects of layer-2 store-and-forward devices.
- Route changes.
- Installation required on both ends of the path.
- Possibility to estimate per hop properties.
- What path property is actually measured for paths with traffic shaping nodes.
- Reliability in case of loss of connectivity between sender and receiver.
- Possibility to run multiple instances simultaneously. Particularly, as already noted in [3], train generation tools hold the CPU for the full packet train generation interval, preventing the simultaneous generation of several trains. In low bandwidth links, the delay is high enough so as to use system calls for scheduling. However, high bandwidth links require the design of packet train generators able to cope with multiple packet trains.
- Congestion avoidance mechanisms implemented in tools.
- Effects of cross traffic.
- Upper and lower bandwidth limits for which the tool works.
- Operating system privileges required to run the tool (privileged socket numbers, raw sockets interface, etc.).
- Underlying suppositions in the network model and estimation technique followed by the tool. path is a static characteristic. As an example, tools based on the probe gap model [3] (as Spruce, and IGI) assume there is a single bottleneck in the path (which is both the narrow and tight link), and also assume that the queue of that link does not become empty between the probe packets [2,3]
- Support for important generic parameters, such as socket numbers and constraints for probe packets size.

Repeated application of the aforementioned practices and procedures has led to the development of a system of tests automation as well as results processing and recording scripts implemented in Ruby. These have been complemented with

additional scripts for visualization based on the comparison criteria described, making up a framework for the comparative analysis of active bandwidth estimation tools.

As for the comparison of output from different tools, we point out the need for a common output format for active bandwidth estimation tools. Output from scripts of the test framework is kept in NetLogger ULM format [21]. Also, we are currently developing scripts that convert, when possible, specific log formats to the NetLogger ULM format (already supported by `pathload` and `pathrate`) for reporting estimates.

Due to space constraints we do not include a complete description of experiments. A more comprehensive description can be found at [13].

We note, however, that current tools are implementations of one bandwidth estimation technique. For this reason, differences in networking code (particularly those portions related to timestamping) may have an undesired impact on comparing similar techniques implemented through different tools. These and other implementation issues are hard to quantify and even isolate.

It has been found that sometimes the analyzed tools provide very wrong estimates or even no estimates at all likely as a consequence of implementation issues, specially noticeable when estimating high bandwidths (around 100 Mbps and above), but also relatively often when estimating bandwidth of 10Mbps or even lower.

Limited clock resolution and system I/O throughput, as well as the fact that these tools are usually run on non real-time systems seem to be the main underlying reasons. Recent developments [22] and analysis [23] confirm our observations.

4 Conclusions and Future Work

We have presented the first systematic comparative analysis of active bandwidth estimation tools performed over a broad range of tools, scenarios (both emulated and real) as well as experimental conditions, which results are publicly available from [13]. Techniques and tools to estimate capacity, available bandwidth and bulk transfer capacity have been studied jointly.

With this work we aim to lay out a common framework for the design and implementation of component based active bandwidth estimation tools. In addition, a set of practices, procedures and tools which make up a test framework for the comparative analysis of active bandwidth estimation tools has been developed. The test framework as well as records from our experiments are also available from [13].

As a result from the tests carried out, a number of failure conditions for the analyzed tools have been identified, as well as the dependency of the estimates accuracy on factors such as system load and network path properties. How these factors impact estimates of the three metrics studied (capacity, available bandwidth and bulk transfer capacity) has been jointly analyzed.

As further development we plan to extend the set of considered scenarios to include network testbeds such as PlanetLab [24]. Also, we are currently working on a web interface for the presented test framework which is intended to be a first experimental platform for cooperative comparative analysis of bandwidth estimation tools through the Internet, based on the practices, procedures and tools developed.

Finally, we expect to raise a well defined set of questions (and rationales from our experiments) in order to establish a technical comparison criteria that could eventually lead to the development and standarization of benchmarks for active bandwidth estimation tools.

Acknowledgement. We would like to thank D.R. López and A. Barriga for their useful comments and assistance. We would also like to thank IMSE and RedIRIS for allowing us to use their resources in our experiments

References

1. Prasad, R.S., Murray, M., Dovrolis, C., Claffy, K.: Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network Magazine* (2003) <http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/publications.html>.
2. Hu, N., Steenkiste, P.: Evaluation and Characterization of Available Bandwidth Probing Techniques. *IEEE Journal on Selected Areas in Communication* **21** (2003) 879–894 <http://gs274.sp.cs.cmu.edu/www/igi/>.
3. Strauss, J., Katabi, D., Kaashoek, F.: A Measurement Study of Available Bandwidth Estimation Tools. In: *ACM SIGCOMM Internet Measurement Workshop*. (2003) <http://www.icir.org/vern/imc-2003/program.html>.
4. Ribeiro, V.J., Riedi, R.H., Baraniuk, R.G., Navratil, J., Cottrell, L.: pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In: *Passive and Active Measurement Workshop*. (2003) <http://www.pam2003.org/proceedings.html>.
5. Banerjee, S., Agrawala, A.K.: Estimating Available Capacity of a Network Connection. In: *IEEE International Conference on Networks*, Singapore (2000) <http://www.cs.umd.edu/~suman/>.
6. Carter, R.L., Crovella, M.E.: Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks. Technical Report TR-96-007, Boston University. Computer Science Department (1996) <http://cs-www.bu.edu/techreports/pdf/1996-007-dss-using-bandwidth.pdf>.
7. Jin, G., Yang, G., Crowley, B., Agarwal, D.: Network Characterization Service (NCS). In: *10th IEEE Symposium on High Performance Distributed Computing*. (2001) <http://dsd.lbl.gov/DIDC/papers/NCS.HPDC01.pdf>.
8. Dovrolis, C., Ramanathan, P., Moore, D.: What Do Packet Dispersion Techniques Measure? In: *INFOCOM*. (2001) 905–914 <http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/publications.html>.
9. Jain, M., Dovrolis, C.: End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In: *ACM SIGCOMM*, Pittsburgh, PA (2002) <http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/publications.html>.

10. Melander, B., Bjorkman, M., unningberg, P.: A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks. In: IEEE Globecom – Global Internet Symposium, San Francisco (2000)
11. Tirumala, A., Qin, F., Dugan, J., Ferguson, J., Gibbs, K.: Iperf Version 1.7.0. NLANR Distributed Applications Support Team (2003)
<http://http://dast.nlanr.net/Projects/Iperf/>.
12. Mah, B.A.: pchar: A Tool for Measuring Internet Path Characteristics (2001)
<http://www.employees.org/~bmah/Software/pchar/>.
13. Federico Montesino-Pouzols et al.: Active Bandwidth Estimation Tools: ... (2004)
<http://www.imse.cnm.es/~fedemp/abet/>.
14. Blom, H., de Laat, C.: User Level Network Performance Monitoring Program. TERENA Technical Report (2000) <http://www.terena.nl>.
15. Jones, R.: Public Netperf Homepage (2004)
<http://www.netperf.org/netperf/NetperfPage.html>.
16. Leinen, S., Reijns, V.: GEANT Deliverable D9.7: Testing of Traffic Measurement Tools. Technical report, DANTE (2002)
<http://archive.dante.net/tf-ngn/activities.html>.
17. Cottrell, L., Matthews, W.: Internet Monitoring at Stanford Linear Accelerator Center (2002) <http://www.slac.stanford.edu/comp/net/wan-mon.html>.
18. Downey, A.B.: Using pathchar to estimate Internet link characteristics. In: ACM SIGCOMM, Cambridge, MA (1999) 241–250
<http://allendowney.com/research/clink/>.
19. Lai, K., Baker, M.: Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. In: USENIX Symposium on Internet Technologies and Systems. (2001)
<http://mosquitonet.stanford.edu/~laik/projects/nettimer/>.
20. Saroiu, S., Gummadi, P.K., Gribble, S.D.: SProbe: A Fast Technique for Measuring Bottleneck Bandwidth in Uncooperative Environments.
<http://sprobe.cs.washington.edu/> (2002)
21. Abela, J., and, T.D.: Universal Format for Logger Messages. Internet draft (1999)
draft-abela-ulm-05.txt.
22. Alberi, J.L., McIntosh, A., Pucci, M., Raleigh, T.: Overcoming Precision Limitations in Adaptive Bandwidth Measurements. In: 3rd New York Metro Area Networking Workshop (NYMAN 2003), New York (2003)
<http://www.nyman-workshop.org/2003/program.html>.
23. Jin, G., Tierney, B.L.: System Capability Effects on Algorithms for Network Bandwidth Measurement. In: Internet Measurement Conference, Miami Beach, FL, USA (2003) 27–38 <http://www.icir.org/vern/imc-2003/program.html>.
24. PlanetLab: An Open Platform for Developing, Deploying, and Accessing Planetary-Scale Services. (2004)
<http://www.planet-lab.org>.

Active Measurement for Multiple Link Failures Diagnosis in IP Networks

Hung X. Nguyen* and Patrick Thiran

EPFL CH-1015 Lausanne, Switzerland

Abstract. Simultaneous link failures are common in IP networks [1]. In this paper, we develop a technique for locating multiple failures in Service Provider or Enterprise IP networks using active measurement. We propose a two-phased approach that minimizes both the additional traffic due to probe messages and the measurement infrastructure costs. In the first phase, using elements from max-plus algebra theory, we show that the optimal set of probes can be determined in polynomial time, and we provide an algorithm to find this optimal set of probes. In the second phase, given the optimal set of probes, we compute the location of a minimal set of measurement points (beacons) that can generate these probes. We show that the beacon placement problem is NP-hard and propose a constant factor approximation algorithm for this problem. We then apply our algorithms to existing ISP networks using topologies inferred by the Rocketfuel tool [2]. We study in particular the difference between the number of probes and beacons that are required for multiple and single failure(s) diagnosis.

1 Introduction

Routing decisions and content distribution require proper connectivity and latency information to direct traffic in an optimal fashion. The family of Internet protocols collect and distribute only a limited amount of information on the topology, connectivity and state of the network. Hence information of interest for Internet Service Providers (ISPs), such as link delays or link failures, has to be inferred from experimental measurements. The strategy to obtain network information through end-to-end measurements, known as Internet tomography, is therefore of great interest to the research community [3,4,5,2,6,7]. The majority of work on network tomography concentrates on either topology discovery (e.g. [3,4,2]), or link delay monitoring (e.g. [5]). Some recent research showed that active measurements can also be used to pinpoint failures in IP networks [6,7].

In general, an active probing system consists of several measurement points. Each measurement point, called a *beacon*, can send IP messages to all nodes in the network. Each message sent from a beacon to a network node for the purpose of monitoring is called a *probe*. To detect failures, the path that each probe actually follows is compared with the path that the probe should follow

* Hung X. Nguyen's work is financially supported by grant DICS 1830 of the Hasler Foundation, Bern, Switzerland.

according to the current network topology information. If the two paths are different, at least one link in the path determined by the current topology has failed [7]. The authors in [6,7] study the problem of detecting and locating failures under the assumption that at most one failure can happen at a time. Under this assumption, a set of probes that traverse all links in the network is sufficient to diagnose any such single failure. The objective in [6,7] is to determine the smallest set of beacons whose probes cover all links in the network. This problem is proven to be NP-complete and approximation algorithms are given in [6] for general topologies and in [7] for the current Internet topology. Once the beacons are located, the smallest set of probes must still be determined, Bejerano et. al. [6] show that this problem is also NP-complete.

Multiple link failures in IP networks happen much more frequently than one might expect; [1] has recently reported that nearly 35% of link failures in the Sprint backbone are multiple failures. This figure emphasizes the need for a failure detection and location scheme that takes into account the existence of multiple failures. Although the active probing scheme in [6] can work under the presence of several link failures, it cannot detect and locate *simultaneous* link failures. To our knowledge, there are no active monitoring techniques to date that apply to simultaneous failures in IP networks.

In this work, we are interested in using active probing to detect and locate link failures, under the assumption that several links can fail at one time. To achieve this goal, a distributed set of beacons running a special software are deployed at key sites across the entire network. A beacon only needs to send probes to other nodes, and see what routes the probes take. Probe messages can be implemented by tools like traceroute [8] or skitter [9]. Using probes to pinpoint network faults has several advantages over monitoring routing protocol messages (e.g., OSPF LSAs), or using SNMP traps to identify failed links. Firstly, probe-based techniques are routing protocol agnostic; as a result, they can be used with a wide range of protocols like OSPF, IS-IS, RIP, etc. Secondly, SNMP trap messages may be unreliable because they use UDP as the transport protocol [6]. Note here that by using active probing, we may not be able to detect and locate all failures (single or multiple) uniquely. This is especially true when we consider multiple failures or when there are constraints on the set of nodes where beacons can be deployed. Even a probing scheme of maximal detection capacity, that is, a scheme that would use all the available beacon nodes and send probes from these beacons to all nodes in the network would only detect and locate a subset of all possible failures. Therefore, instead of looking for a probing scheme that guarantees the detection and location of every failure of a given multiplicity, we find a probing scheme that can detect and locate every failure that the probing scheme with maximal detection capacity can detect and locate.

The cost of using probes for fault diagnosis comprises two components: the additional traffic due to probe packets and the infrastructure cost for beacons. Similarly to [6], we use a two-phased approach to minimize both the number of probes and the number of beacons. Whereas, unlike [6], we first minimize the number of probes and next the number of beacons. This enables us to use results from max-plus algebra for the probe selection problem. Our main contributions are as follows. (i) We show that, contrary to the single failure case [6] and

surprisingly so, the optimal set of probes for multiple failures can be found in *polynomial time*. (ii) However, like the single failure case, we show that the beacon placement problem in the multiple failure case is NP-hard. We provide a constant factor approximation algorithm for the beacon placement problem, which is very close to the best possible bound. (iii) We show that our algorithms perform well on existing networks, and that there is a substantial difference between the number of probes and beacons that are required for multiple failures diagnosis and for single failure diagnosis.

The remainder of this paper is organized as follows. Section 2 introduces the network model. Section 3 presents the probe selection problem, and Section 4 describes the beacon placement problem. Section 5 contains experiment studies of our algorithms on existing ISP networks. Finally, we conclude the paper in Section 6.

2 Network Model

We model the network as an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where the graph nodes, \mathcal{V} , denote the network elements and the edges, \mathcal{E} , represent the communication links connecting them. The number of nodes and edges is denoted by $n = |\mathcal{V}|$ and $e = |\mathcal{E}|$, respectively. Further, we use $P_{s,t}$ to denote the path traversed by an IP packet from a source node s to a destination node t . If there is no failure in the network, an IP packet that is sent from a node s to a destination t will follow the path $P_{s,t}$. When there is/are failure(s) of links on the path $P_{s,t}$, the probe has to be rerouted around the failed link(s); Therefore the actual path that the probe takes will be different from $P_{s,t}$. By comparing the actual path that the probe from s to t takes and $P_{s,t}$, we can detect if any link in the path $P_{s,t}$ has failed or not. When a probe detects link failure(s) in its path, we say that the probe has *failed*.

For a known topology $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a set of probes \mathcal{P} , we can compute which probes will fail when a network link goes down. We call the above relations between network links and probes *dependency relations*. Dependency relations of a network can be represented by a *dependency matrix* D of dimension $e \times n_p$, where e is the number of links and $n_p = |\mathcal{P}|$ is the number of probes in the network. D is constructed as follows. Let $P_{s,t}$ be the path followed by probe p_i in the normal situation without failures. Then the entry $D(i, j) = 1$ if the path $P_{s,t}$ contains the link e_j and $D(i, j) = 0$ otherwise. A row of D therefore corresponds to a probe (more precisely, to the path that the probe take), whereas a column corresponds to a link.

3 Probe Selection Problem

Given a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of beacons \mathcal{V}_B , we denote by \mathcal{P}_{max} the set of probes generated when each beacon sends probes to all nodes in the network. \mathcal{P}_{max} represents an upper bound on the probing capability of the network. It is the largest set of probes that can be sent to different destinations in the network. Let D_{max} denote the dependency matrix when the set of probes is \mathcal{P}_{max} .

Let $\mathcal{S} \subseteq \mathcal{E}$ be a set of links. The set of failed probes triggered by the failure of all links in \mathcal{S} is made of all the failed probes that would be triggered by the individual failure of each link in \mathcal{S} . The failure of links in \mathcal{S} can thus be represented by a vector $\vec{d}^{\mathcal{S}}$ that is obtained by OR-ing all the column vectors of D_{max} representing individual links in \mathcal{S} . Let us define the failure of a set of network links as the failure of all links in that set. A probe p_k is said to distinguish the failures of two subsets $\mathcal{E}_1, \mathcal{E}_2 \subseteq \mathcal{E}$ if and only if the corresponding k th entries in $\vec{d}^{\mathcal{E}_1}$ and $\vec{d}^{\mathcal{E}_2}$ are different, i.e., $d^{\mathcal{E}_1}(k) \neq d^{\mathcal{E}_2}(k)$. The probe set \mathcal{P} is said to distinguish the failures of two subsets $\mathcal{E}_1, \mathcal{E}_2 \subseteq \mathcal{E}$ if and only if there exists a probe $p \in \mathcal{P}$ such that p distinguishes $\mathcal{E}_1, \mathcal{E}_2$. We are interested in the following optimization problem.

Definition 1. [PS problem] *The probe selection problem is the determination of the smallest subset \mathcal{P}^* of \mathcal{P}_{max} , such that any two subsets of \mathcal{E} whose failures are distinguished by \mathcal{P}_{max} are also distinguished by \mathcal{P}^* .*

Let D^* be the dependency matrix for the system with the set of probes \mathcal{P}^* . In terms of dependency matrices, the probe selection problem amounts to removing some rows of D_{max} in order to obtain a new matrix D^* that verifies the following properties: (i) D_{max} and D^* have the same number of columns, and (ii) whenever two vectors obtained by OR-ing up to $e = |\mathcal{E}|$ columns of D_{max} are different, the two vectors obtained by OR-ing the same columns in D^* are also different.

The set of all binary vectors that represent single and multiple link failures of a network has a special property that the OR-ing of any two vectors is also a vector in the set. Any set of vectors with the above property is called a vector span [10]. To solve the probe selection problem, we need to employ some special properties of a vector span. We, therefore, first study in Section 3.1 properties of general vector spans, and then show how these properties can be applied to develop an algorithm for the probes selection problem in Section 3.2.

3.1 Mathematical Basis

Let $\mathcal{D} = \{\vec{d}_i\}_{1 \leq i \leq h}$ be a set of binary vectors of equal length, and let $I = \{1, \dots, h\}$ be the index set of \mathcal{D} . A vector span \mathcal{S} can be defined on \mathcal{D} as follows.

Definition 2. [Vector span] *The vector span of \mathcal{D} is*

$$\mathcal{S} = \langle \mathcal{D} \rangle = \left\{ \bigvee_{i \in I} \alpha_i \cdot \vec{d}_i \mid \alpha_i \in \{0, 1\}, \vec{d}_i \in \mathcal{D} \right\}$$

where “ \vee ” denotes the binary max operation, and “ \cdot ” denotes the usual multiplication operation. Vectors in \mathcal{D} are called the *generator vectors* of \mathcal{S} .

On the set \mathcal{D} , we define the following independence property [10].

Definition 3. [IP] *The set $\mathcal{D} = \{\vec{d}_i\}_{i \in I}$ is independent if for all $i \in I$ and $I_2 \subseteq I \setminus \{i\}$, $\vec{d}_i \notin \langle \{\vec{d}_j\}_{j \in I_2} \rangle$.*

Merging Definition 2 and Definition 3, we obtain the following definition.

Definition 4. [Basis] A basis \mathcal{B} of a span \mathcal{S} is a set of independent vectors of \mathcal{S} such that $\langle \mathcal{B} \rangle = \mathcal{S}$.

Assume we have a span \mathcal{S} that is generated by a set of generator vectors \mathcal{D} and has a basis \mathcal{B} . The following lemma follows from Definition 4, and is needed to solve the probe selection problem.

Lemma 1. If \mathcal{D} is finite, then \mathcal{S} has a unique basis \mathcal{B} that is the subset of \mathcal{D} with smallest cardinality such that $\langle \mathcal{B} \rangle = \mathcal{S}$.

Proof. Wagneur [10] proved that spans over general vector sets have a unique basis, and hence this conclusion is also true for spans over binary vectors.

We prove the second assertion of the lemma by contradiction. Assume that there is a smaller subset of \mathcal{D} , namely \mathcal{B}' , which satisfies $\langle \mathcal{B}' \rangle = \mathcal{S}$, i.e., that there exists at least one vector \vec{v} of \mathcal{B} that does not belong to \mathcal{B}' . Let us denote by $I_{\mathcal{B}}$ the index set of \mathcal{B} , and by $I'_{\mathcal{B}}$ the index set of \mathcal{B}' . Let \vec{d}_i , $i \in I_{\mathcal{B}}$ be the vectors of \mathcal{B} . Since $\langle \mathcal{B}' \rangle = \mathcal{S}$, there exists a non empty subset $I_v \subseteq I'_{\mathcal{B}}$ such that:

$$\vec{v} = \bigvee_{i \in I_v} \vec{d}_i. \quad (1)$$

Furthermore, since \mathcal{B} is also a basis of \mathcal{S} , for each $\vec{d}_i \in \mathcal{B}'$ there is a nonempty subset $I_i \subseteq I_{\mathcal{B}}$ such that:

$$\vec{d}_i = \bigvee_{j \in I_i} \vec{d}_j. \quad (2)$$

Substituting (2) in (1) yields $\vec{v} = \bigvee_{j \in I_v} \vec{d}_j$, where $I_v = \bigcup_{i \in I_v} I_i$. Since \mathcal{B} is independent, the only case where this can happen is that there exists an index $k \in I_v$ such that: $\vec{v} = \vec{d}_k$ and $\vec{d}_k \vee \vec{d}_l = \vec{d}_k$ for all $l \in I_v \setminus \{k\}$. From (2), this implies that there exists $i \in I'_v$ such that $\vec{d}_i = \vec{v}$, which in turn indicates that $\vec{v} \in \mathcal{B}'$; a contradiction to the assumption $\vec{v} \notin \mathcal{B}'$. \square

3.2 Probe Selection Algorithm

Denote by $C(D_{max})$ and $R(D_{max})$ the set of column vectors and row vectors of the matrix D_{max} . Let $\langle C(D_{max}) \rangle$ be the span generated by column vectors, called column span of D_{max} , and let $\langle R(D_{max}) \rangle$ be the span generated by row vectors, called row span of D_{max} . A vector in $\langle C(D_{max}) \rangle$ represents subsets of \mathcal{E} whose failures generate the same set of failed probes. We call the set of all the subsets of \mathcal{E} whose failures are represented by the same vector a *failure set*. Two different vectors in $\langle C(D_{max}) \rangle$ represent two failure sets that are distinguished by \mathcal{P}_{max} . Therefore, its cardinality $|\langle C(D_{max}) \rangle|$ is the number of failure sets that can be distinguished by \mathcal{P}_{max} . Similarly, let D^* be the dependency matrix of the system with the set of probes \mathcal{P}^* . Let $R(D^*)$ and $C(D^*)$ be respectively the set of column vectors and row vectors of D^* . $|\langle C(D^*) \rangle|$ is the number of failure sets that can be distinguished by \mathcal{P}^* .

Since \mathcal{P}^* is a subset of \mathcal{P}_{max} , the number of failure sets that can be distinguished by \mathcal{P}^* is always less than or equal to the number of failure sets that can be distinguished by \mathcal{P}_{max} . Furthermore, any two subsets of \mathcal{E} that can be distinguished by \mathcal{P}^* can also be distinguished by \mathcal{P}_{max} . Thus, \mathcal{P}^* distinguishes any two subsets of \mathcal{E} that \mathcal{P}_{max} distinguishes if and only if the number of failure sets that are distinguishable by \mathcal{P}^* is equal to the number of failure sets that are distinguishable by \mathcal{P}_{max} . Consequently, the probe selection problem amounts to find \mathcal{P}^* such that the number of failure sets that can be distinguished by \mathcal{P}^* and by \mathcal{P}_{max} are equal. Since each failure set is respectively represented by a column of D^* or D_{max} , the solution of the probe selection problem is the smallest subset $R(D^*)$ of $R(D_{max})$ such that $|< C(D^*)>| = |< C(D_{max})>|$. Theorem 1 below gives the solution to the probe selection problem.

Theorem 1. *The solution to the probe selection problem is the set of probes whose corresponding rows in D_{max} form the basis of $< R(D_{max})>$.*

Proof. Let D^* be a matrix whose rows are the basis of the span $< R(D_{max})>$, i.e., such that $< R(D^*)> = < R(D_{max})>$. From [11], Theorem 1.2.3, the row span and column span of any binary matrix have the same cardinality. Therefore, $|< C(D^*)>| = |< R(D^*)>| = |< R(D_{max})>| = |< C(D_{max})>|$, which yields that $R(D^*)$ is a solution for the probe selection problem. Now, Lemmas 1 yields that $R(D^*)$ is the smallest subset of $< R(D_{max})>$ such that $|< R(D^*)>| = |< R(D_{max})>|$. Therefore, $R(D^*)$ is the unique solution to the probe selection problem. \square

We now give an algorithm, which we call the Probe Selection (PS) algorithm, that finds the basis of $< R(D_{max})>$. The *weight* of a vector is defined as the number of 1 entries in that vector. Let us denote the elements of $R(D_{max})$ by $\{\vec{r}_1, \vec{r}_2, \dots, \vec{r}_{n_{P_{max}}}\}$, where $n_{P_{max}} = |\mathcal{P}_{max}|$. The PS algorithm constructs the set $R(D^*)$ as follows.

The PS algorithm

Step 1: Initialize $R(D^*)$ to an empty set, and set an integer i to 1;

Step 2: Sort and re-index the vectors in $R(D_{max})$ in increasing weight order.

Step 3: Until $R(D_{max}) \neq \emptyset$ repeat the loop: remove \vec{r}_i from $R(D_{max})$; increase i by 1; if $\vec{r}_i \notin < R(D^*)>$, then append \vec{r}_i to $R(D^*)$.

Step 4: return $R(D^*)$.

Theorem 2. *The set of vectors $R(D^*)$ returned by the PS algorithm is the basis of $< R(D_{max})>$.*

Proof. First, we prove that the vectors in $R(D^*)$ are independent. Let us re-index the vectors \vec{r}_i in $R(D^*)$ in the order of inclusion to $R(D^*)$ by the PS algorithm. Step 3 of the PS algorithm prevents any $\vec{r}_i \in R(D^*)$ to be obtained by OR-ing any combination of vectors in $\{\vec{r}_{i-1}, \vec{r}_{i-2}, \dots, \vec{r}_1\}$. Furthermore, \vec{r}_i is not an OR-ing of any combination of vectors in $\{\vec{r}_{i+1}, \dots, \vec{r}_{|R(D^*)|}\}$. Indeed, if this was true, then the weight of \vec{r}_i would be larger than the weight of these vectors, which is impossible because by construction the weight of \vec{r}_i is smaller than or equal to the weight of \vec{r}_j for all $j > i$. From the above results, any

vector $\vec{r}_i \in R(D^*)$ is not an OR-ing of any combination of other vectors in $R(D^*)$, hence the set $R(D^*)$ is independent.

Second, we prove that $\langle R(D^*) \rangle = \langle R(D_{max}) \rangle$. Indeed, by construction, any vector in the set $\{\vec{r}_1, \dots, \vec{r}_{n_{P_{max}}}\}$ either belongs to $R(D^*)$, or is an OR-ing of some combinations of vectors in $R(D^*)$. \square

Note here that for any two nodes s and t , if $P_{t,s}$ and $P_{s,t}$ contain the same links, then the two row vectors that represent the probe from s to t , and the probe from t to s are equal. Furthermore, the vector that represents the path $P_{t,s}$ can only be generated by probes from at most two beacons (at s and t).

4 Beacon Placement Problem

We have shown in Section 3 that given a set of beacons \mathcal{V}_B , there is an optimal set of row vectors $R(D^*)$ of the dependency matrix that the beacons need to generate for multiple failure diagnosis. Once the optimal set of row vectors $R(D^*)$ is determined, to generate these vectors we may not need all the available beacons. We are now interested in finding the minimal number of beacons needed to generate the probes corresponding to the vectors in $R(D^*)$; this is the beacon placement problem defined as follows.

Definition 5. [BP problem] Given a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with \mathcal{V}_B possible positions where we can place the beacons, the beacon placement (BP) problem is the determination of the smallest subset \mathcal{V}^* of \mathcal{V}_B such that any two sets of links \mathcal{E}_1 and \mathcal{E}_2 that are distinguished by probes of \mathcal{V}_B are also distinguished by probes of \mathcal{V}^* .

We solve the beacon placement problem as follows. We first assume that all nodes in \mathcal{V}_B are used as beacons and find the optimal set of row vectors $R(D^*)$ for this setting using the PS algorithm described in Section 3. We then look for the smallest subset \mathcal{V}^* of \mathcal{V}_B such that the set of vectors corresponding to probes from nodes in \mathcal{V}^* contains $R(D^*)$.

4.1 Hardness of the Beacon Placement Problem

Theorem 3. The beacon placement problem is NP-hard.

Proof. The proof for this theorem is straightforward when we realize that in a network that uses shortest path routing with all links have equal weight, and with $\mathcal{V}_B = \mathcal{V}$, the BP problem reduces to the well-known vertex cover problem [12]. \square

4.2 A Greedy Algorithm for the Beacon Placement Problem

We now present a greedy beacon placement (BP) algorithm, which constructs \mathcal{V}^* , given $R(D^*)$. Let \mathcal{V}_{approx} be the set of beacons returned by the BP algorithm, and let \mathcal{A} and \mathcal{C} be two subsets of $R(D^*)$. For a beacon u , let us denote by $\mathcal{R}(u)$

the set of all vectors representing probes that can be generated from u . The steps of the BP algorithm are as follows.

The BP algorithm

Step 1: Initialize \mathcal{V}_{approx} and \mathcal{A} to empty sets, \mathcal{C} to $R(D^*)$.

Step 2: Until $\mathcal{C} \neq \emptyset$ repeat the loop: pick a vector $\vec{r} \in \mathcal{C}$ and append \vec{r} to \mathcal{A} ; if for all nodes u in \mathcal{V}_B , $\vec{r} \in \mathcal{R}(u)$, then include u in \mathcal{V}_{approx} and remove all the vectors in $\mathcal{R}(u)$ from \mathcal{C} .

Step 3: Return \mathcal{V}_{approx} .

The BP algorithm is said to have an approximation ratio $\rho(n)$ if for every graph \mathcal{G} with n nodes, $|\mathcal{V}_{approx}|/|\mathcal{V}^*| \leq \rho(n)$.

Theorem 4. *The BP algorithm has a constant approximation factor of 2.*

Proof. From Step 2 of the algorithm, for each vector in \mathcal{A} , we add one or two nodes (as explained at the end of Section 3.2, there is a maximum of two beacons that can send probes taking the same path and hence are represented by the same vector) to \mathcal{V}_{approx} , hence $|\mathcal{V}_{approx}| \leq 2 \cdot |\mathcal{A}|$ (*).

Any solution must include, for every vector in \mathcal{A} , at least one beacon that can generate it. Furthermore, by construction, no two vectors in \mathcal{A} can be generated by the same beacon. So the optimal beacon selection \mathcal{V}^* is of size $|\mathcal{V}^*| \geq |\mathcal{A}|$ (**).

Combining (*) and (**), we get $|\mathcal{V}_{approx}| \leq 2 \cdot |\mathcal{V}^*|$. □

5 Experimental Study

We apply our algorithms on ISP topologies that are inferred by the Rocketfuel tool [2]. We investigate the number of beacons and the number of probes required for multiple failures diagnosis by our algorithms to that needed for single failure diagnosis by algorithms in [6] on three backbone ISP topologies with sizes ranging from small (Exodus: 80 nodes and 147 links) to medium (Telstra: 115 nodes and 153 links), and large (Tiscali: 164 nodes and 328 links). For the sake of simplicity, we assume that all the ISPs use shortest path routing to route traffic.

Recall that n is the number of nodes in the network. In our experiments, the number of nodes that can be used as beacons (beacon candidates) $|\mathcal{V}_B|$ is varied from $n/100$ to n . We select the beacon candidates randomly by picking a random permutation of the set of nodes in the network. After building the dependency matrix as in Section 3, we run the PS algorithm to find the optimal set of probes and then the BP algorithm to find the set of beacons that are required to generate these probes. We also run the algorithms described in [6] to find the set of beacons and the set of probes for single fault diagnosis. For space constraints, we only present in this paper the results for the Tiscali topology. Similar results are obtained for the two other topologies. In Fig. 1, we plot the percentage of nodes that are actually used as beacons for multiple failures diagnosis and single failure diagnosis. We also plot the percentage of useful probes returned by the PS algorithm and the single fault algorithm [6] in Fig. 2.

The number of useful beacons $|\mathcal{V}^*|$ for multiple failures diagnosis is notably smaller than the number of possible beacon nodes for all sizes of the set $|\mathcal{V}_B|$ of

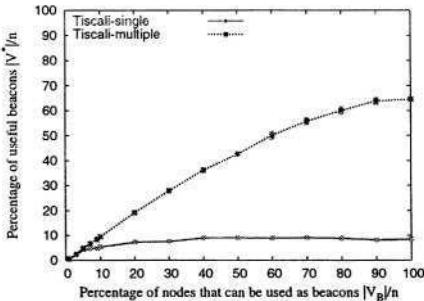


Fig. 1. The number of beacons for single and multiple failure(s) diagnosis.

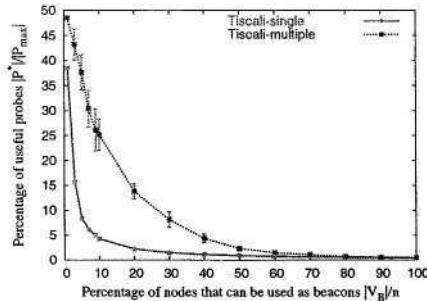


Fig. 2. The number of probes for single and multiple failure(s) diagnosis.

beacon candidates. This is especially true when the number of beacon candidates is large. We also observe a fast increase in the number of beacons useful for multiple failure diagnosis when the number of beacon candidates increases. This can be explained by the increase in number of multiple failures that can be distinguished by the beacons. However, for single fault diagnosis, the number of beacons increases very slowly, as probes from even a small set of beacons are enough to detect and locate all single failures.

We observe that for all sizes of the set of beacon candidates, the number of useful probes $|\mathcal{P}^*|$ is less than a half of the total number of probes $|\mathcal{P}_{max}|$ that can be sent in the network. The percentage of useful probes also decreases rapidly as the number of beacons increases for both multiple and single failure(s) cases. This observation can be explained for both the single and multiple failure cases as follows. In the single failure case, the number of useful probes remains relatively unchanged (approximately equal to the number of links) for various numbers of beacon candidates. Hence, when the total number of probes that can be sent in the network increases with the number of beacon candidates, the percentage of useful probes decreases. In the multiple fault case, the number of useful probes decreases as the number of beacon candidates increases, because these additional beacon candidates make it possible to replace a number of different probes by a single probe that distinguishes the same failure sets. There remains however a large difference in the number of probes for multiple and single failure diagnosis in Fig. 2.

6 Conclusions

In this paper, we have investigated the use of active probing for link failure diagnosis in ISP networks. We have shown that for multiple failures diagnosis the optimal set of probes can be found in polynomial time. On the contrary, the problem of optimizing the number of beacons for multiple failures diagnosis is NP-hard. We have studied the performance of our algorithms on existing ISP topologies. Our studies show that there is a great reduction in the number of probes and beacons that are available and the number of probes and beacons

that are actually useful for multiple failures diagnosis. However, there are also remarkable differences in the number of probe and the number of beacons for single and multiple fault diagnosis in all analyzed ISP topologies. These figures can be used by ISP operators to trade off accuracy of diagnosis for probing costs.

We are working on various extensions of the present work. We obtained in this paper the optimal set of useful probes. The next step is to determine the failure detection capacity of the optimal set of probes, that is, to find the number of failures of a given multiplicity (double, triple, etc.) that can be detected and located by this probing scheme. We are also investigating how an active probing scheme can cope with missing and false probes. The challenge is to improve the robustness of the probing scheme to probe errors without much increasing in the probing costs.

References

1. A.Markopoulou, G.Iannaccone, S.Bhattacharyya, C.N.Chuah, C.Diot: Characterization of Failures in an IP Backbone. Proc. IEEE INFOCOM'04 (2004)
2. N.Spring, R. Mahajan, D.Wetherall: Measuring ISP topologies with Rocketfuel. Proc. of ACM SIGCOMM (2002)
3. G.Govindan, H.Tangmunarunkit: Heuristics for internet map discovery. Proc. IEEE INFOCOM'00 (2000)
4. S.Jamin, C.Jin, Y.Jin, D.Raz, L.Zhang: On the placement of internet instrumentation. Proc. IEEE INFOCOM'00 (2000)
5. M.Adler, T.Bu, R.K.Sitaraman, D.Towsley: Tree layout for internal network characterizations in multicast networks. Networked Group Comm. (2001) 189–204
6. Y.Bejerano, R.Rastogi: Robust monitoring of link delays and faults in IP networks. Proc. IEEE INFOCOM'03 (2003)
7. J.Horton, A.Lopez-Ortiz: On the number of distributed measurement points for network tomography. IMC'03 (2003)
8. S.W.Richard: TCP/IP illustrated. Addison-Wesley Publishing Company (1994)
9. CAIDA: Cooperative Association for Internet Data Analysis.
(<http://www.caida.org/Tools/Skitter>)
10. E.Wagneur: Moduloids and pseudomodules: 1 dimension theory. Discrete Mathematics **98** (1991) 57–73
11. J.Kim : Boolean Matrix theory and applications. Marcel Dekker (1982)
12. M.R.Garey, D.S.Johnson: Computers and Intractability: A guide to the theory of NP-completeness. W. H. Freeman (1979)

Structured Errors in Optical Gigabit Ethernet Packets

Laura James¹, Andrew Moore², and Madeleine Click³

¹ Centre for Photonic Systems, University of Cambridge
lbj20@eng.cam.ac.uk

² Computer Laboratory, University of Cambridge
andrew.moore@cl.cam.ac.uk
³ Intel Research Cambridge
madeleine.glick@intel.com

Abstract. This paper presents a study of the errors observed when an optical Gigabit Ethernet link is subject to attenuation. We use a set of purpose-built tools which allows us to examine the errors observed on a per-octet basis. We find that some octets suffer from far higher probability of error than others, and that the distribution of errors varies depending on the type of packet transmitted.

1 Introduction

Optical communications assessment is conventionally based upon Bit Error Rate (BER), a purely statistical measurement. Higher level network design decisions are often based on an assumption of uniform error at the physical layer, with errors occurring independently and with equal probability regardless of data value or position.

We examine the assumption of uniform behaviour in Gigabit Ethernet at low optical power, and find that the failure mode observed is non-uniform, caused by interactions between the physical layer, the coding system and the data being carried.

1.1 Motivation

The assumed operating environment of the underlying coding scheme must be re-examined as new more complex optical systems are developed. In these systems containing longer runs of fibre, splitters, and active optical devices, the overall system loss will be greater than in today's point-to-point links, and the receivers may have to cope with much lower optical powers. Increased fibre lengths used to deliver Ethernet services, e.g. the push for Ethernet in the first mile [1], and switched optical networks [2] are examples of this trend.

The design of the physical and data link layers affects how resilient a network is to errors in the communications medium. Understanding the behaviour of the medium and probable error types and distributions, is necessary to design a network that will avoid error causes, and compensate for others. Coding schemes

such as 8B/10B have many desirable properties, and therefore a thorough understanding of the effects of error is important for future optical network design.

1.2 Gigabit Ethernet Physical Coding Sublayer

Using a transmission code improves the resilience of a communications link, by ensuring the data stream has known characteristics that are well matched to the physical behaviour of the link. A coding scheme must ensure the recovery of transmitted bits; often this requires a minimum number of bit transitions to occur for successful clock and data recovery. In most systems, transceivers are AC coupled, which can lead to distorted pulses and baseline wander (as the DC component of the signal builds up); these can be reduced by the use of a block code which is symmetrical around the zero line.

The 8B/10B codec, originally described by Widmer & Franaszek [3] sees use in 1000BASE-X, optical Ethernet. This scheme converts 8 bits of data for transmission (ideal for any octet-orientated system) into a 10 bit line code. Although this adds a 25% overhead, 8B/10B has many valuable properties; a transition density of at least 3 per 10 bit code group and a maximum run length of 5 bits for clock recovery, along with virtually no DC spectral component. These characteristics also reduce the possible signal damage due to jitter, which is particularly critical in optical systems, and can also reduce multimodal noise in multimode fibre connections.

This coding scheme is royalty-free and well understood, and is currently used in a wide range of applications; in addition to being the standard for optical gigabit Ethernet, it is used in the Fibre Channel system [4], and 8B/10B coding will be the basis of coding for the electrical signals of the upcoming PCI Express standard [5].

8B/10B Coding. The 8B/10B codec defines encodings for data octets, and control codes which are used to delimit the data sections and maintain the link. Individual codes or combinations of codes are defined for Start of Packet, End of Packet, line Configuration, and so on. Also, Idle codes are transmitted when there is no data to be sent, and these keep the transceiver optics and electronics active. The Physical Coding Sublayer (PCS) of the Gigabit Ethernet specification [6] defines how these various codes are used.

Individual ten bit code-groups are constructed from the groups generated by 5B/6B and 3B/4B coding on the first five and last three bits of a data octet respectively. Some examples are given in Table 1; the running disparity is the sign of the running sum of the code bits, where a one is counted as 1 and a zero as -1. During an Idle sequence between packet transmissions, the running disparity is changed (if necessary) to -1, and then maintained at that value. Both control and data codes may change the running disparity, or may preserve its existing value; examples of both types are shown in Table 1. The code-group used for the transmission of an octet depends upon the existing running disparity – hence the two alternative codes given in the table. A received code-group is compared

against the set of valid code-groups for the current receiver running disparity, and decoded to the corresponding octet if it is found. If the received code is not found in that set, the specification states that the group is deemed invalid. In either case, the received code-group is used to calculate a new value for the running disparity. A code-group received containing errors may thus be decoded and considered valid. It is also possible for an earlier error to throw off the running disparity calculation, such that a later code-group may be deemed invalid, as the running disparity at the receiver is no longer correct. This can amplify the effect of a single bit error at the physical layer. Line coding schemes, although

Table 1. Examples of 8B/10B control and data codes

Type	Octet	Octet bits	Current RD -	Current RD +	Note
data	0x00	000 00000	100111 0100	011000 1011	preserves RD value
data	0xf2	111 10010	010011 0111	010011 0001	swaps RD value
control	K27.7	111 11011	110110 1000	001001 0111	preserves RD value
control	K28.5	101 11100	001111 1010	110000 0101	swaps RD value

they handle many of the physical layer constraints, can introduce problems. In the case of 8B/10B coding, a single bit error on the line can lead to multiple bit errors in the received data byte. For example, with one bit error the code-group DO.1 (current running disparity negative) becomes the code-group D9.1 (also negative disparity); these decode to give bytes with 4 bits of difference. In addition, the running disparity after the code-group is miscalculated, potentially leading to future errors. There are other similar examples [6].

2 Method

In this paper we investigate Gigabit Ethernet on optical fibre, (1000BASE-X [6]) when the receiver power is sufficiently low as to induce errors in the Ethernet frames. We assume that while the CRC mechanism within Ethernet is sufficiently strong as to catch the errors, the dropped frame and resulting packet loss will result in hosts, applications and perhaps users whose packets will be in error with a significantly higher probability than the norm.

In our main test environment an optical attenuator is placed in one direction of a Gigabit Ethernet link. A traffic generator feeds a Fast Ethernet link to an Ethernet switch, and a Gigabit Ethernet link is connected between this switch and a traffic sink and tester. The variable optical attenuator is placed in the fibre in the direction from the switch to the sink.

A packet capture and measurement system is implemented within the traffic sink using an enhanced driver for the SysKonnect SK-9844 network interface card (NIC). Among a number of additional features, the modified driver allows

application processes to receive frames that contain errors that would normally be discarded. Alongside purpose-built code for the receiving system we use a special-purpose traffic generator. Pre-constructed test data in tcpdump-format file is transmitted from one or more traffic generators using *tcpfire* [7]. By transmitting a pre-determined traffic stream we can identify specific errored octets within the received data stream.

2.1 Octet Analysis

Each octet for transmission is coded using 8B/10B into a 10 bit code group or *symbol*, and we analyze these for frames which are received in error at the octet level. By comparing the two possible transmitted symbols for each octet in the original frame, to the two possible symbols corresponding to the received octet, we can deduce the bit errors which occurred in the symbol at the physical layer. In order to infer which symbol was sent and which received, we assume that the combination giving the minimum number of bit errors on the line is most likely to have occurred. This allows us to determine the line errors which most probably occurred.

Various types of symbol damage may be observed. One of these is the single bit error caused by the low signal to noise ratio at the receiver. A second form of error results from a loss of bit clock causing *smeared* bits: where a subsequent bit is read as having the value of the previous bit. A final example results from the loss of symbol clock synchronization. This can result in the symbol boundaries being misplaced, so that a sequence of several symbols, and thus several octets, will be incorrectly recovered.

2.2 Real Traffic

Some experiments are conducted with real network traffic referred to as the *day-trace*. This network traffic was captured from the interconnect between a large University department and that universities' principle data-backbone over the course of two working days. We consider it to contain a representative sample of network traffic for an academic/research department network with approximately 150 users.

The probability of occurrence of each octet value within the day-trace is given in Figure 1. The illustrated probabilities allow specific insight into the effect of symbol-errors within a realistic traffic workload.

Other traffic tested included *pseudo-random data*, consisting of a series of 1500 octet frames each filled with octets whose values were each drawn from a pseudo-random number generator. *Structured test data* consists of a single frame containing repeated octets: 0x00–0xff, to make a frame 1500 octets long. The *low error testframe* consists of 1500 octets of 0xCC data (selected for a low symbol error rate); the *high error testframe* is 1500 octets of 0x34 data (which displays a high symbol error rate).

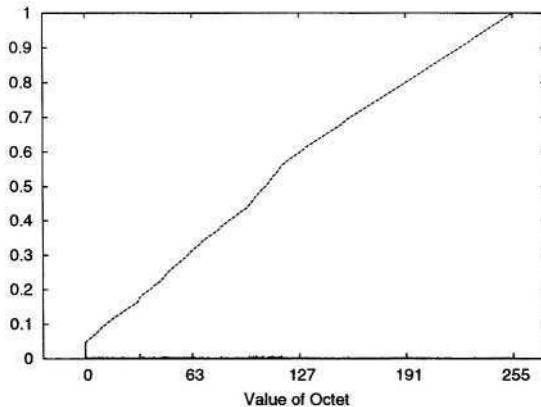


Fig. 1. Probability of occurrence for a particular octet within *daytrace*

2.3 Bit Error Rate Measurements

In photonics laboratories, a Bit Error Rate Test kit (BERT) is commonly used to assess the quality of an optical system [8]. This comprises both a data source and a receiving unit, which compares the incoming signal with the known transmitted one. Individual bit errors are counted both during short sampling periods and over a fixed period (defined in time, bits, or errors). The output data can be used to modulate a laser, and a photodiode can be placed before the BERT receiver to set up an optical link; optical system components can then be placed between the two parts of the BERT for testing. Usually, a pseudo-random bit sequence is used; but any defined bit sequence can be transmitted repeatedly and the error count examined.

For the bit error rate measurements presented here, a directly modulated 1548nm laser was used (the wavelength of 1000BASE-ZX). The optical signal was then subjected to variable attenuation, before returning via an Agilent Lightwave (11982A) receiver unit into the BERT (Agilent parts 70841B and 70842B). The BERT was programmed with a series of bit sequences, each corresponding to a frame of Gigabit Ethernet data, encoded as it would be for the line in 8B/10B. Purpose-built code is able to construct the bit-sequence, suitable for the BERT, from a frame of known data. The bit error rates for these various packet bit sequences were measured at a range of attenuation values, using identical BERT settings for all frames (eg. 0/1 thresholding value).

3 Results

A plot of the cumulative distribution of errors for three traffic types is shown in Figure 2. Note that while the random data frames closely follow the expected uniform error distribution, the *day-trace* frames suffer from higher error rates in the low value octets, especially the value 0x00. The structured test data shows

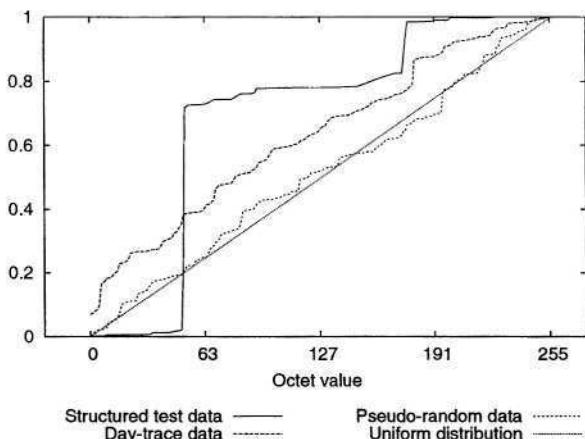


Fig. 2. Cumulative distribution of errors versus octet value

an even more significant error rate focused upon only a small number of octets (e.g., 0x34). The test attenuation used here corresponds to a frame loss of 892 in 10^6 pkts for structured test data, to 233 in 10^6 pkts for pseudo-random data, and to 98 in 10^6 pkts for the *day-trace* data.

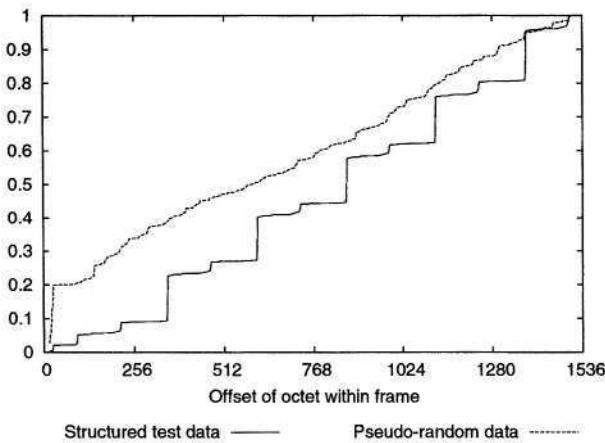


Fig. 3. Cumulative distribution of errors versus position in frame

Figure 3 contrasts the position of errors in the pseudo-random and structured frames. The *day-trace* results are not shown as this data, consisting of packets of varying length, is not directly comparable. The positions of the symbols most

subject to error can be clearly observed. In contrast with the structured data, the random data shows a much increased error rate at the beginning of the frame.

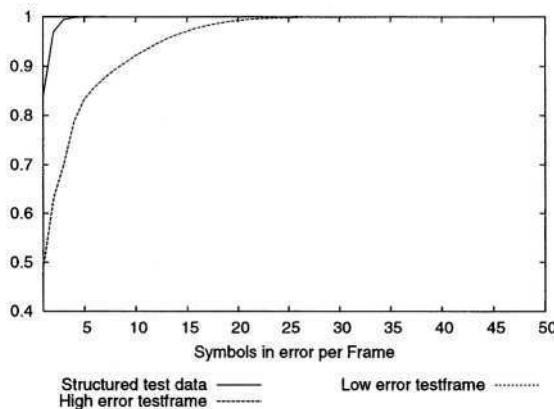


Fig. 4. Cumulative distribution of symbol errors per frame

Figure 4 indicates the number of incorrectly received symbols per frame. The *low error testframe* generated no symbols in error, and thus no frames in error. The *high error testframe* results show an increase in errored symbols per frame. It is important to note that the errors occur uniformly across the whole packet, and there are no correlations evident in the position of errors within the frame.

Figure 5 shows the frequency of received packets in error, for a range of receiver optical power and for the three different testframes.

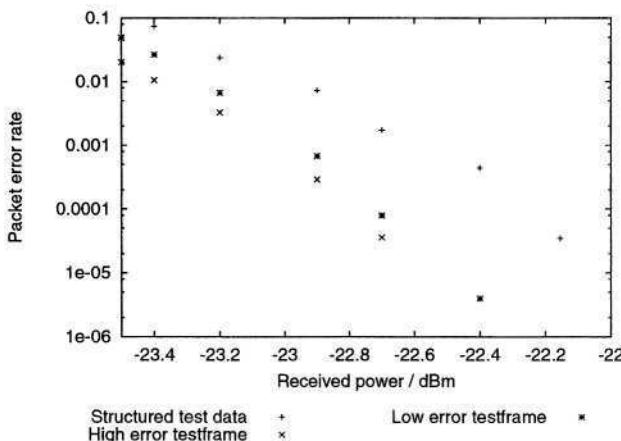


Fig. 5. Frequency of packet error versus received power

Figure 6 shows the bit error rate, as measured using the BERT, for a range of receiver optical power. These powers are different to those in Figure 5 due to the different experimental setup; packet error is measured using the SysKonnect 1000BASE-X NIC as a receiver, and bit error with an Agilent Lightwave receiver, which have different sensitivities.

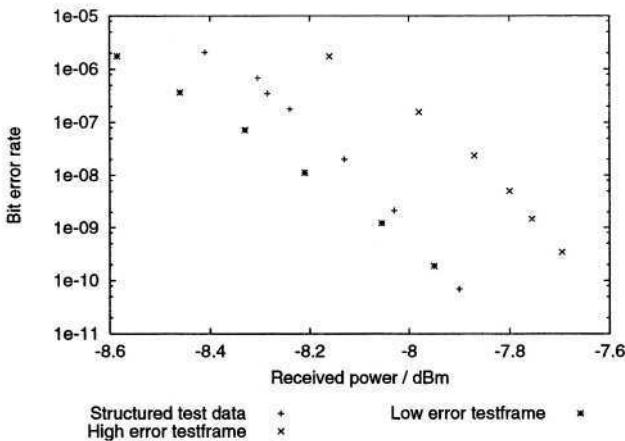


Fig. 6. Bit error rate versus received power

Figures 5 and 6 show clearly different error rates for the different frames transmitted. A frame giving unusually high bit error rates at the physical layer does not necessarily lead to high rates of packet error.

4 Discussion

From Figure 2, it can be seen that 0x00 data suffers from greater error than the assumed uniform distribution would suggest when transmitted as part of a real traffic stream. Given the high proportion of 0x00 octets in genuine traffic (Figure 1), this effect is likely to be amplified. When the position of errors within a frame is considered in Figure 3, we find that pseudo-random data is particularly subject to errors at the start of the frame (compared to the structured testframe, where errors occur at certain octets throughout the frame). If this behaviour also occurred in real frames, it may be the header fields which would be most subject to error. We consider example frames which consist of octets leading to high and low symbol error rates, and find that a higher symbol error probability leads to an increased number of errors per frame (Figure 4). Regardless of the pattern or data structure in which these octets are found in the frame, the individual likelihoods of their being received in error still have a noticeable effect.

When we compare the packet error rates obtained using our main test system with the bit error rates from the BERT measurements (Figures 5 and 6), we see

that these different testframes lead to substantially different BER performance. Also, the BER is no indicator of the packet error probability.

We thus find that some octets suffer from a far higher probability of error than others, and that the distribution of errors varies depending on the type of packet transmitted. When the network carrying data is operating in a regime of limited power at the receiver, the initial implication of our work is that certain combinations of data will be subject to a significantly increased error rate. We conjecture that the PCS of Gigabit Ethernet can cause *hot-spots* to occur within the data symbols, hot-spots that contradict the underlying assumption of uniformity of error. The result is that, in our example, some Ethernet frames carried over 1000BASE-X upon the 8B/10B coding scheme will suffer a higher rate of error than might otherwise be assumed. More specifically, an Ethernet frame carrying certain octets, early in the payload, is up to 100 times more likely to be received with errors (and thus dropped), than if the payload does not contain such *hot-spot* octets. In addition, we observe that the structure of the packet can worsen this effect.

The error *hot-spots* we have observed may have implications for higher level network protocols. Frame-integrity checks, such as a cyclic redundancy check, assume that there will be a uniformity of errors within the frame, justifying detection of single-bit errors with a given precision. One example: Stone *et al.* [9] provides insight into such non-uniformity of data, discussing the impact this has for the checksum of TCP. These results may call into question our assumption that only increased packet-loss will be the result of the error *hot-spots*. Instead of just lost packets, Stone *et al.* noted certain “unlucky” data would rarely have errors detected. Another example of related work is Jain [10] which illustrates how errors will impact the PCS of FDDI and require specific error detection/recovery in the data-link layer.

In further related work, researchers have observed that up to 60% of faults in an ISP-grade network are due to optical-events [11]; while the majority of these will be catastrophic events, we would speculate that including the hot-spotting we observed, a higher still optically-induced fault rate will exist.

We highlight here an unexpected failure mode that occurs in the low-power regime, inducing at worst: errors, and at best: poor performance, that may focus upon specific networks, applications and users.

5 Conclusion

We have shown that the errors observed in Gigabit Ethernet in a low-power regime are not uniform as may be assumed, and that some packets will suffer greater loss rates than the norm. This content-specific effect will occur without a total failure of the network, and so must be given careful consideration. Even running the system above the power limit, with a BER of 10^{-12} , a line error may occur as often as every 800 seconds. Our observation that real network traffic exhibited non-uniform symbol damage suggests that optical networks carrying

real traffic will suffer hot-spots of higher frame loss rate than the physical layer conditions would indicate. Future work will clarify these effects.

Acknowledgements. Many thanks to Derek McAuley, Richard Penty, Dick Plumb, Adrian P. Stephens, Ian White and Adrian Wonfor for their assistance, insight and feedback throughout the preparation of this paper. Laura James would like to thank Marconi Communications for their support of her research. Andrew Moore would like to thank Intel Corporation for their support of his research.

References

1. IEEE: IEEE 802.3ah — Ethernet in the First Mile (2003) Task Force.
2. McAuley, D.: Optical Local Area Network. In Herbert, A., Spärck-Jones, K., eds.: Computer Systems: Theory, Technology and Applications. Springer-Verlag (2003)
3. Widmer, A.X., Franaszek, P.A.: A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code. IBM Journal of Research and Development **27** (1983) 440–451
4. The Fibre Channel Association: Fibre Channel Storage Area Networks. LLH Technology Publishing, Eagle Rock, VA (2001)
5. Solari, E., Congdon, B.: The Complete PCIExpress Reference. Intel Press, Hillsboro, OR (2003)
6. IEEE: IEEE 802.3z — Gigabit Ethernet (1998) Standard.
7. tcpfire (2003) <http://www.nprobe.org/tools/>.
8. Ramaswami, R., Sivarajan, K.N. In: Optical Networks. Morgan Kaufmann (2002) 258–263
9. Stone, J., Greenwald, M., Partridge, C., Hughes, J.: Performance of Checksums and CRCs over Real Data. In: Proceedings of ACM SIGCOMM 2000, Stockholm, Sweden (2000)
10. Jain, R.: Error Characteristics of Fiber Distributed Data Interface (FDDI). IEEE Transactions on Communications **38** (1990) 1244–1252
11. Markopoulou, A., Iannaccone, G., Bhattacharyya, S., Chuah, C.N., Diot, C.: Characterization of failures in an IP backbone. In: To appear in IEEE Infocom, Hong Kong (2004)

Flow Clustering Using Machine Learning Techniques

Anthony McGregor^{1,2}, Mark Hall¹, Perry Lorier¹, and James Brunskill¹

¹ The University of Waikato, Private BAG 3105, Hamilton, New Zealand

mhall@cs.waikato.ac.nz,

<http://www.cs.waikato.ac.nz/>

² The National Laboratory of Applied Network Research (NLANR), San Diego Supercomputer Center, University of California San Diego, 10100 Hopkins Drive, CA 92186-0505, USA
tonym@nlanr.net,
<http://www.nlanr.net/>

Abstract. Packet header traces are widely used in network analysis. Header traces are the aggregate of traffic from many concurrent applications. We present a methodology, based on machine learning, that can break the trace down into clusters of traffic where each cluster has different traffic characteristics. Typical clusters include bulk transfer, single and multiple transactions and interactive traffic, amongst others. The paper includes a description of the methodology, a visualisation of the attribute statistics that aids in recognising cluster types and a discussion of the stability and effectiveness of the methodology.

1 Introduction

Passive header trace measurements, like the ones performed by NLANR/MNA[7] and WAND[8] produce a detailed record of the packet by packet behaviour of all traffic on a link. These traces have been used in a wide range of computer network research.

A packet header trace is an aggregate of the packets produced by many network processes. There are several techniques that can be used to disaggregate a packet trace. Simple approaches divide packet headers into classes based on some header field, often the protocol or port number. The general notion of a *flow* of packets[9] (where a flow roughly corresponds to a sequence of packets related to a single application exchange) is also well known and widely used. Different classifications support different uses of a packet header trace. In this paper we introduce a new methodology for classification of the packet headers that divides the traffic into similar application types (single transaction, bulk transfer etc). Our target analysis is workload generation for simulation, however we believe the technique has much wider application.

Packet traces may be used as raw material for driving simulations. We are interested in using the traffic captured on a network to answer “what if” questions about the network’s performance under workloads derived from the one we

captured. We wish to allow a network manager to understand the major types of traffic on the network and then discover how the network is likely to perform if the source, destination, quantity and proportions of those traffic types changes through simulation. Central to this work is the ability to decompose captured traffic into its component traffic types. As noted above, several different decompositions are possible but for this work we are interested in a decomposition which reflects the workload generating the traffic, rather than characteristics of the network, its protocols or the total traffic profile.

The most obvious classification (by IP protocol and port) was rejected for three reasons. The first is that within a single protocol (e.g. HTTP) there may be several quite distinct classes of traffic. For example, HTTP traffic includes fetches of small objects, such as HTML pages and icons, as well as large file transfers and tunnelled applications. The second, is that similar traffic types may use different protocols. Fetching a large file by HTTP or FTP, for example, has very similar characteristics. The final reason is that the protocol and port numbers may not be available. Tunnels, especially encrypted tunnels like IPSec, obscure this information.

2 Packet Interarrival/Size Plots

In our quest for a generic classification methodology we first examined plots of packet size against packet interarrival time (IAT) for the two unidirectional flows that make up a single ‘connection’. (We define a unidirectional flow in the conventional sense of a series of packets sharing the same five-tuple (source IP address and port, destination IP address port, and protocol number). We do not timeout flows, except where they exceed the length of the trace (6 hours). From this point in this paper, we will refer to these pairs of unidirectional flows as a bidirectional flow, or just a flow.

The IAT/packet size plots exhibit a number of characteristic shapes that we believe are indicative of the application type. Examples of these plots (produced from the Auckland-VI[10]) trace are shown in figure 1¹. Only four example plots are shown here, however there are other types that we have not shown due to space limitations. To illustrate the point that the same protocol may carry different traffic types, two different HTTP sessions are included (1(a) and (b)). In the following paragraphs we explain the most likely causes for the major characteristics of these plots. Our analysis is not exhaustive and is primarily intended to illustrate the point that some (but not all) of these characteristics are indicative of the type of application that generated the traffic. For simplicity, we have stated our analysis in the imperative. There were approximately 20,000 flows in the trace we analysed but less than a dozen of these characteristics plot types (plus a some plots that did not fit any characteristic type).

Fig 1(a) shows a flow containing a single HTTP request for a large object. There is a single large packet from the server to the client (the HTML GET request) and many small packets (the TCP acknowledgements). There are many large packets of the same size (full data packets) from the server to the client.

¹ A colour version of this paper is available at <http://www.wand.net.nz/pubs.php>.

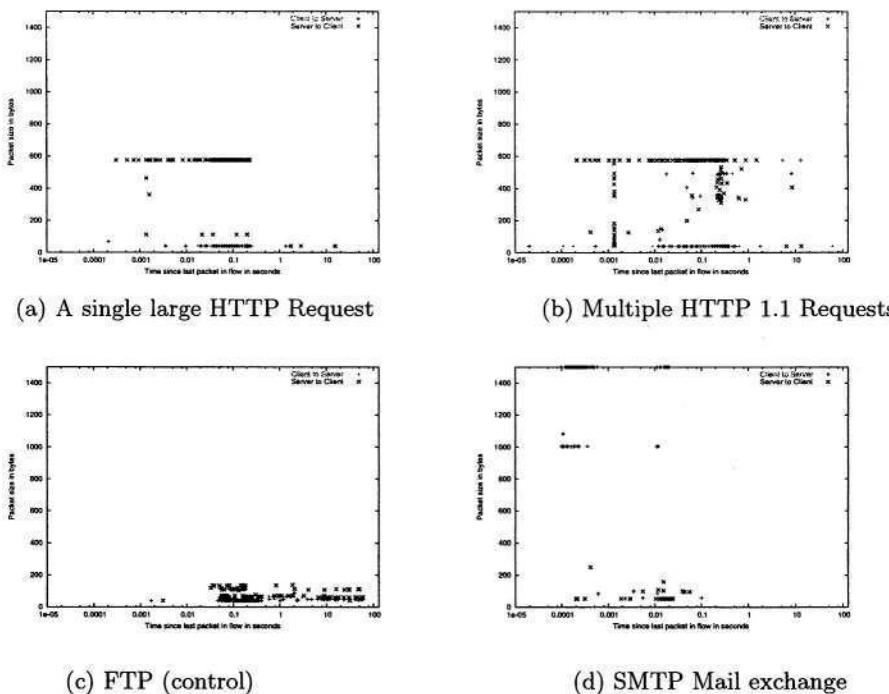


Fig. 1. HTTP Packet Interarrival/Size plots

There are also a few smaller packets (partly full data packets) and a single minimum sized packet (an ACK without data) from the server to the client.

Fig 1(b) shows a flow containing multiple HTTP requests in a single connection (HTTP 1.1). In addition to the packets of the Fig 1(a) there are additional request packets (mostly ranging between 300 and 600 bytes). Notice the cluster of packets centred roughly around ($size = 500, IAT = 0.3s$). This type of cluster is typical of protocols in query/reply/query transaction mode and is the result of fetching small objects via HTTP. Finally, note the vertical grouping at approximately $IAT = 1.1ms$. This is probably the result of the TCP Nagle Algorithm operating to delay the transmission of partly full packets until the previous packet is acked.

Fig 1(c) is a similar plot for an FTP control connection. It shows a cluster of points around $IAT = 100ms$ and then a spread of points above that. The cluster is again a query/response/query cluster (but with smaller queries and responses than the multiple HTTP request example). The spread, which ranges up to about a minute, is related to human interaction times required to generate a new command. Because FTP uses a separate connection for data transfer there is no bulk data phase for this flow.

The final plot in the set, Fig 1(d), shows a mail transfer using SMTP. Again there is a transaction cluster (but with a wider IAT range, indicating that some queries took longer to resolve than others). There is also a bulk transfer com-

ponent with large packets. The reason for some of the large packets being 1500 bytes and others being 1004 bytes was investigated at some length. The 1004 byte packets are final packet in a repeated series of transfers. That is, this SMTP session contained 25 transfers of messages requiring five 1500 byte and one 1004 byte packets to transfer. Given SMTP's multiple recipient feature and the normal variation in length of email addresses in the message header, this is almost certainly spam.

3 Clustering and Classification

While it would be possible to form groups of flows by writing code that was aware of the characteristics we discovered in the IAT/Packet size plots, this approach imposes a high degree of human interpretation in the results. It is also unlikely to be sufficiently flexible to allow the methodology to be used in diverse network types. Machine learning techniques can also be used to cluster the flows present in the data and then to create a classification from the clusters. This is a multiple step process. The data is first divided into flows as described above. A range of attributes are extracted from each flow. These attributes are:

- packet size statistics (minimum, maximum, quartiles, minimum as fraction of max and the first five modes)
- interarrival statistics
- byte counts
- connection duration
- the number of transitions between transaction mode and bulk transfer mode, where bulk transfer mode is defined as the time when there are more than three successive packets in the same direction without any packets carrying data in the other direction
- the time spent: idle (where idle time is the accumulation of all periods of 2 seconds or greater when no packet was seen in either direction), in bulk transfer and in transaction mode

These characteristics are then used by the EM clustering algorithm (described in section 3.1 below) to group the flows into a small number of clusters.

This process is not a precise one. To refine the clusters we generate classification rules that characterise the clusters based on the raw data. From these rules attributes that do not have a large impact on the classification are identified and removed from the input to the clusterer and the process is repeated. Although it is not discussed further in this paper, the process is also repeated within each cluster, creating sub-clusters of each of the major flow types.

3.1 The EM Algorithm for Probabilistic Clustering

The goal of clustering is to divide flows (instances in the generic terminology of machine learning) into natural groups. The instances contained in a cluster are considered to be similar to one another according to some metric based on the underlying domain from which the instances are drawn.

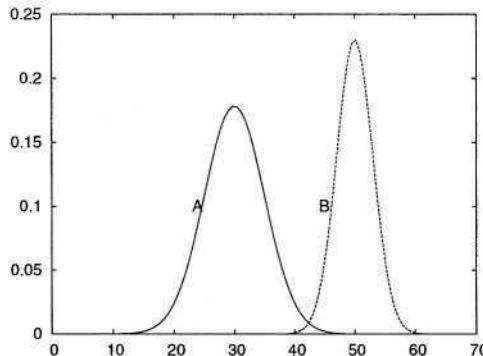


Fig. 2. A two cluster mixture model.

The results of clustering and the algorithms that generate clusters, can typically be described as either “hard” or “soft”. Hard clusters (such as those generated by the simple k -means method [5]) have the property that a given data point belongs to exactly one of several mutually exclusive groups. Soft clustering algorithms, on the other hand, assign a given data point to more than one group. Furthermore, a probabilistic clustering method (such as the EM algorithm [6]) assigns a data point to each group with a certain probability. Such statistical approaches make sense in practical situations where no amount of training data is sufficient to make a completely firm decision about cluster memberships.

Methods such as EM [6] have a statistical basis in probability density estimation. The goal is to find the most likely set of clusters given the training data and prior expectations. The underlying model is called a finite mixture. A mixture is a set of probability distributions—one for each cluster—that model the attribute values for members of that cluster. Figure 2 shows a simple finite mixture example with two clusters—each modelled by a normal or Gaussian distribution—based on a single numeric attribute. Suppose we took samples from the distribution of cluster A with probability p and the distribution of cluster B with probability $1 - p$. If we made a note of which cluster generated each sample it would be easy to compute the maximum likelihood estimates for the parameters of each normal distribution (the sample mean and variance of the points sampled from A and B respectively) and the mixing probability p . Of course, the whole problem is that we do not know which cluster a particular data point came from, nor the parameters of the cluster distributions.

The EM (Expectation-Maximisation) algorithm can be used to find the maximum likelihood estimate for the parameters of the probability distributions in the mixture model². The basic idea is simple and involves considering unobserved latent variables z_{ij} . The z_{ij} 's take on values 1 or 0 to indicate whether data point i comes from cluster j 's model or not. The EM algorithm starts with an initial guess for the parameters of the models for each cluster and then iterati-

² Actually EM is a very general purpose tool that is applicable to many maximum likelihood estimation settings (not just clustering).

vely applies a two step process in order to converge to the maximum likelihood fit. In the *expectation* step, a soft assignment of each training point to each cluster is performed—i.e. the current estimates of the parameters are used to assign cluster membership values according to the relative density of the training points under each model. In the *maximisation* step, these density values are treated as weights and used in the computation of new weighted estimates for the parameters of each model. These two steps are repeated until the increase in log-likelihood (the sum of the log of the density for each training point) of the data given the current fitted model becomes negligible. The EM algorithm is guaranteed to converge to a local maximum which may or may not be the same as the global maximum. It is normal practice to run EM multiple times with different initial settings for the parameter values, and then choose the final clustering with the largest log-likelihood score.

In practical situations there is likely to be more than just a single attribute to be modelled. One simple extension of the univariate situation described above is to treat attributes as independent within clusters. In this case the log-densities for each attribute are summed to form the joint log-density for each data point. This is the approach taken by our implementation³ of EM. In the case of a nominal attribute with v values, a discrete distribution—computed from frequency counts for the v values—is used in place of a normal distribution. Zero frequency problems for nominal attributes can be handled by using the Laplace estimator. Of course it is unlikely that the attribute independence assumption holds in real world data sets. In these cases, where there are known correlations between attributes, various multivariate distributions can be used instead of the simple univariate normal and discrete distributions. Using multivariate distributions increases the number of parameters that have to be estimated, which in turn increases the risk of overfitting the training data.

Another issue to be considered involves choosing the number of clusters to model. If the number of clusters in the data is not known a-priori then hold-out data can be used to evaluate the fit obtained by modelling $1, 2, 3, \dots, k$ clusters [4]. The training data can not be used for this purpose because of overfitting problems (i.e. greater numbers of clusters will fit the training data more closely and will result in better log-likelihood scores). Our implementation of EM has an option to allow the number of clusters to be found automatically via cross-validation. Cross-validation is a method for estimating the generalisation performance (i.e. the performance on data that has not been seen during training) of an algorithm based on resampling. The resulting estimates of performance (in this case the log-likelihood) are often used to select among competing models (in this case the number of clusters) for a particular problem.

4 Cluster Visualisation

The result of clustering is a grouping of flows. By examining the IAT/packet size plots it is possible, given enough thought, to make sense of the clusters produced

³ Included as part of the freely available Weka machine learning workbench (<http://www.cs.waikato.ac.nz/ml/weka>)

but this is a difficult process. To aid the interpretation of the meaning of the clusters, we developed a visualisation based on the Kiviat graph[1]. The six top-level clusters for one of our sample data sets are shown in figure 3. Each graph describes the set of flows in a cluster. The (blue) lines radiating out from the centre point are axes representing each of the attributes we used for clustering. The thick part of the axes represents one standard deviation above and below the mean, the medium thickness lines represent two standard deviations, and the thin lines represent three standard deviations. Note that, in some cases, the axis extends beyond the graph plane and has been truncated. The mean point for each attribute is connected to form the (red) shape in the centre of the graph. Different shapes are characteristic of different traffic profiles. The standard deviation as a percentage of the mean is shown on each axis. This figure gives an indication of how important this attribute is in forming this cluster. If the percentage is high, then this attribute is likely to be a strong classifier.

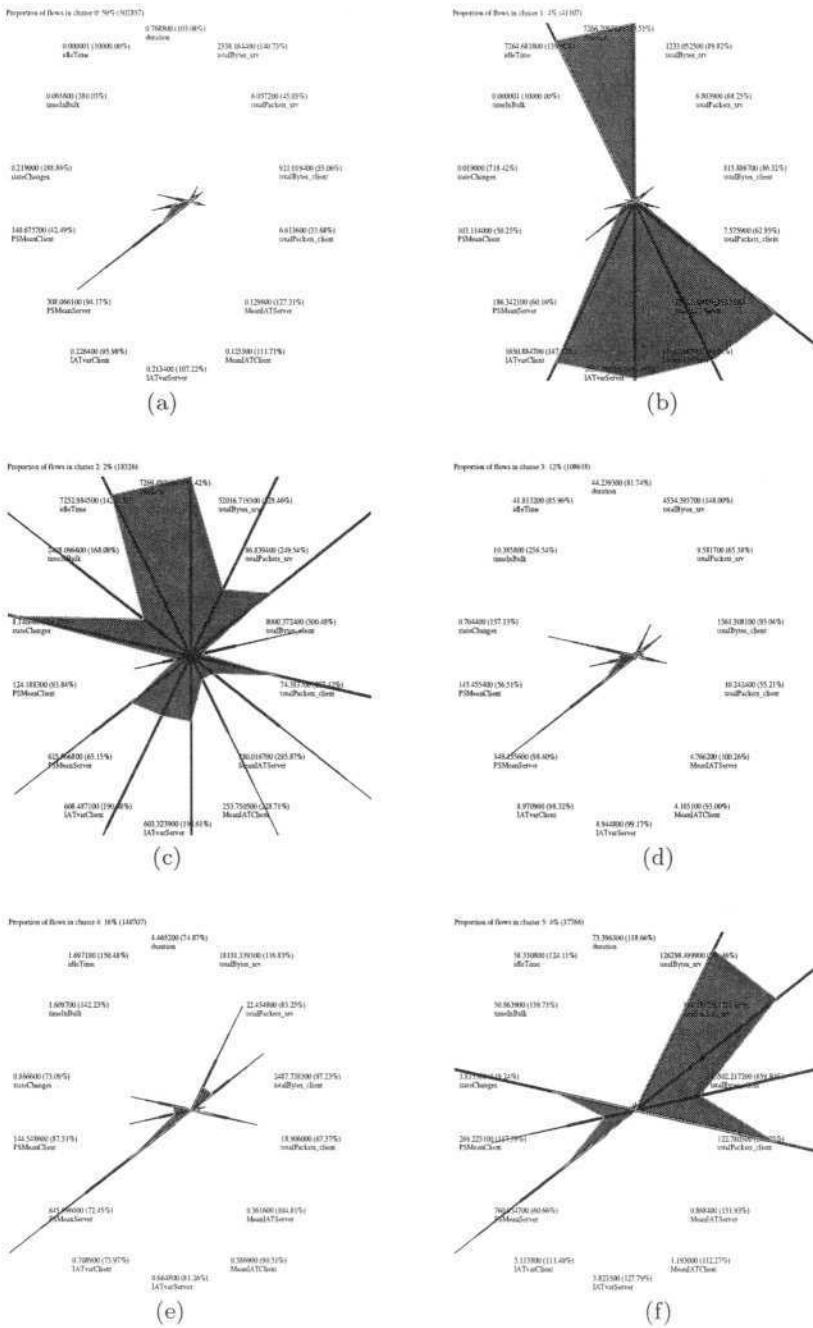
The cluster shown in figure 3(a) contains 59% of the flows in this sample. In this cluster the mean packet size from the server is about 300 bytes and has a large standard deviation. The total number of packets is small (especially remembering the 7 packet overhead of a normal TCP connection). Clients send about 900 bytes of data and servers send an average of about 2300. The duration is short ($\approx 1\text{s}$) and the flow normally stays in transaction mode. This cluster is mostly typical web traffic, fetching small and medium sized objects, for example HTML pages, icons and other small images.

There are two other similar clusters, clusters 3 and 4, shown in Fig 3(d) and (e) respectively. These clusters represent a further 20% of flows. Cluster 4 represents larger objects (with a mean server bytes of about 18000 bytes. In addition to HTTP, quite a lot of SMTP traffic is included in this cluster. The flows in cluster 3 have a significant idle time. These are mostly HTTP 1.1 flows with one or more objects fetched over the same connection. The connection is held open, in the ideal state, for a time after an object is transferred to give the client time to request another object.

Cluster five (Fig. 3(f)) contains classic bulk transfer flows. They are short to medium term (a mean of $1m\ 13s$) and transfer a lot of data from the server to the client. Clusters one and two (Fig. 3(b) and (c)) are long duration flows with a lot of idle time. Flows in cluster two have many small packets transferred in both directions. These are transaction based with multiple transactions in the flow, separated by significant delays. IMAP and NTP are examples. Cluster one has only a few packets. This is predominantly TCP DNS traffic. We suspect this cluster includes applications where the connection is not correctly terminated.

5 Validation

We are still actively developing the methodology. As part of this process we have undertaken four types of validation. We looked at the stability of the clusters within different segments of a single trace, comparing the clusters produced from the whole trace with those produced from half of the trace. Secondly, we compared two traces from different, but similar, locations (the University of Auckland and The University of Waikato). While there were differences in the

**Fig. 3.** Clusters 1 (a), 2 (b) and 4 (c)

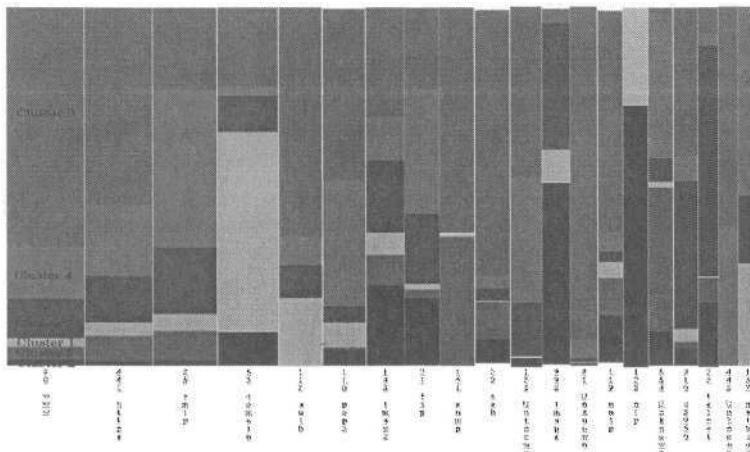


Fig. 4. Ports Across Clusters

statistics of the attributes, both these exercises yielded the same basic set of clusters, as shown by their Kiviat graphs.

Next we examined the distribution of ports across the clusters. Because port numbers are indicative of the application in use we expected ports to be focused on particular clusters. The graphic in Fig 4 shows this distribution for the 20 most common ports. Each stack of bars represents a port with the most common ports on the left and the least common on the right. The width of the bar represents the number of flows with that port type, on a logarithmic scale. Each band represents a cluster, with the largest cluster (cluster 0) at the top, and the smallest (cluster 2) at the bottom.

The distribution of ports across clusters is less differentiated than we expected. There are several reasons for this. First it should be noted that, the log scale of figure 4 (which is necessary to allow the visualisation to show more than just the two or three dominant port types) creates a false impression of the distribution of ports. The second, and more important reason is one we alluded to in the introduction, we just under estimated its significance. HTTP is the predominant traffic type in these traces. HTTP has a wide range of uses and consequently there is a significant amount of HTTP traffic in all clusters.

Finally, we examined whether the algorithm was assigning flows of particular port types to clusters differently than a random assignment would. This analysis, which we can not present here for space reasons, indicated that for most ports there was good discrimination between clusters but for a few, there was not. IMAP is one example where the discrimination was poor.

It seems that, the clustering is generally doing a good job of grouping flows together by their traffic type (bulk transfer, small transactions, multiple transactions etc.) but that individual applications behave more differently across different connections than we had expected. Even given these reasons, the clustering does not currently meet our needs and we are continuing to develop the approach, especially through the derivation of new attributes that we believe

will further discriminate between applications. The existence of idle time at the end of a connection is one example.

6 Conclusion

The initial results of the methodology appear promising. The clusters are sensible and the clustering and classification algorithms indicate that a good fit has been obtained to the data. Initial analysis indicates that the clusters are stable over a range of different data with the same overall characteristics. The existing clusters provide an alternative way to disaggregate a packet header stream and we expect it to prove useful in traffic analysis that focuses on a particular traffic type. For example, simulation of TCP optimisations for high performance bulk transfer. However, further work is required to fully meet our initial goal of clustering traffic into groups that a network manager would recognise as related to the particular application types on their network.

References

1. Kolence, K., Kiviat, P.: Software Unit Profiles and Kiviat Figures. ACM SIGMETRICS Performance Evaluation Review, Vol. 2, No. 3 September (1973) 2–12
2. Witten, I., and Frank, E., Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann (2000)
3. Hastie, T., Tibshirani, R., and Friedman, J., The Elements of Statistical Learning: Data Mining, Inference, and Prediction Springer-Verlag (2001)
4. Smyth, P., Clustering Using Monte Carlo Cross-Validation Proceedings of the Second International Conference on Knowledge Discovery and Data Mining AAAI Press (1996) 126–133
5. Hartigan, J., Clustering Algorithms John Wiley (1975)
6. Dempster, A., Laird, N., and Rubin, D., Maximum Likelihood from Incomplete Data Via the EM Algorithm Journal of the Royal Statistical Society Series B, Vol. 30, No. 1 (1977) 1–38
7. <http://www.nlarr.net/>
8. <http://www.wand.net.nz/>
9. Claffy, K., Braun, H.-W. and Polyzos, G. Internet traffic flow profiling Applied Network Research, San Diego Supercomputer Center (1994) Available at: <http://www.caida.org/outreach/papers/>
10. Mochalski, K., Micheel, J. and Donnelly, S. Packet Delay and Loss at the Auckland Internet Access Path Proceedings of the PAM2002 Passive and Active Measurement Conference, Fort Collins, Colorado USA, (2002)

Using Data Stream Management Systems for Traffic Analysis – A Case Study –

Thomas Plagemann^{1,2}, Vera Goebel^{1,2}, Andrea Bergamini¹,
Giacomo Tolu¹, Guillaume Urvoy-Keller¹, and Ernst W. Biersack¹

¹Institut Eurecom, Corporate Communications, 2229 Route des Crêtes
BP 193 F-06904 Sophia Antipolis Cedex, France
{bergamin, tolu, urvoy, erbi}@eurecom.fr

²University of Oslo, Department of Informatics,
Postbox 1080 Blindern, 0316 Oslo, Norway
{plageman, goebel}@ifi.uio.no

Abstract. Many traffic analysis tasks are solved with tools that are developed in an ad-hoc, incremental, and cumbersome way instead of seeking systematic solutions that are easy to reuse and understand. The huge amount of data that has to be managed and analyzed together with the fact that many different analysis tasks are performed over a small set of different network trace formats, motivates us to study whether *Data Stream Management Systems* (DSMSs) might be useful to develop traffic analysis tools. We have performed an experimental study to analyze the advantages and limitations of using DSMS in practice. We study how simple and complex analysis tasks can be solved with TelegraphCQ, a public domain DSMS, and present a preliminary performance analysis.

1 Introduction and Motivation

The number of tools for analyzing data traffic in the Internet is continuously increasing, because there is an increasing need in many different application domains. For example, network operators and service providers need to monitor data traffic to analyze the provided service level, to identify bottlenecks, and initiate appropriate counter measures, if possible. This is especially important in the Internet, because the amount of data traffic continuously increases and the behavior and requirements of end-users are changing over time, like accepted response time from web servers. Another application domain is the development and improvement of new protocols and applications, like overlay networks and peer-to-peer (P2P) file sharing applications. The complexity of these protocols and applications, as well as the complexity of the environment they are used in, often impose that a meaningful analysis can only be done during their operation in the Internet. The typical coarse grain architecture of these tools consist of two components, first, a packet capturing or flow statistic component like TCPdump or NetFlow, and second, an analysis component to examine the resulting traces and draw certain conclusions.

Performing traffic analysis to gain new knowledge is normally an iterative process. Traffic analysis tools are used to get a better understanding of network dynamics, protocol behavior, etc. Based on these new insights and influenced by changes in the Internet, e.g., traffic mix and end-user behavior, new analysis goals are defined for the

next iteration step. For example, in our recent BitTorrent work, we measured the average throughput of *leechers* (i.e., clients that have not completed the download) by analyzing trace files and identified their origin country [4]. We saw that most leechers are either from the US, Canada, Netherlands, or Australia. Therefore, we analyzed in the next step the average throughput for these countries.

It is a common practice for this type of research to either change or extend existing traffic analysis tools, if it can be avoided to develop a new one. Since many tools are implemented as PERL scripts this often means to study a PERL script and change it. This is not an easy task, even for the author of the script itself if it is not well documented, because the variables in PERL scripts do not have meaningful names. In other words, many problems are solved in an ad-hoc, incremental, and cumbersome way instead of seeking systematic solutions that are easy to reuse and easy to understand. Obviously, the more iterations a traffic analysis study comprises the bigger are the advantages of easily reusable tools.

Another problem with today's tools is the huge amount of data that is generated. Trace files should be archived in order to use them at a later point in time for further studies and evaluations. However, the tools themselves do generally not provide any support for managing these large amounts of data. Therefore, trace files are typically archived as plain files in the file system. Depending on the amount of trace files and the discipline of the researcher to annotate trace files, the retrieval of a particular trace file that is a couple of months old can represent a non-trivial problem.

The huge amount of data that has to be managed and analyzed together with the fact that many different analysis tasks are performed over a small set of different network trace formats, motivates us to study whether Database Management Systems (DBMSs) might be a useful platform for developing tools for traffic analysis. DBMSs are designed to separate data and their management from application semantics to facilitate independent application development. They are designed to handle very large amounts of data, and to efficiently perform searches, comparisons, ordering, summarize data, etc. Furthermore, Internet traffic consists of well-structured data, due to the standardized packet structures, and can therefore easily handled with DBMSs.

Traditional DBMS need to store the data before they can handle it. However, many application domains would benefit from on-line analysis and immediate indication of results. Therefore, we focus our attention on a new emerging DBMS technology called *Data Stream Management Systems* (DSMSs). In contrast to traditional DBMSs, DSMSs can execute continuous queries over continuous data streams that enter and leave the system in real-time, i.e., data is only stored in main memory for processing. Such data streams could be sensor readings, stock market data, or network traffic [1].

Since DSMSs are a promising technology for traffic analysis, we have performed an experimental study to analyze the advantages and limitations of using public domain DSMSs in practice and report our experiences in this paper. In the following section, we discuss in more detail our expectations and requirements on using DSMSs for traffic analysis and describe the approach of our study. We give in Section 3 a brief introduction to DSMSs and the particular DSMS TelegraphCQ [5], [9] we are using. In Section 4, our experiments and their results are presented. In Section 5, we conclude with a general discussion of advantages and limitations and an outlook to our ongoing and future research in this area.

2 Expectations, Requirements, and Approach

Our main expectation with respect to the use of DSMSs for traffic analysis is that DSMSs might be a generic platform that simplify the development of analysis components, are easily reusable, are self-documenting, allow on-line and off-line analysis with the same tool, and support management and archival of data. Typical tasks are to analyze:

- the load of a system, e.g., how often are certain ports, like FTP, or HTTP, of a server contacted; which share of bandwidth is used by different applications; which departments use how much bandwidth on the university backbone,
- characteristics of flows, like distribution of life time and size of flows; relation between number of lost packets and life time of flows; what are the reasons for throughput limitations, or
- characteristics of sessions, like how long do clients interact with a web server; which response time do clients accept from servers; how long are P2P clients on-line after they have successfully downloaded a file.

The above examples indicate an important functional requirement, i.e., the system should be capable to handle all protocol layers including the application layer. An important non-functional requirement is introduced by the need for real-time analysis. A DSMS should be able to handle data with a throughput that is proportional to the network load. For example, to analyze IP and TCP headers on a fully utilized gigabit/s network would require to handle more than 42 megabit/s of data (assuming a fixed packet size of 1500 bytes and a header size of 64 bytes). However, a more realistic assumption is that it is only possible to reduce the data stream by a factor of 4:1 to 9:1 relative to the current network load [6].

DSMSs require like any other DBMS a schema describing the type and structure of the data to be handled. Therefore, we expect that reuse and changes of DSMS applications for traffic analysis will be easier than changing PERL scripts. A side effect of this property might also be that applications can be easier exchanged between researchers and results from others can be easier reproduced.

The major publications in the research area of DSMSs raise the expectations that the above requirements can be met by DSMSs. Even quite high performance numbers are reported for a proprietary DSMS from AT&T, called GigaScope [2]. It has been successfully used for network monitoring and is able to handle at peak periods 1.2 million packets per seconds on a dual 2.4 Ghz CPU server.

We are interested in whether public domain DSMS technology is already mature enough to be useful in practice for traffic analysis. Therefore, we used a public-domain DSMS, called TelegraphCQ [5], and studied how simple and complex analysis tasks can be solved with it.¹ Since it is not the goal of this work to develop new solutions for complex tasks, we investigate how the functionality of an existing tool can be re-implemented with TelegraphCQ. We selected the tool T-RAT [10], because we are using and improving it in our ongoing research work [7].

¹ At the beginning of this project (August 2003), we identified TelegraphCQ as the only available public domain DSMS.

3 Data Stream Management Systems

The fundamental difference between a classical DBMS and a DSMS is the *data stream model*. Instead of processing a query over a persistent set of data that is stored in advance on disk, queries are performed in DSMSs over a data stream. In a data stream, data elements arrive on-line and stay only for a limited time period in memory. Consequently, the DSMS has to handle the data elements before the buffer is overwritten by new incoming data elements. The order in which the data elements arrive cannot be controlled by the system. Once a data element has been processed it cannot be retrieved again without storing it explicitly. The size of data streams is potentially unbounded and can be thought of as an open-ended relation. In DSMSs, *continuous queries* evaluate continuously the arriving data elements. Standard operator types that are supported by most existing DSMSs are filtering, mapping, aggregates, and joins. Since continuous streams may not end, intermediate results of continuous queries are often generated over a predefined *window* and then either stored, updated, or used to generate a new data stream of intermediate results [1]. Window techniques are especially important for aggregation and join queries. Examples for DSMSs include STREAM [1], GigaScope [2], and TelegraphCQ [5]. The interested reader can find an extensive overview on DSMSs in [1] and [3].

TelegraphCQ is characterized by its developers as “a system for continuous dataflow processing” that “aims at handling large streams of continuous queries over high-volume highly variable data streams” [5]. TelegraphCQ is based on the code of the relational DBMS PostgreSQL and required major extensions to it to support continuous queries over data streams, like adaptive query processing operators, shared continuous queries, and data ingress operations. We focus in this paper on the extensions visible for user, i.e., data model and query language extensions. The format of a data stream is defined as any other PostgreSQL table in PostgreSQL’s Data Definition Language (DDL) and created with the `CREATE STREAM` command before a continuous query can be launched and processed.

Figure 1 illustrates the data flow during processing of a continuous query and shows also the main components of TelegraphCQ. The Wrapper ClearingHouse (WCH) is responsible for the acquisition of data from external sources, like the packet capturing tool TCPdump. The WCH loads the user-defined wrapper function for the source. The wrapper is reformatting the output of the source into the PostgreSQL data types according to the DDL definition of the particular stream. There is always a one-to-one relation between source and wrapper, and between wrapper and stream. The WCH can manage multiple wrappers respectively streams and fetches the stream data via TCP connections from the wrapper(s). The newly created tuples of each stream are placed by the WCH into the shared memory infrastructure to make them available to the rest of the system. The query processing is performed in the BackEnd and the results are placed in corresponding queues in the shared memory infrastructure. Finally, the FrontEnd continually dequeues the results and sends them to the client. In order to reuse results, it is necessary to start TelegraphCQ in such a way that the standard output is written to a file.

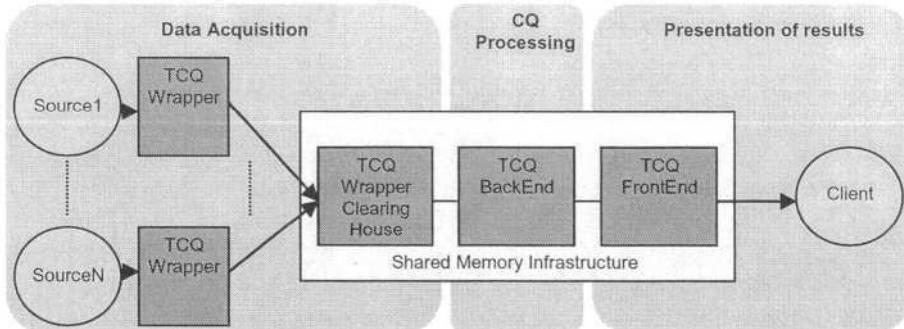


Fig. 1. Data flow in TelegraphCQ (TCQ) during continuous queries (CQ) processing

All interactions between client and TelegraphCQ are performed through the FrontEnd, including processing of DDL statements and accepting queries from the client. Queries over tables are only directly processed in the FrontEnd. Continuous queries, i.e., queries over streams (and stored tables) are pre-planned by the FrontEnd and passed to the BackEnd.

Continuous queries over data streams are written in SQL with the SELECT statement, but only a subset of the full SQL syntax is supported. The modified SELECT statement has the following form:

```

SELECT <select_list>
FROM <relation_and_pstream_list>
WHERE <predicate>
GROUP BY <group_by_expressions>
WINDOW stream[interval], ...
ORDER BY <order_by_expressions>;

```

Continuous queries may include a WINDOW clause to specify the window size for the stream operations. A window is defined in terms of a time interval. Each arriving data element is assigned by the wrapper a special time attribute (called TIMESTAMP) and the window borders are continuously updated with respect to the timestamp of the most recently arriving data element, i.e., evaluation of continuous queries is based on *sliding windows* [5]. All other clauses in the SELECT statement behave like the PostgreSQL select statement with the following additional restrictions in the TelegraphCQ 0.2 alpha release [9]: windows can only be defined over streams (not for PostgreSQL tables); WHERE clause qualifications that join two streams may only involve attributes, not attribute expressions or functions; WHERE clause qualifications that filter tuples must be of the form attribute operand constant; WHERE clause may only contain AND (not OR); subqueries are not allowed; GROUP BY and ORDER BY clauses are only allowed in window queries.

4 Experiments and Experiences with TelegraphCQ

In this section, we describe some simple, but typical traffic analysis tasks, the design of a complex analysis tasks, and give some performance bounds for TelegraphCQ.

4.1 Solving Simple Traffic Analysis Tasks

For each of the tasks that are discussed in this subsection, we explain how it can be solved online with a continuous query in TelegraphCQ, or why it cannot be solved with a continuous query. For all examples, we assume an input stream that has been defined with the following DDL statement:

```
CREATE STREAM p6trace.tcp (ip_src cidr, ip_dst cidr, hlen
bigint, tos int, length bigint, id bigint, frag_off bigint,
ttl bigint, prot int, ip_hcrcsum bigint, port_src bigint,
port_dst bigint, sgn bigint, ack bigint, tcp_hlen bigint,
flags varchar(10), window bigint, tcp_csum bigint, tcqtime
timestamp TIMESTAMPCOLUMN) type ARCHIVED;
```

Each tuple in the stream p6trace. tcp comprises IP and TCP header fields and an attribute for timestamp values that are assigned by the wrapper. The attribute type cidr defines an attribute to store an IPv4 or IPv6 address in dotted notation.

Task 1. *How many packets have been sent during the last five minutes to certain ports?*

This is an example for a join operation between a stream and a table. The port numbers of interest are stored in the table services. The TCP destination port field port_dst in all tuples in the stream p6trace. tcp is compared with those in the table.

```
CREATE TABLE services (port bigint, counter bigint);
SELECT services.port, count(*)
FROM p6trace.tcp, services
WHERE p6trace.tcp.port_dst=services.port
GROUP BY services.port
WINDOW p6trace.tcp ['5 min'] ;
```

This is also an example for the use of the sliding window. Each new arriving tuple is advancing the sliding window and the intermediate result of the continuous query for this window is passed to the client. This is a powerful feature for on-line monitoring, since it makes sure that the client receives always the most recent statistic. The drawback of sliding windows is the inherent redundancy in the intermediate results. Assuming that the one minute window covers always n tuples, each tuple will contribute n times to an intermediate result. This increases the amount of output data and makes it impossible to perform absolute statistics over a stream, like it is necessary for Task 3.

Task 2. *How many bytes have been exchanged on each connection during the last minute?*

```
SELECT ip_src, port_src, ip_dst, port_dst, sum(length-ip_len-
tcp_hlen)
FROM p6trace.tcp
GROUP BY ip_src, port_src, ip_dst, port_dst
WINDOW pStrace.tcp ['1 min'];
```

From the stream `p6trace.tcp`, all tuples, i.e., packet headers, that have arrived during the last minute are grouped according to their source and destination IP address and their port numbers. The `SELECT` statement specifies that all address information for each group together with the sum of the payload length of all packets in this group are returned. The identification of connections in this continuous query is based on a simple heuristic: during a one minute window all packets with the same sender and receiver IP addresses and port numbers belong to the same connection.

Task 3. *How many bytes are exchanged over the different connections during each week?*

There are two basic deficiencies in the current TelegraphCQ prototype that make it impossible to solve this task with a continuous query. The first deficiency is that a `GROUP BY` clause can only be used together with a `WINDOW` clause. It is obvious that the window size must be much smaller than one week, because all incoming data is kept in main memory until the entire window has been computed. The sliding window imposes that the payload of each packet would contribute several times to intermediate results when the inter arrival time of packets is smaller than the window size. In order to calculate a correct final result that summarizes all intermediate results, it is necessary to remove the redundant information from all the intermediate results. In other words, to calculate absolute statistics with continuous queries, non-overlapping windows (also called *jumping* or *tumbling windows*) are needed, but this type of windows is not supported in TelegraphCQ.

The second deficiency relates to connection identification. The simple heuristic we used for Task 2 to identify connections cannot be used, because the same quadruple of sender and receiver IP addresses and port numbers can identify many different connections during a week. In other words, a simple `GROUP BY` clause over a set of attributes (in this example the four address fields) is not sufficient. Additional rules are needed to distinguish different connections with the same attribute values. For example, we could define a connection as released if for T time units no packets have been sent, i.e., packets with the same address quadruple that are sent after such a break belong to a new connection. However, the important aspect of this example is not the particular approach of how connections are identified. Instead, it illustrates a problem that is common to many traffic analysis tasks that cannot be solved by a continuous query in TelegraphCQ. The basic problem is that associations, like flows, sessions etc., must be recognized and each packet must be related to a single association such that statistics for each association can be calculated. For the identification of associations, the address fields in packets are typically used together with certain rules that are depending on the protocol. Before a packet stream is analyzed it is not known which associations with their particular attribute values will be seen. Instead, this information has to be extracted from the data stream. Afterwards, it is possible to assign packets to their associations, based on the attribute values. The traditional solution is to maintain a data structure to store data about associations. Each packet in the data stream can contribute in two ways to the data that is stored in the data structure. First, it is used to analyze whether it belongs to a new association that has to be inserted in the data structure and second, it is used to update the statistic for the particular association it belongs to. With the SQL `SELECT` statement it is not possible to

maintain a data structure to store intermediate results. Therefore, the only on-line solution is to analyze the data stream two times, first to identify associations and second to calculate association statistics. Since DSMSs do only allow a single pass over the data stream, it is only possible to perform such a task in a continuous query with subqueries. Since TelegraphCQ does not support subqueries, this type of task cannot be solved on-line in TelegraphCQ.

Task 4. *Which department has used how much bandwidth on the university backbone in the last five minutes?*

To solve this task, a wrapper has to capture all packets from the backbone. A join operation has to be performed between a predefined stored table that contains the IP addresses that are used in the different departments and the stream p6trace.tcp from the wrapper. The best solution would be to define for each department the existing IP address range and check with the PostgreSQL operator “>>” which address range contains the IP address of the packet in the data stream. The corresponding DDL statement to create such a table is as follows:

```
CREATE TABLE departments (name varchar(30), prefix cidr,
traffic bigint);
SELECT departments.name, sum(length-hlen-tcp_hlen)
FROM p6trace.tcp, departments
WHERE departments.prefix >> p6trace.tcp.ip_src
GROUP BY departments.name
WINDOW p6trace.tcp ['5 min'];
```

Unfortunately, the current TelegraphCQ prototype produces incorrect results if a “>>” operator is used in a join. However, it works correctly if a “=” operator is used. To solve the task, it is therefore necessary to store all IP numbers that are used by the departments in a stored table:

```
CREATE TABLE departments (name varchar(30), ip_addr cidr,
traffic bigint);

SELECT departments.name, sum(length-hlen-tcp_hlen)
FROM p6trace.tcp, departments
WHERE departments.ip_addr = p6trace.tcp.ip_src
GROUP BY departments.name
WINDOW p6trace.tcp ['5 min'];
```

The major disadvantage of this solution is that all IP addresses must be enumerated and can lead to a very large table.

4.2 TCP Rate Analysis with TelegraphCQ and PostgreSQL

T-RAT [10] aims to identify the reasons for rate limitations, e.g., congestion, receiver buffer constraints, of TCP flows. For simplicity reasons, we focus our description just on the first important steps of the T-RAT algorithm (Fig. 2.a) and how they could be performed in TelegraphCQ and PostgreSQL (Fig. 2.b). The initial step is to identify connections in a TCPdump trace file based on matching IP addresses and port numbers of source and destination. This is not possible on-line. Therefore, in TelegraphCQ a continuous query is used to put the relevant data from a TCPdump wrapper

into table T1. The remaining process has to be done off-line in PostgreSQL. First, query Q1 is used to generate table T2 with all connections. Afterwards, query Q2 relates in table T3 all packets to their connections. In order to partition the set of packets in each connection in flights (based on 27 RTT candidates), query Q3 has to join table T3 and the table with the RTT candidates. The join operation in Q3 cannot be solely expressed in SQL and requires an external C function. The next query Q4 requires also an external C function to classify the flights into the different protocols states “slow start”, “congestion avoidance”, and “unknown”, and to find the best fit which indicates the best RTT for the particular connection.

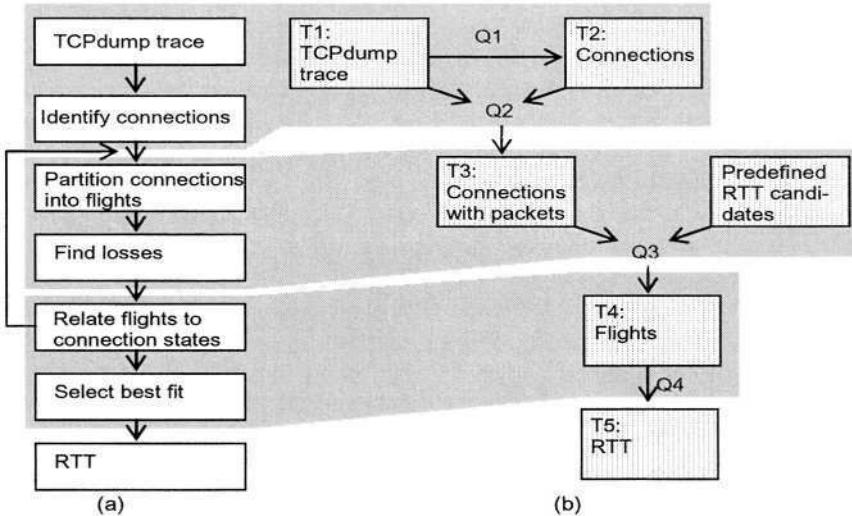


Fig. 2. Structure of the original T-RAT algorithm (a) and its design in Telegraph CQ (b)

The main insights from this design exercise are the following: With TelegraphCQ it is not possible to perform this task in a continuous query, because to identify connections and flights it would be necessary to handle dynamic tables in continuous queries with subqueries. Both are not supported in the current release. The main functionality of T-RAT has to be performed off-line with PostgreSQL. T-RAT is using complex heuristics which cannot be expressed in SQL. However, the extensibility of TelegraphCQ, respectively PostgreSQL, allows to increase the expressiveness of queries with external functions. Thus, T-RAT and other complex analysis can be performed off-line with TelegraphCQ, even if the main functionality is then hidden in external functions instead of SQL statements.

In general, it should be noted that DSMSs are not appropriate for all problems, and that DSMSs require a new way of designing tools for traffic analysis. A developer should especially pay attention to the question which functionality should be performed in a continuous query and which should be performed off-line.

4.3 Performance Evaluation

In order to get a first idea about the performance of TelegraphCQ we performed some experiments in a subnetwork at Institute Eurecom which is based on a 100 Mbit/s Ethernet. The load in this subnetwork is close to zero and its impact on our results can be ignored. We used a simple workload generator to generate a well-defined load for the wrapper and TelegraphCQ. A client streams a fixed amount of data with fixed rates to a server via a TCP connection. This enables us to use the same wrapper and stream definition, i.e., `p6trace.tcp`, as defined in Section 4.1. The size of all TCP packets is equal to the maximum segment size. Due to practical reasons, the wrapper, TelegraphCQ, and the server are running on the same machine, a Pentium 4 machine with a 2Ghz CPU, 524 MB RAM, and a 100 Mb/s Ethernet card.

The basic idea of our performance experiments is to increase the network load until TelegraphCQ is no longer able to handle all data. This load indicates an upper performance bound for TelegraphCQ. We use two mechanisms to recognize whether TelegraphCQ could handle all data or not. First, we log the data that is forwarded from the wrapper to TelegraphCQ, and second, TelegraphCQ itself summarizes how much data it handled. We compare these results with the (amount of) data the workload generator streamed on the network.

Figure 3 shows how the query type and the number of attributes that are handled in the query impact the performance. With a projection, i.e., a simple query that selects a certain number of attributes, TelegraphCQ can handle in maximum 3.6 megabyte per second (MB/s) network traffic for one attribute and 2.7 MB/s for 16 attributes. The upper performance bound for an aggregation is 3.4 MB/s for one attribute (Section 4.1, Task 1) and 2.7 MB/s for four attributes.² The most surprising results we got for queries that perform a JOIN operation over a stream and a table (Section 4.1 Task 4). A join is normally the most costly operation in a DBMS. We assumed that the performance of a join is not better than the performance of the projection if both generate the same amount of output data. However, in our measurements TelegraphCQ can keep up with 5.8 MB/s of network data for handling one, two, and four attributes in a join when using a small table with ten entries. Further investigations showed that the number of matches between table and stream does impact the performance. The more entries in the table match the attribute value of a stream tuple, the lower the performance. The results in Figure 3 are based on a unique match. The size of the table and the position of the matching entry in the table influence the performance. However, even with a table of 100000 entries in which the matching entry is the last one, TelegraphCQ can still handle more than 2 MB/s of network data without loss. In order to verify these preliminary results, more in depth investigations have to be performed. Our results so far can only document that TelegraphCQ is fast enough to perform meaningful network analysis tasks on a commodity PC with a standard Linux configuration (OS release 2.4.18-3) without loosing data up to network loads of 2,5 MB/s.

² For aggregation and join we increased the number of attributes only to four, because we cannot see any meaningful application that would handle more aggregation or join attributes in these queries.

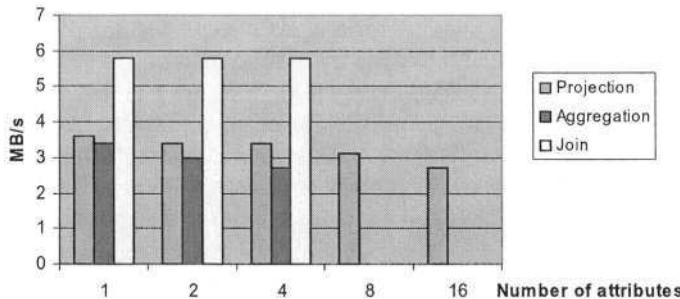


Fig. 3. Upper performance bounds for continuous queries

5 Discussion and Conclusions

Comparing our expectations and requirements with the experiences we gained in this study, we can state that the recent TelegraphCQ prototype is quite useful for many on-line monitoring tasks. The sliding window concept assures that the client gets always the most recent statistics and the system can keep up with relatively high link speeds by running only on commodity hardware. We have shown some preliminary performance measurements and that the performance of TelegraphCQ is influenced by the number of attributes in the output. It should also be noted that the query language can be extended with inbuild C-functions. This feature enables a developer to implement and use in TelegraphCQ complex algorithms that cannot be expressed with SQL. As it is natural for an early prototype, not all features are fully implemented yet. For example, joins between a stream and a table can only compare elements with the “=” operator (i.e., only equi-joins are supported in the current release). Therefore, it is not possible to match prefixes of IP addresses, even if it would be very helpful to identify the origin IP domain of packets. Instead, all possible origin IP addresses we want to compare with have to be stored in the table in advance.

We have identified three restrictions in the design of TelegraphCQ that makes it not suitable as a general tool for traffic analysis in its current stage:

- *Subqueries are not supported:* all tasks that require to identify associations by inspecting the data stream twice, i.e., a simple GROUP BY statement over certain attributes is not sufficient, since it cannot be solved on-line.
- *Jumping or tumbling windows are not supported:* a sliding window introduces redundancy, because each tuple contributes multiple times to a sliding window result. To calculate statistics that are correct for time intervals that are longer than a single window, this redundancy has to be removed. However, this can only be done if it is known how and to which intermediate result each packet contributes. However, this knowledge cannot be assumed to be present at the client or an off-line application that is using the set of intermediate results.
- *On-line and off-line handling is not integrated:* TelegraphCQ is designed to forward all results to the user (client). There is no direct way to insert results

from a continuous query into a PostgreSQL database. The workaround is to start the system such that its standard output is placed in a file. This file in turn could be later imported into a PostgreSQL database.

These aspects are supported by those DSMS that are targeted for network monitoring, i.e., Tribeca [8] and GigaScope [2]. However, both systems are not available as public domain DSMS. Therefore, we will in future studies focus on STREAM [1] which is promised to be public domain within the next months and supports subqueries. We will also closely follow future releases of TelegraphCQ.

References

1. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and Issues in Data Stream Systems, ACM Symposium on Principles of Database Systems PODS 2002, Dallas, Texas, USA, May 2000
2. Cranor, C., Johnson, T., Spatchcheck, O., Shkpenyuk, V.: Gigascope: A Stream Database for Network Applications, ACM SIGMOD 2003, San Diego, California, USA, June 2003
3. Golab, L., Özsu, M. T.: Issues in Data Stream Management, ACM SIGMOD Record, Vol. 32, No. 2, June 2003, pp. 5-14
4. Izal, M., Biersack, E. W., Felber, P. A., Urvoy-Keller, G., Al Hamra, A., Garces-Erice, L.: Dissecting BitTorrent: Five Months in a Torrent's Lifetime, PAM2004, Antibes Juan-les-Pins, France April 2004
5. Krishnamurthy, S., Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., Madden, S., Reiss, F., Shah, M. A.: TelegraphCQ: An Architectural Status report, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, March 2003
6. Micheel, J., Braun, H.-W., Graham, I.: Storage and Bandwidth Requirements for Passive Internet Header Traces, Workshop on Network-Related Data Management, in conjunction with ACM SIGMOD/PODS 2001, Santa Barbara, California, USA, May 2001
7. Schleippmann, C.: Design and Implementation of a TCP Rate Analysis Tool. Master Thesis, TU München/Institut Eurecom, August 2003
8. Sullivan, M., Heybey, A.: Tribeca: A System for Managing Large Databases of Network Traffic, Proc. USENIX Annual Technical Conference, New Orleans, USA, June 1998
9. TelegraphCQ: <http://telegraph.cs.berkeley.edu/>, 2003
10. Zhang, Y., Breslau, L., Paxson, V., Shenker, S.: On the Characteristics and Origins of Internet Flow Rates, ACM SIGCOMM'02, Pittsburg, USA, August 2002

Inferring Queue Sizes in Access Networks by Active Measurement

Mark Claypool, Robert Kinicki, Mingzhe Li, James Nichols, and Huahui Wu

CS Department at Worcester Polytechnic Institute
Worcester, MA, 01609, USA {claypool, rek, lmz, jnick, flashine}@cs.wpi.edu

Abstract. Router queues can impact both round-trip times and throughput. Yet little is publicly known about queue provisioning employed by Internet services providers for the routers that control the access links to home computers. This paper proposes QFind, a black-box measurement technique, as a simple method to approximate the size of the access queue at last mile router. We evaluate QFind through simulation, emulation, and measurement. Although precise access queue results are limited by receiver window sizes and other system events, we find there are distinct difference between DSL and cable access queue sizes.

1 Introduction

The current conventional wisdom is that over-provisioning in core network routers has moved Internet performance bottlenecks to network access points [1]. Since typical broadband access link capacities (hundreds of kilobytes per second) are considerably lower than Internet Service Provider (ISP) core router capacities (millions of kilobytes per second), last-mile access links need queues to accommodate traffic bursts. Given the bursty nature of Internet traffic [7] that is partially due to flows with high round-trip times or large congestion windows, it is clear that the provider's choice for access link queue size may have a direct impact on a flow's achievable bitrate. A small queue can keep achieved bitrates significantly below the available capacity, while a large queue can negatively impact a flow's end-to-end delay. Interactive applications, such as IP telephony and some network games, with strict delay bounds in the range of hundreds of milliseconds experience degraded Quality of Service when large access queues become saturated with other, concurrent flows.

Despite the importance of queue size to achievable throughput and added delay, there is little documentation on queue size settings in practice. Guidelines for determining the “best” queue sizes have often been debated on the e2e mailing list,¹ an active forum for network related discussion by researchers and practitioners alike. While general consensus has the access queue size ranging

¹ In particular, see the e2e list archives at: <ftp://ftp.isi.edu/end2end/end2end-interest-1998.mail> and <http://www.postel.org/pipermail/end2end-interest/2003-January-002702.html>.

from one to four times the capacity-delay product of the link, measured round-trip times vary by at least two orders of magnitude (10 ms to 1 second) [6]. Thus, this research consensus provides little help for network practitioners to select the best size for the access queue link. Moreover, a lack of proper queue size information has ramifications for network simulations, the most common form of evaluation in the network research community, where access queue sizes are often chosen with no confidence that these queue choices accurately reflect current practices.

A primary goal of this investigation is to experimentally estimate the queue size of numerous access links, for both cable modem and DSL connections managed by a variety of ISPs. Network researchers should find these results useful in designing simulations that more accurately depict current practices.

2 QFind

Based on related work and pilot studies, the following assumptions are made in this study: each access link has a relatively small queue size - between 10 and 100 packets; the maximum queue length is independent of the access link capacity or other specific link characteristics; and the queue size is constant and independent of the incoming traffic load with no attempt made by the router to increase the queue sizes under heavier loads or when flows with large round-trip times are detected. Below is our proposed *QFind* methodology for inferring the access network queue size from an end-host:

1. Locate an Internet host that is slightly upstream of the access link while still being “close” to the end-host. For the test results discussed in this paper, the DNS name server provided by the ISP is used since DNS servers are typically close in terms of round-trip time and easy to find by inexperienced end-users.
2. Start a ping from the end-host to the close Internet host and let it run for up to a minute. The minimum value returned during this time is the baseline latency typically without any queuing delays since there is no competing traffic causing congestion. This ping process continues to run until the end of the experiment.
3. Download a large file from a remote server to the end-host. For the test results in this paper, a 5 MByte file was used since it typically provided adequate time for TCP to reach congestion avoidance and saturate the access queue downlink capacity.
4. Stop the ping process. Record the minimum and maximum round-trip times as reported by ping and the total time to download the large file. The maximum ping value recorded during the download typically represents the baseline latency plus the access link queuing delay.

The queue size of the access link can be inferred using the data obtained above. Let D_t be the total delay (the maximum delay seen by ping):

$$D_t = D_l + D_q \quad (1)$$

where D_l is the latency (the minimum delay seen by ping) and D_q is the queuing delay. Therefore:

$$D_q = D_t - D_l \quad (2)$$

Given throughput T (measured during the download), the access link queue size in bytes, q_b , can be computed by:

$$q_b = D_q \times T \quad (3)$$

For a packet size s (say 1500 bytes, a typical MTU), the queue size in packets, q_p , becomes:

$$q_p = \frac{(D_t - D_l) \times T}{s} \quad (4)$$

The strength of the QFind methodology lies in its simplicity. Unlike other approaches [1,8,10], QFind does not require custom end-host software, making it easier to convince volunteers to participate in an Internet study. Moreover, the simple methodology makes the results reproducible from user to user and in both simulation and emulation environments.

2.1 Possible Sources of Error

The maximum ping time recorded may be due to congestion on a queue other than the access queue. However, this is unlikely since the typical path from the end-host to the DNS name server is short. Pilot tests [3] suggest any congestion from the home node to the DNS name server typically causes less than 40 ms of added latency. Moreover, by having users repeat steps 2-4 of the QFind methodology multiple times (steps 2-4 take only a couple of minutes), apparent outliers can be discarded. This reduces the possibility of over-reporting queue sizes.

The queue size computed in Equation 4 may underestimate the actual queue size since it may happen that the ping packets always arrive to a nearly empty queue. However, if the file download is long enough, it is unlikely that every ping packet will be so lucky. Results in Section 3 suggest that the 5 MB file is of sufficient length to fill queues over a range of queue sizes.

If there is underutilization on the access link then the access queue will not build up and QFind may under-report the queue size. This can happen if there are sources of congestion at the home node network before ping packets even reach the ISP. Most notably, home users with wireless networks may have contention on the wireless medium between the ping and download packets. Pilot tests [3] suggest that congestion on a wireless network during QFind tests adds at most 30 ms to any recorded ping times. As 30 ms may be significant in computing

an access queue size, we ask QFind volunteers to indicate wireless/wired settings when reporting QFind results.

If the TCP download is limited by the receiver advertised window instead of by the network congestion window, then the queue sizes reported may be the limit imposed by TCP and not be the access link queue. However, recent versions of Microsoft Windows as well as Linux support TCP window scaling (RFC 1323), allowing the receiver advertised window to grow up to 1 Gbyte. Even if window scaling is not used, it is still possible to detect when the receiver advertised window might limit the reported queue. The lack of ping packet losses during the download would suggest that the access queue was not saturated and the queue size could actually be greater than reported.

For actual TCP receiver window settings, Windows 98 has a default of 8192 bytes, Windows 2000 has a default of 17520 bytes, Linux has a default of 65535 bytes, and Windows XP may have a window size of 17520, but it also has a mostly undocumented ability to scale the receiver window size dynamically.

Additionally, some router interfaces may process ping packets differently than other data packets. However, in practice, hundreds of empirical measurements in [2] suggest ping packets usually provide round-trip time measurements that are effectively the same as those obtained by TCP.

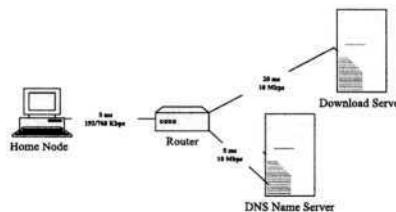


Fig. 1. Topology

3 Experiments

To determine whether the QFind methodology could effectively predict access link queue sizes in real last-mile Internet connections, we evaluated the QFind approach first with simulations using NS² (see Section 3.1) and then emulations using NIST Net³ (see Section 3.2). After reviewing these proof-of-concept results, we enlisted many volunteers from the WPI community to run QFind experiments over a variety of DSL and cable modem configurations from home (see Section 3.3).

² <http://www.isi.edu/nsnam/ns/>

³ <http://snad.ncsl.nist.gov/itg/nistnet/>

3.1 Simulation

QFind was simulated with the configuration depicted in Figure 1 consisting of a home node, an ISP last-mile access router, a TCP download server and a DNS name server. The simulated link latencies used in the emulations were based on prototype QFind measurements.

The delays built into the testbed emulations were 5 ms from home to router, 5 ms from router to DNS, and 20 ms from router to download server. Link capacities were set to reflect typical asymmetric broadband data rates [8], with the router-to-home downstream link capacity set at 768 Kbps, the home-to-router upstream link capacity set at 192 Kbps, and the link capacities in both directions between router and both upstream servers set at 10 Mbps. 1500 byte packets were used to model the typical Ethernet frame size found in home LANs and TCP receiver windows were set to 150 packets.

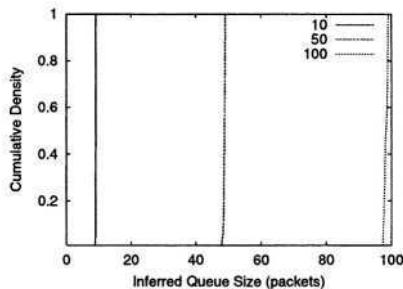


Fig. 2. Cumulative Density Functions of Inferred Queue Sizes for Actual Queue Sizes of 10, 50 and 100 Packets using NS Simulator

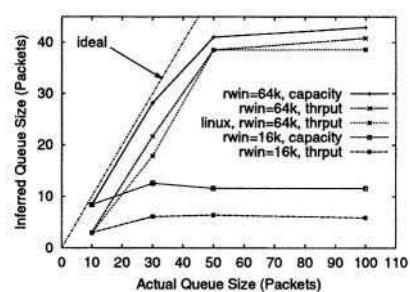


Fig. 3. Median of Inferred Queue Sizes versus Actual Queue Sizes using NIST Net Emulator

Figure 2 displays the cumulative density functions for 100 simulations of the QFind methodology (steps 2 to 4 in Section 2) with downstream access link queues of 10, 50 and 100 packets respectively. QFind predicts the access queue size remarkably well in this simulated environment. Of the 100 runs at each queue size, the *most* the predicted queue size was smaller than the actual queue size was 1 packet for the 10 packet queue, 1.5 packets for the 50 packet queue and 2.5 packets for the 100 packet queue. The median predicted queue size was less than the actual queue size by about 1 packet in all cases.

3.2 Emulation

To further investigate QFind feasibility, we setup a testbed to emulate a last-mile access router in a controlled LAN environment. Two computers were used as home nodes with one computer running Windows 2000 and the other running

Linux in order to test the impact of the operating system type on QFind. The download server ran on Windows Server 2003 while the DNS name server ran on Linux. A NIST Net PC router emulated the ISP's Internet connection with link capacities set to reflect typical broadband asymmetry with the downstream router-to-home link capacity set to 768 Kbps, the upstream home-to-router link set to 192 Kbps, and the router link capacities to and from both servers using 10 Mbps LAN connections. The home-to-server round-trip delay was 20 ms for both the download server and the DNS server since the NIST Net implementation does not allow two host pairs to have different induced delays while sharing a router queue.

Using this testbed, the QFind methodology was emulated (steps 2 to 4 in Section 2) with home nodes running Windows 2000 with a TCP receiver window size of 16 Kbytes, Windows 2000 with a TCP receiver window sizes set to 64 Kbytes, and Linux with a TCP receiver window sizes set to 64 Kbytes. Three QFind emulations were run for each of the queue sizes of 10, 30, 50 and 100 packets, with a packet size of 1500 bytes.

Figure 3 presents the median of the inferred queue sizes. The inferred queue sizes labeled "thrput" are computed using the measured download capacity. The inferred queue sizes labeled "capacity" are computed using the capacity of the link. In those cases where the NIST Net queue size is smaller than the TCP receiver window size, QFind is able to infer the queue size closely, even for different operating systems. The queue sizes computed using link capacity are more accurate than those computed using download throughput. However, while the link capacity was, of course, known by us for our testbed, it is not, in general, known by an end-host operating systems nor by most of the home users who participated in our study.

Intermediate results that can be drawn from these emulations even before evaluating actual QFind measurements include: the QFind emulation estimates of queue size are not as accurate as the simulation estimates; using the maximum link capacity provides a better estimate of the access queue size than using the measured download data rate; ping outliers in the testbed did not cause over prediction of the queue length; small TCP receiver windows result in significant underestimation of the access queue size since the ability of the download to fill the access queue is restricted by a small maximum TCP receiver window size setting.

3.3 Measurement

The final stage of this investigation involved putting together an easy-to-follow set of directions to be used by volunteers to execute three QFind experiments and record results such they could be easily emailed to a centralized repository. One of the key elements of the whole QFind concept was to develop a measurement procedure that could be run by a variety of volunteers using different cable and DSL providers on home computers with different speeds and operating systems. To maximize participation, the intent was to avoid having users download and run custom programs and avoid any changes to system configuration

settings (such as packet size or receiver window). The final set of instructions arrived upon can be found at <http://www.cs.wpi.edu/~claypool/qfind/instructions.html>.

During January 2004, we received QFind experimental results⁴ from 47 Qfind volunteers, primarily from within the WPI CS community of undergraduate students, graduate students and faculty. These users had 16 different ISPs: Charter (16 users), Verizon (11), Comcast (4), Speakeasy (4), Earthlink (2), AOL (1), Winternet (1), RR (1), RCN (1), NetAccess (1), MTS (1), Cyberonic (1), Cox (1), Covad (1) and Adelphia (1). The QFind home nodes had 5 different operating systems: WinXP (18 users), Win2k (11), Linux (6), Mac OS-X (3), and Win98 (1) and 12 unreported. Approximately one-third of the volunteers had a wireless LAN connecting their home node to their broadband ISP.

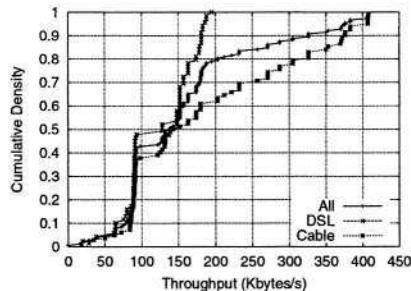


Fig. 4. Cumulative Density Functions of Throughput

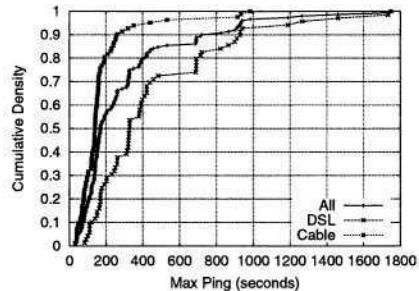


Fig. 5. Cumulative Density Functions of Max Ping Times

About 45% of the volunteers used DSL and 55% used cable modems. Figure 4 presents CDFs the throughput for all QFind tests, with separate CDFs for cable and DSL. The CDF for cable modems show a sharp increase corresponding to a standard 768 Kbps downlink capacity, which is also the median capacity. Above this median, however, the distributions separate with cable getting substantially higher throughput than DSL.

Figure 5 depicts CDFs of the maximum ping times reported for all QFind tests, with separate CDFs for cable and DSL. The median max ping time is about 200 ms, but is significantly higher for DSL (350 ms) than for cable (175 ms). Ping times of 350 ms are significant since this is enough delay to affect interactive applications [4,5]. In fact, the entire body of the DSL CDF is to the right of the cable CDF, indicating a significant difference in the max ping times for DSL versus cable. Also, the maximum ping times for cable can be up to a second and can be well over a second for DSL, a detriment to any kind of real-time interaction.

⁴ The QFind data we collected can be downloaded from <http://perform.wpi.edu/downloads/#qfind>

In analyzing the full data set to infer queue sizes (see the analysis in [3]), it appeared QFind may not clearly distinguish delays from the access queue from other system delays. We noted considerable variance in inferred access queue sizes even for volunteers within the same provider, an unlikely occurrence given that ISP providers tend to standardize their equipment at the edge by home users. This suggests that for some experiment runs, the ping delays that QFind uses to infer the queue size are a result of something other than delay at the access queue.

To remove data that does not accurately report access queue sizes, we winnow the full data set by taking advantage of the fact that the QFind volunteers produced three measurements. For each user's three measurements, if any pair have throughputs that differ by more than 10% or maximum ping times that differ by more than 10%, then all three measurements are removed. This winnowing removed the data from 17 users and all subsequent analysis is based on this winnowed data set.

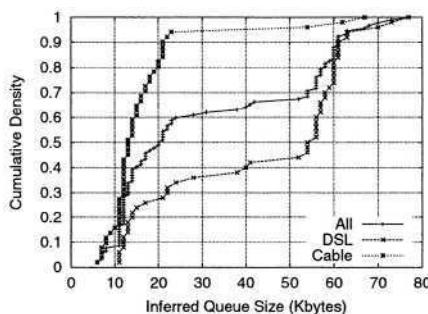


Fig. 6. Cumulative Density Functions of Access Queue Size Inferred by QFind

Figure 6 depicts a CDF of the access queue sizes measured by QFind, with separate CDFs for DSL and cable. There is a marked difference between the DSL and cable inferred queues, with cable having queue sizes under 20 Kbytes while DSL queues are generally larger. The steep increase in the DSL queue sizes around 60 Kbytes is near the limit of the receiver window size of most OSes (64 Kbytes), so the actual queue limits may be higher.

Figure 7 depicts CDFs for the access queue sizes for Charter,⁵ the primary cable provider in our data set, and non-Charter cable customers. There are some marked differences in the distributions, with Charter cable queues appearing to be slightly smaller than non-Charter cable queues. The extremely large non-Charter cable queue reported above 0.8 is from one cable provider.

⁵ <http://www.charter.com/>

Figure 8 depicts similar CDFs for the access queue sizes for Verizon,⁶ the primary DSL provider in our data set, and non-Verizon DSL customers. Here, there are very few differences between the different DSL provider distributions, suggesting there may be common queue size settings across providers.

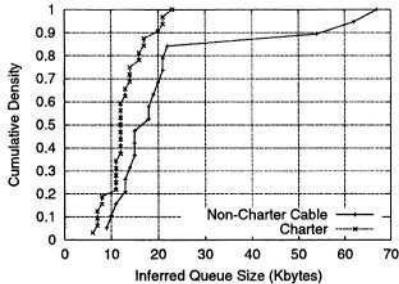


Fig. 7. Charter-Non-Charter: Inferred Access Queues

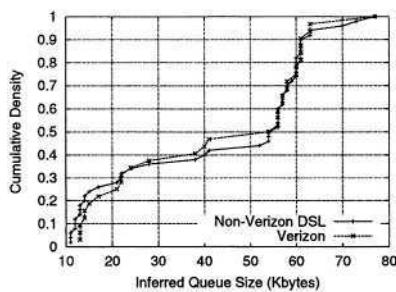


Fig. 8. Verizon-Non-Verizon: Inferred Access Queues

4 Summary

The QFind methodology for inferring queue sizes is attractive in several ways: 1) by using a standard ping and a download through a Web browser, QFind does not require any custom software or special end-host configuration; 2) by using a single TCP flow, QFind so does not cause excessive congestion. This provides the potential for QFind to be used to measure access queues from a wide-range of volunteers.

Simulation and emulation results show that QFind can be effective at inferring queue sizes, even across multiple operating systems, as long as receiver window sizes are large enough and access queues are not so small as to limit throughput.

Unfortunately, measurement results suggest QFind is substantially less accurate than in simulation for determining access queue sizes. By doing multiple QFind experiments, it is possible to ensure analysis on only consistent results, but this results in the discarding of many data samples, thus somewhat defeating the purpose of having a readily available, non-intrusive methodology.

Based on the winnowed data set from our 47 QFind volunteers, DSL appears to have significantly smaller access queues than does cable, and the corresponding ping delays when such a queue is full can significantly degrade interactive applications with real-time constraints. A secondary goal of this investigation was to determine, using both emulation and simulation, to what extent access link queue sizes can impact the throughput of flows with high round-trip times

⁶ <http://www.verizon.com/>

and the delays of flows for delay-sensitive applications, such as IP telephony and network games. Due to space constraints, these results can only be found in [3].

Future work could include exploring technologies that have been used for bandwidth estimation (a survey of such technologies is in [9]). In particular, techniques such as [10] that detect congestion by a filling router queue may be used to determine maximum queue sizes. The drawback of such techniques is they require custom software and may be intrusive, so these drawbacks would need to be weighed against the benefit of possibly more accurate results.

References

1. Aditya Akella, Srinivasan Seshan, and Anees Shaikh. An Empirical Evaluation of Wide-Area Internet Bottlenecks. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, October 2003.
2. Jae Chung, Mark Claypool, and Yali Zhu. Measurement of the Congestion Responsiveness of RealPlayer Streaming Video Over UDP. In *Proceedings of the Packet Video Workshop (PV)*, April 2003.
3. Mark Claypool, Robert Kinicki, Mingzhe Li, James Nichols, and Huahui Wu. Inferring Queue Sizes in Access Networks by Active Measurement. Technical Report WPI-CS-TR-04-04, CS Department, Worcester Polytechnic Institute, February 2004.
4. Spiros Dimolitsas, Franklin L. Corcoran, and John G. Phipps Jr. Impact of Transmission Delay on ISDN Videotelephony. In *Proceedings of Globecom '93 – IEEE Telecommunications Conference*, pages 376 – 379, Houston, TX, November 1993.
5. Tristan Henderson. Latency and User Behaviour on a Multiplayer Game Server. In *Proceedings of the Third International COST Workshop (NGC 2001)*, number 2233 in LNCS, pages 1–13, London, UK, November 2001. Springer-Verlag.
6. Sharad Jaiswal, Gianluca Iannaccone, Christophe Diot, Jim Kurose, and Don Towsley. Inferring TCP Connection Characteristics Through Passive Measurements. In *Proceedings of IEEE Infocom*, March 2004.
7. H. Jiang and C. Dovrolis. Source-Level IP Packet Bursts: Causes and Effects. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, October 2003.
8. Karthik Lakshminarayanan and Venkata Padmanabhan. Some Findings on the Network Performance of Broadband Hosts. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, October 2003.
9. R.S. Prasad, M. Murray, C. Dovrolis, and K.C. Claffy. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. *IEEE Network*, November-December 2003.
10. Vinay Ribeiro, Rudolf Riedi, Richard Baraniuk, Jiri Navratil, and Les Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *PAM*, 2003.

Reordering of IP Packets in Internet

Xiaoming Zhou and Piet Van Mieghem

Delft University of Technology

Faculty of Electrical Engineering, Mathematics, and Computer Science

P.O. Box 5031, 2600 GA Delft, The Netherlands

{x.Zhou, P.VanMieghem}@ewi.tudelft.nl

Abstract. We have analyzed the measurements of the end-to-end packet reordering by tracing UDP packets between 12 testboxes of RIPE NCC. We showed that reordering quite often happens in Internet. For bursts of 50 100-byte UDP packets, there were about 56% of all the streams arrived at the destinations out-of-order. We studied the extent of the reordering in these streams, and observed that most reordered streams have a relative small number of reordered packets: about 14% of all the streams have more than 2 reordered packets in a bursts of 50 UDP packets. In addition, we showed that packet reordering has a significant impact on UDP performance since reordering adds a high cost of recovering from the reordering on the end host. On the other hand, packet reordering does not have a significant impact on the UDP delay. We also compared the reordered stream ratios in the different directions of Internet paths, and observed that reordered stream ratios are asymmetric, but they vary largely from testbox-to-testbox.

1 Introduction

Reordering, the out-of-order arrival of packets at the destination, is a common phenomenon in the Internet [3][1], and it frequently occurs on the latest, high-speed links. The major cause of reordering has been found to be the parallelism in Internet components (switches) and links [1]. For example, due to load balancing in a router, the packets of a same stream may traverse different routers, where each packet experiences a different propagation delay, and thus may arrive at the destination out-of-order. Reordering depends on the network load, although below a certain load very little reordering occurs. Reordering may also be caused by the configuration of the hardware (i.e., multiple switches in a router) and software (i.e., class-based scheduling or priority queueing) in the routers.

The interest in analyzing end-to-end reordering is twofold. First, reordering greatly impacts the performance of applications in the Internet. In a TCP connection, the reordering of three or more packet positions within a flow may cause fast retransmission and fast recovery multiple times resulting in a reduced TCP window and consequently in a drop in link utilization and, hence, in less throughput for the application [5]. For delay-based real-time service in UDP (such as VoIP or video conference), the ability to restore order at the destination

will likely have finite limits. The deployment of a real-time service necessitates certain reordering constraints to be met. For example, in case of VoIP, to maintain the high quality of voice, packets need to be received in order, and also within 150 millisecond (ms). To verify whether these QoS requirements can be satisfied, knowledge about the reordering behavior in the Internet seems desirable. Second, these end-to-end reordering measurements may shed light on the underlying properties of the current topology and traffic patterns of the Internet.

In this paper, packet reordering is measured between 12 Internet testboxes of RIPE TTM (Test Traffic Measurement) [4] project. Our observations are based on packet level traces collected through the network. The main aim lies in the understanding of the nature of reordering. The remainder of the paper is structured as follows. In Section 2, we review the related work relevant to this paper. In Section 3, we describe the methodology used to observe reordering behavior. Our experiments are explained in Section 4. Finally, we conclude in Section 5.

2 Related Work

Quantifying the extent of reordering has been initiated by Paxson [3], and has been further investigated by Bennett *et al.* [1], and Jaiswal *et al.* [6]. During Dec. 1994 and Nov-Dec. 1995, Paxson measured the reordering ratio by tracing 20,000 TCP bulk transfers between 35 computers on the Internet. His results showed that the fraction of sessions with at least one reordering (data or acknowledgements) was 12% and 36% across the two measurement periods. About 2% of all of the data packets and 0.6% of the acknowledgements arrived out of order in the first measurement (0.3% and 0.1% in the second measurement). In 1998, Bennett *et al.* did their experiments by measuring over active TCP probe flows through the MAE-East Internet exchange point. They reported reordering in two different ways: for bursts of five 56-byte ICMP-ping packets, over 90% has at least one reordering event. After isolating a host with significant reordering, they reported that 100 packet bursts of 512-byte each produce similar results. In 2002, Jaiswal *et al.* measured and classified out-of-sequence packets in TCP connections at a single point in the Sprint IP backbone. Their results showed that the magnitude of reordering (1.57%) is much smaller than those in [1][3].

Our work distinguishes from the above in terms of experimental setup since we used and analyzed real network traces based on UDP streams.

3 Problem Description and Definitions

A packet is classified as a reordered or out-of-order packet if it has a sequence number smaller than its predecessors. Specifically, let M streams, denoted as (S_1, \dots, S_M) , be the total number of streams sent from node A to B . In each stream S_i consisting of K packets, we assign to each packet j a sequence number a_j , which is a consecutive integer from 1 to K in the order of the packet emission and so we establish the source sequence as (a_1, \dots, a_K) . Assume an output

sequence (b_1, \dots, b_P) of stream S_i observed at the receiving node B , where $P \leq K$ be the total number of packets received out of the K packets sent. The amount $K - P$ is due to loss. The sequence is said to be in order if for each index k ($1 \leq k \leq P$) holds $b_k < b_q$ ($0 < q < k$), else the stream is said to arrive at the destination out-of-order, and the packet k is a reordered packet in the reordered stream. The total number of reordered packets in stream S_i is written as L_i . For example, for the sequence of an arrived reordered stream $(1,2,3,5,4,7,6,8)$, there are 2 reordered packets (packet 4 and packet 6), which leads to $L = 2$. Note that in our paper reordering does not correlate with loss (same as [2] [8] [9]). For example, a received stream $(1,2,3,4,5,6,8)$ is considered as in order.

We denote the reordered stream ratio by

$$R_{AB} = \frac{M_R}{M_a} \quad (1)$$

where M_a is the total number of received streams out of M streams sent and M_R is the total number of streams having at least one reordered packet. The reordering asymmetry is defined to be the difference of two ratios $|R_{AB} - R_{BA}|$.

Let $U_n = \Pr[L_n > 0]$ denote the unconditional reordered stream probability for the received stream n . And let C_n denote the probability that a stream $n+1$ is reordered given that the previous stream n was reordered, defined by

$$C_n = \Pr[L_{n+1} > 0 | L_n > 0] \quad (2)$$

In order to predict whether a reordered packet will be useful in a receiver buffer with finite limit, for each reordered packet k ($1 \leq k \leq P$), this paper studies two more metrics: packet lag P_L and time lag T_L . Packet lag is proposed in [6] and refers to the number of packets k ($1 \leq k \leq P$), with a sequence number greater than the reordered packet that have been received before the reordered packet itself. Thus,

$$P_L = \sum_{q=1}^{k-1} 1_{b_k < b_q} \quad (3)$$

where the indicator function 1_y defined as 1 if the condition y is true and otherwise it is zero. For example, consider two packet sequences $(2,1,3,4,5,6,7,8)$ and $(2,3,4,5,6,7,8,1)$ which both consist of one reordered packet (packet 1), due to the different arrival positions of packet 1 in the two received sequences, then $P_L = 1$ for the previous sequence, while $P_L = 7$ for the latter sequence. For a receiver with a finite buffer or a time constraint, recovering the latter sequence from reordering may be impossible.

Let t_k ($1 \leq k \leq P$) be the 1-way delay of packet k . T_L is defined as the difference between the delay t_k of the reordered packet k and its expected delay $t_{k'}$ without reordering,

$$T_L = |t_k - t_{k'}| \quad (4)$$

In practice, $t_{k'}$ is replaced by $\min(t_1, \dots, t_P)$.

P_L is useful to evaluate the impact of reordering on TCP's performance since $P_L \geq 3$ would trigger the fast retransmit algorithms that halve the TCP sender's congestion window. We believe P_L is also a useful metric to study the impact of reordering events on UDP's performance. In addition to the P_L , T_L is a delay-based metric to more precisely evaluate the impact of reordered packets on the end hosts. For delay sensitive applications based on UDP, reordering can have a drastic effect on the application's performance. For example, in case of VoIP, to maintain the high quality of voice, packets need to be received in order, and also before playback time. If a reordered packet arrives after its playback time has elapsed, that packet may be treated as lost.

4 Experiment Results

In the following we analyze the end-to-end packets reordering measurements performed in 12 "testboxes" of RIPE TTM project. The TTM infrastructure consists of approximately 60 measurement testboxes scattered over Europe (and a few in the US and Asia). Due to the synchronizations with GPS in all testboxes, RIPE TTM achieves a delay accuracy within 10 μs . We have analyzed the data collected between 12 test-boxes; where 3 hosts are located in the Netherlands, 2 in Great Britain, and 1 in Sweden, Slovakia, Belgium, Australia, USA, Denmark and Greece. 12 testboxes participated in two experiments. Firstly, between each sender-destination pair of measurement boxes, IP probe-streams of a back-to-back burst of 50 100-byte UDP packets, called probe-streams, are continuously transmitted with interarrival times of about 30 seconds, resulting in a total of about 360 probe-streams in 3 hours from 5 to 8 PM (Greenwich Mean Time) on October 16, 2003. Secondly, we repeated the same experiment with a burst of 100 UDP packets in 3 hours from 5 to 8 PM on October 17, 2003. In order to get the snapshot of traffic patterns in the Internet, we limited each measurement to a total of 3 hours. We denoted the first experiment by N_{50} and the second by N_{100} . The difference between N_{50} and N_{100} may indicate how Internet packet dynamics change under two difference load situations. In a complete graph, ideally, 12 test-boxes should consist of exactly 132 unidirectional links. In practice, the experiment generally consisted of 104 unidirectional paths due to the erroneous effects during the measurement.

To limit the influence of large packet loss, we only analyzed those streams which received at least 90% of all their total packets (i.e. a valid arrival stream has at least 45 UDP packets in N_{50} and 90 in N_{100}).

4.1 Reordered Probe-Stream Ratio R_{AB}

R_{AB} can give insight how often reordering happened in the probe-streams. For each sender-destination pair we examined each arrival stream by checking its arrival sequence order. We calculated how many reordered probe-streams have been received over 3 hours (there are approximately 360 probe-streams). Table 1 summarizes the total number of observed UDP streams and the packets in these

streams on 104 paths over 3 hours. The results of our experiment (Table 1) indicate that reordering quite often occurs in the probe-streams. In N_{50} , about 56% of the probe-streams included at least one packet delivered out-of-order, while 66% did in N_{100} . Overall, 6% of all the received packets in N_{50} arrived reordered while 5.6% in N_{100} . It is interesting to note that this large fraction of reordering in streams is also reported by Bennett et al. [1] (over 90% with at least one reordering event). On the other hand, our results of the number of probe-stream that experience reordering is lower compared to the number in [1]. This discrepancy may be caused by methodological differences between the studies: we used UDP probe-stream in one-way, while the authors in [1] used TCP round-trip measurements.

Table 1. Details of the packets used to measure the reordering on 104 paths

Received data	N_{50}	N_{100}
UDP streams	36762	32691
Reorder streams	20445	21649
UDP packets	1655120	2828834
Reordered UDP packets	101018	158413
Measurement duration	3 hours	3 hours

We observed that a large fraction (>26%) of all UDP probe-stream suffered from reordering on all the experiment paths. In general, the probe-streams in N_{100} are more often reordered than those in N_{50} . For example, the average (over the 104 paths) is $E[R_{AB}] = 0.53$, where the standard deviation is $\sigma_{50} = 0.12$ in N_{50} . While $E[R_{AB}] = 0.65$ and $\sigma_{100} = 0.12$ in N_{100} . This is because higher traffic load likely contributes more to reordering.

We also observed that reordering varies greatly from testbox-to-testbox, for instance more than 70% of the streams transmitted from some testboxes in West Europe to two testboxes (a site in Australia, and another in Nottingham, Great Britain) in N_{50} arrived out-of-order; much higher than the 56% overall average, while 80% in N_{100} . This is may be caused by the heavy load at the links to these two testboxes.

4.2 Reordered Packet Lengths L

In order to quantify the extent of reordering, for each source-destination pair, we examined each arrival stream by checking its arrival sequence order and by calculating the reordered packet length L (the number of reordered packets).

Figure 1(a) plots a probability density function (pdf) of how many reordered packets are observed in N_{50} and in N_{100} . We found that the pdf of the reordered length has a relative heavy tail. Specifically, $L = 0$ for 44% of total probe-streams in N_{50} , $L = 1$ for 32% and $L = 2$ for another 10% of them. The maximum of L was 49 (about 0.15%). While $L = 0$ for 34% of N_{100} , $L = 1$ for 28% and $L = 2$

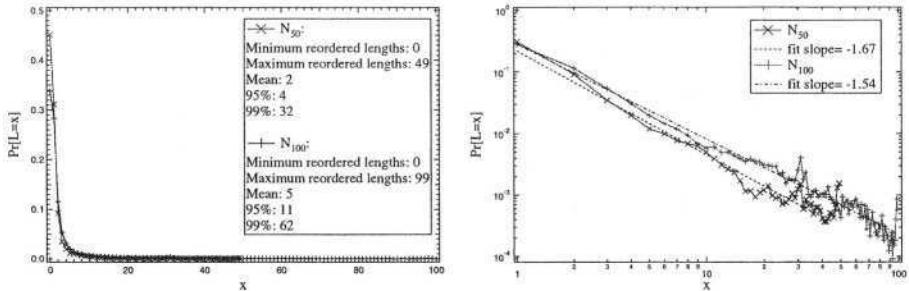


Fig. 1. (a) The pdf of the reordered packet lengths L for the RIPE data sets. (b) The pdf of the reordered packet lengths L and the power law fit

for another 12% of them in N_{100} . This suggests that most individual reordered streams have a relatively small number of lengths. Fitting the probability density function of L on a log-log scale seems to indicate power law behavior for L . A power law is defined as $\Pr[L=x] \simeq C \cdot x^{-b}$, where the exponent b is the power law exponent and slope in a log-log plot. Figure 1(b) shows the exponent $b_{50} = 1.67$ in N_{50} and $b_{100} = 1.54$ in N_{100} , which are shown in dotted lines.

We found that each IP packet in a sequence had nearly a same probability to be reordered. This suggests that the cause of reordering acts upon a stream of IP packets almost as a Poisson process.

4.3 Packet Lag P_L and Time Lag T_L

In this section, we analyzed P_L and T_L on 104 unidirectional paths. To measure P_L , for each source-destination pair, we examined each arrival stream by checking its arrival sequence order. For each reordered packet in a reordered stream, we determined P_L by calculating how many packets with greater sequence numbers have been received before it.

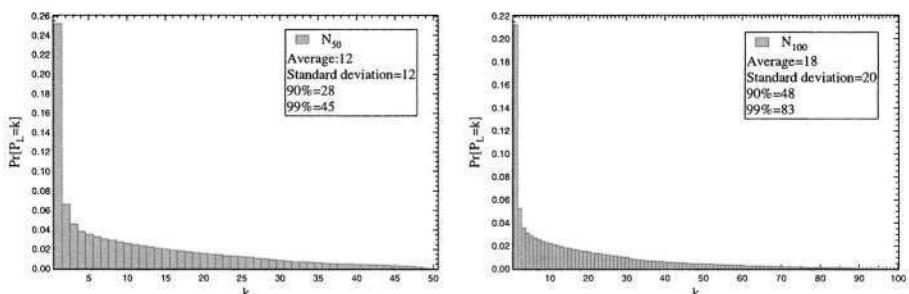


Fig. 2. The pdf of packet lag P_L for 2 data sets

The resulting pdf of P_L (Fig 2) shows that only around 40% of reordered packets occurs within 1, 2 or 3 packets in N_{50} , while around 35% in N_{100} . Moreover, the long tails of the distributions (up to 49 in N_{50} , while 99 in N_{100}) certainly impact the UDP performance because the reordering adds a high cost of recovering on the end host with finite buffer.

In the following, we computed the time lags T_L of different reordered packets. For each source-destination, we examined each arrival stream by checking its arrival sequence order to determine the reordered packets. For each reordered packet, we determined T_L by calculating the difference between the 1-way delay and the minimal delay in its sequence $\min(t_1, \dots, t_P)$. Each time lag T_L of a reordered packet was normalized by the minimal one-way delay of the packets in its sequence, thus

$$T = \frac{(t_i - \min(t_1, \dots, t_P))}{\min(t_1, \dots, t_P)} \quad (5)$$

A normalized time lag T around 0 means that T_L is very small compared to 1-way delay and a normalized T of 1 means that T_L is very comparable to 1-way delay.

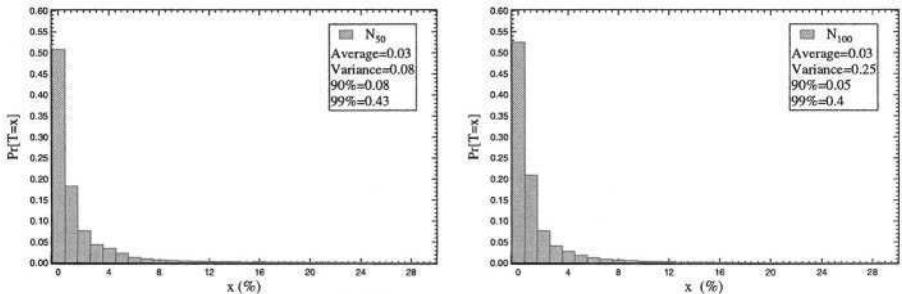


Fig. 3. The pdf of normalized time lag T for 2 data sets

Figure 3 shows the pdf of time lags normalized by the minimal one-way delay of its sequence. In our experiments the 90% percentile of the normalized time lag was 5% of the minimal one-way delay, 99% percentile of the normalized time lag was 40% of the minimal one-way delay. The Figures 3(a) and 3(b) indicate that most of the reordered time lag is very small, which suggests that packet reordering does not have a significant impact on the UDP delay since the reordering does not add large delay on the end hosts. However, the time lag can be also very large (up to 4 times the minimal one-way delay in N_{50} , while 17 times in N_{100}). Due to scale limitation, we did not show this large value in the figures.

4.4 Dependence of Reordered Probe-Streams

For each source-destination pair, we calculated the unconditional reordered streams probability U . For each reordered stream, we examined whether the next stream was out-of-order or not to calculate the conditional stream probability C .

Figure 4 presents the measured values of the conditional reordering probability C and the unconditional reordering probability U in N_{50} and N_{100} : U is close to C for most paths. The weak dependence between two consecutive streams tells us that once a stream is reordered, the probability that the next stream is reordered does not seem to depend on whether the first was reordered or not. The effects that cause reordering seem to affect bursts at random, very similar to a Poisson process (which is memoryless).

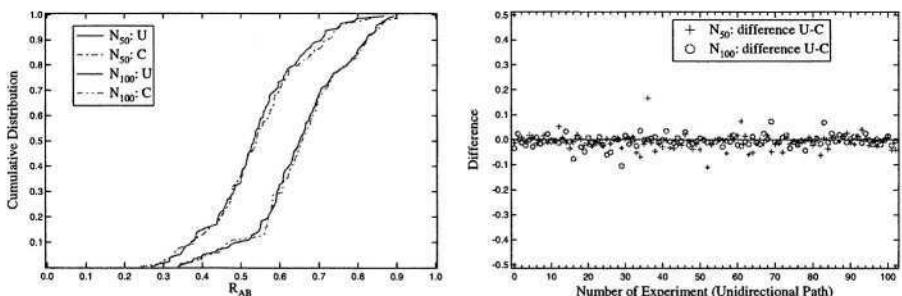


Fig. 4. The difference between U and C for 2 data sets

These observations are expected: the interarrival time between streams is large (30 seconds). The measurement shows that this interarrival time seems to be long enough to treat the streams as independent.

4.5 Asymmetry of Reordered Probe-Streams

Since unidirectional packet delay and traffic are highly asymmetric, and it would be no surprise if reordering is asymmetric as well. For a UDP-based application, such as VoIP, asymmetric reordering may result in a transaction in which one party has an acceptable quality of service while the other has not. In this section, we analyzed the asymmetry of reordered stream ratios R_{AB} and R_{BA} . We omitted pairs for which the probe-streams in one of the directions were missing or received less than 50% of all the streams sent (there are approximately 180 probe-streams), leaving the data from in total 39 pairs.

For the purpose of analysis, we defined the DAR to be the degree of asymmetry of reordering as:

$$DAR = \frac{|R_{AB} - R_{BA}|}{\min(R_{AB}, R_{BA})} \quad (6)$$

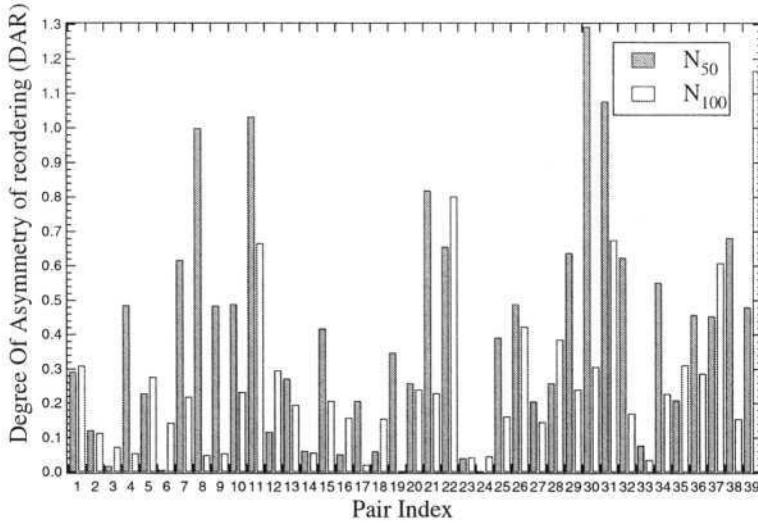


Fig. 5. Degree of Asymmetry of Reordered streams in all 39 symmetric traces. The average (over the 38 traces) is $E[DAR] = 0.41$, $\sigma_{50} = 0.32$ in N_{50} , while $E[DAR] = 0.26$, $\sigma_{100} = 0.26$ in N_{100} .

The function DAR plays a special role in the sense that it quantifies asymmetry, i.e., a DAR around 0 suggests the difference of R_{AB} and R_{BA} is very small compared to $\min(R_{AB}, R_{BA})$. Figure 5 presents the plots of unidirectional packet reordering ratios of all traces in our measurements.

We observed that the asymmetry exists on all traces, but it varies largely from testbox-to-testbox. For example, depending on the measurement set, DAR ranges from only 0.001 up to 1.29. We note that in N_{50} , 4 of the 38 traces have DAR larger than 1, and 3 of the 4 traces consist of a testbox in Slovakia. We have also studied and observed a similar behavior for N_{100} . Further, 72% of the probe-streams in N_{50} experienced reordering on a path from Greece to Slovakia (where the average probe-packets delay (on this link) is 25.1 ms, whereas the standard deviation is 19.5 ms and the hopcount is 15), while 34% of the probe-streams experienced in the opposite direction (where the average probe-packets delay is 25.9 ms, whereas the standard deviation is 10.6 ms, and hopcount is 17 or 18). We don't argue that the site-specific behavior reflect general Internet behavior, since it is found in [3] that site-specific effects can completely change. However, we suspect that the difference in stream reordering may be caused by the routing policies of the nodes on the path.

5 Conclusions

In this paper, we have analyzed the measurements of the end-to-end packet reordering by tracing UDP packets between 12 testboxes in RIPE NCC. Our results lead to several observations:

- Packet reordering is a frequent phenomenon in Internet. For bursts of 50 100-byte UDP packets, there were about 56% of the probe-streams with at least one reordering event, while about 66% for bursts of 100 100-byte UDP packets.
- Most individual streams have a relatively small number of reordering events. For bursts of 50 100-byte UDP packets, there were about 14% of probe-streams with more than two reordering events, while about 26% for bursts of 100 100-byte UDP packets. Also, the heavy tails on Figure 1(b) suggest that fitting the probability density function of reordered packet length L on a log-log scale seems to indicate power law behavior for L .
- Packet reordering has a significant impact on the UDP performance since the reordering increases a high cost of recovering on the end host. On the other hand, packet reordering does not have a significant impact on the UDP delay.
- The large interarrival time (30 seconds) between streams seems to be long enough to treat the streams as independent.
- The asymmetry of reordered streams ratios exist on all experiment pairs, but it varies greatly from testbox-to-testbox.

Acknowledgement. We thank Dr. Henk Uijterwaal of RIPE NCC for his support with the measurements of the packet reordering.

References

1. C. R. Bennett, C. Patridge and N. Shetman. “Packet Reordering is Not Pathological Network Behavior”, Trans, on Networking IEEE/ACM, December 1999.
2. A. Morton, L. Ciavattone, G. Ramachandran and J.Perser. “draft-ietf-ippm-reordering-04.txt”, IETF, work in progress.
3. V.Paxson. Automated Packet Trace Analysis of TCP Implementations. In Proceedings of the 1997 SIGCOMM Conference, pages 167-179, Cannes, France, September 1997.
4. RIPE Test Traffic Measurements, <http://www.ripe.net/ttm>
5. Michael Laor and Lior Gendel, The Effect of Packet Reordering in a Backbone Link on Application Throughput, IEEE Network, September 2002.
6. S.Jaiswal, G. Iannaccone, C. Diot, J.Kurose and D.Towsley, “Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone”, Proceedings of the ACM SIGCOMM Internet Measurement Workshop 2002, November 6-8, Marseille, France.
7. G.Iannaccone, S.Jaiswal and C.Diot, “Packet Reordering Inside the Sprint Backbone”. Technical Report TR01-ATL-062917, Sprint ATL, June 2001
8. M. Allman and E. Blanton. “On Making TCP More Robust To Packet Reordering,”, ACM Computer Communication Review, 32(1), January 2002.
9. V. Paxson, G. Almes, J. Mahdavi and M. Mathis. “Framework for IP Performance Metrics”, RFC 2330, May 1998.

Effects of Interrupt Coalescence on Network Measurements*

Ravi Prasad, Manish Jain, and Constantinos Dovrolis

College of Computing, Georgia Tech., USA

{ravi, jain, dovrolis}@cc.gatech.edu

Abstract. Several high-bandwidth network interfaces use Interrupt Coalescence (IC), i.e., they generate a single interrupt for multiple packets sent or received in a short time interval. IC decreases the per-packet interrupt processing overhead. However, IC also introduces queuing delays and alters the “dispersion” (i.e., interarrival time spacing) of packet pairs or trains. In this work, we first explain how IC works in two popular Gigabit Ethernet controllers. Then, we identify the negative effects of IC on active and passive network measurements. Specifically, we show that IC can affect active bandwidth estimation techniques, causing erroneous measurements. It can also alter the packet interarrivals in passive monitors that use commodity network interfaces. Based on the “signature” that IC leaves on the dispersion and one-way delays of packet trains, we show how to detect IC and how to filter its effects from raw measurements. Finally, we show that IC can be detrimental to TCP self-clocking, causing bursty delivery of ACKs and subsequent bursty transmission of data segments.

1 Introduction

The arrival and departure of packets at a Network Interface Card (NIC) are two events that the CPU learns about through interrupts. An interrupt-driven kernel, however, can get in a *receive livelock* state, where the CPU is fully consumed with processing network interrupts instead of processing the data contained in received packets [1]. In Gigabit Ethernet (GigE) paths, 1500-byte packets can arrive to a host in every $12\mu s$. If the interrupt processing overhead is longer than $12\mu s$, receive livelock will occur when an interrupt is generated for every arriving packet.

The system overhead for each arriving packet consists of a context switch, followed by the execution of the network interrupt service routine, followed by

* This work was supported by the “Scientific Discovery through Advanced Computing” program of the US Department of Energy (award number: DE-FG02-02ER25517), by the “Strategic Technologies for the Internet” program of the US National Science Foundation (award number: 0230841), and by an equipment donation from Intel. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the previous funding sources.

another context switch. In order to reduce the per-packet CPU overhead, and to avoid receive livelock, most high-bandwidth network interfaces today (in particular GigE cards) use *Interrupt Coalescence* (IC). IC is a technique in which NICs attempt to group multiple packets, received in a short time interval, in a single interrupt. Similarly, at the sending host, the CPU learns about the departure of several packets through a single interrupt. IC is not a new technique. It becomes common whenever a new LAN technology brings the network speed closer to the CPU speed; for instance, it was also used in the early days of Fast Ethernet NICs [1].

In this work, we identify and describe the negative effects of IC on both active and passive network measurements. Specifically, IC can affect active bandwidth estimation techniques, when the latter ignore the possibility of IC at the receiver's NIC [2]. Bandwidth measurements are classified as either capacity estimation or available bandwidth estimation. For a recent survey, we refer the reader to [3]. Capacity estimation is based on the analysis of dispersion measurements from back-to-back packet pairs and packet trains. Available bandwidth estimation, on the other hand, is based on the relation between the departure and arrival rates of periodic packet streams.

We have modified our previous capacity and available bandwidth measurement techniques, presented in [4] and [5] respectively, so that they can provide accurate estimates *in the presence of IC*. For capacity estimation, a sufficiently long packet train can provide a clear IC signature, in which packets are received in successive bursts at a certain time period. Based on this signature, we can detect IC, filter out the erroneous measurements, and estimate the correct path capacity. For available bandwidth estimation, we can similarly detect the IC signature in the one-way delay measurements of periodic packet streams. It is then simple to ignore the one-way delay measurements that have been affected by IC, and to determine whether the remaining measurements show an increasing trend.

IC can also affect passive measurements that use commodity NICs for packet trace capture. The reason is that IC can alter the packet interarrivals in the monitored traffic stream. We finally show that IC can be detrimental to TCP self-clocking, causing bursty delivery of ACKs to the sender and bursty transmission of data segments.

The rest of the paper is structured as follows. §2 explains how IC works in two popular GigE controllers. §3 illustrates the negative effects of IC in both active and passive network measurements, and it describes how to filter dispersion and one-way delay measurements in the presence of IC. We conclude in §4.

2 Description of Interrupt Coalescence

The basic objective behind IC is to reduce the number of network interrupts generated in short time intervals (in the order of a few hundredths of microseconds). This can be achieved by delaying the generation of an interrupt for a few hundreds of microseconds, expecting that more packets will be received in

the meanwhile. Most GigE NICs today support IC by buffering packets for a variable time period (*dynamic mode*), or for a fixed time period (*static mode*) [6,7]. In *dynamic mode*, the NIC controller determines an appropriate interrupt generation rate based on the network/system load. In *static mode*, the interrupt generation rate or latency is set to a fixed value, specified in the driver. Users (with root privileges) can modify the default controller behavior by changing the parameters provided by the NIC drivers. For instance, the driver for the Intel GigE E1000 controller [8] provides the following parameters for adjusting the IC receive (Rx) and transmit (Tx) operations:

- *InterruptThrottleRate*: the maximum number of interrupts per second. It is implemented by limiting the minimum time interval between consecutive interrupts.
- *(Rx/Tx)AbsIntDelay*: the delay between the arrival of the first packet after the last interrupt and the generation of a new interrupt. It controls the maximum queueing delay of a packet at the NIC buffers.
- *(Rx/Tx)IntDelay*: the delay between the last arrival of a packet and the generation of a new interrupt. It controls the minimum queueing delay of a packet at the NIC buffers.

Note that the previous parameters can be combined to meet constraints on the maximum interrupt rate and on the maximum/minimum IC-induced queueing delays. In case of conflict, *InterruptThrottleRate* has a higher precedence than *(Rx, Tx)AbsIntDelay* and *(Rx, Tx)IntDelay* parameters.

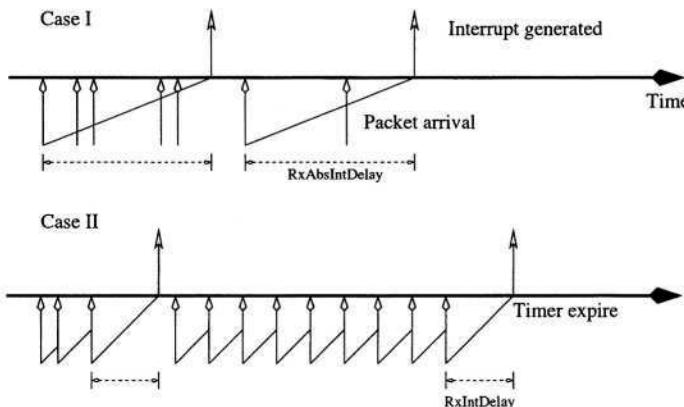


Fig. 1. Illustration of IC with *RxAbsIntDelay* (Case I) and with *RxIntDelay* (Case II).

Figure 1 illustrates the interrupt generation process at the receiving host with only the *RxIntDelay* or *RxAbsIntDelay* constraint. In case I, the *RxAbsIntDelay* timer is set to expire after a certain delay since the arrival of the first packet after

the last interrupt. All the subsequent packets are buffered at the NIC buffers, and they are delivered with the next interrupt. If the minimum packet spacing at the given NIC is T , the maximum number of packets that may need to be buffered is $RxAbsIntDelay/T$. In case II, every packet arrival resets the $RxIntDelay$ interrupt timer. Consequently, if packets arrive periodically with a lower dispersion than $RxIntDelay$, the interrupt generation can be delayed indefinitely, causing NIC buffer overflows.

The SysKollect GigE NIC [9] provides two parameters: *moderate* and *intpersec*. *Moderate* determines whether IC is enabled, and whether it is in *static* or *dynamic* mode. *Intpersec* determines the maximum interrupt generation rate in static mode. It is similar to the *InterruptThrottleRate* of the Intel GigE NIC.

3 Effects of Interrupt Coalescence

In the following, we explain how IC can affect active and passive measurements as well as TCP self-clocking. We also describe the IC signature on packet train dispersion and one-way delay measurements, and show how to filter out the effects of IC.

3.1 Capacity Estimation

Active bandwidth estimation tools use the received dispersion (interarrival time spacing) of back-to-back packet pairs (or trains) to estimate the end-to-end capacity of a network path [3]. The basic idea is that, in the absence of cross traffic, the dispersion of a packet pair is inversely proportional to the path capacity. Note that the received dispersion is typically measured at the application or kernel, and not at the NIC. If the receiver NIC performs IC, however, the packet pair will be delivered to the kernel with a single interrupt, destroying the dispersion that the packet pair had at the network.

Figure 2 shows the cumulative dispersion of an 100-packet Ethernet MTU (1500B) train, with and without IC, at the receiver of a GigE path. The capacity estimate, as obtained without IC, is quite close to the actual capacity of the path (about 1000Mbps). In the case of IC, however, we see that the application receives 10 to 12 packets at a very high rate (unrelated to the network capacity), separated by about $125\mu s$ of idle time. Note that, if a bandwidth estimation tool would attempt to measure the path capacity from the dispersion of packet pairs, it would fail miserably because there is not a single packet pair with the correct dispersion.

Detection of IC: In the presence of IC, many packets will be delivered from the NIC to the kernel, and then to the application, with a single interrupt. We refer to the packets that are transferred to the application with a single interrupt as a *burst*. The dispersion of successive packets in a burst, measured at the application layer, will be equal to the latency T of the *recyfrom* system

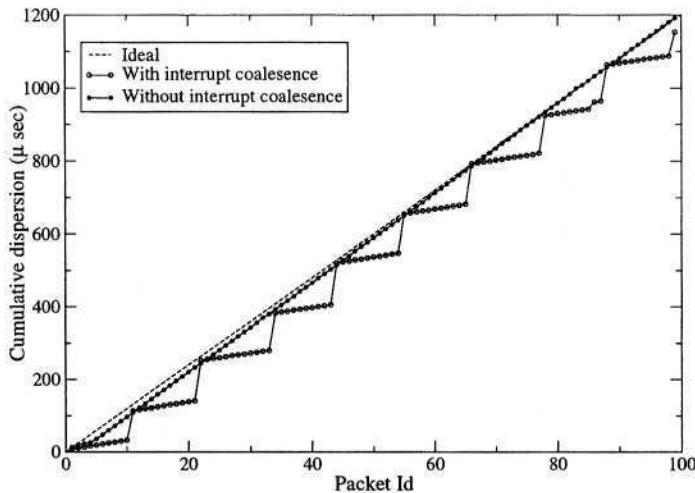


Fig. 2. Cumulative dispersion in an 100-packet train with and without IC.

call. A measurement tool can measure T , and so it can determine which packets of a received train form a *burst*.

The presence of IC at the receiver NIC can be detected by sending a long back-to-back packet train from the sender to the receiver. If IC is enabled, the receiving application will observe a sequence of bursts of approximately the same length and duration. For example, Figure 3 shows that bursts of 10-12 packets arrive roughly every $125\mu\text{s}$. A packet train of about 50 packets would be sufficient to detect the IC signature reliably.

An important point here is that a context switch at the receiver can cause a somewhat similar signature with IC. Figure 3 illustrates that context switching can also force several consecutive packets to be delivered to the application with the *recvfrom* period. However, as shown in Figure 3, there is a clear difference between the signatures of context switching and IC. The former is a probabilistic event that only affects a subset of successive packets, starting from a random initial location in the packet train. The latter is a deterministic event that affects all the packets of a packet train, starting from the first, and creating a regular pattern of burst lengths and durations.

Estimation with IC: Our capacity estimation tool, pathrate [10], uses the dispersion of consecutive packets in 50-packet trains to detect the presence of IC. To estimate the path capacity in the presence of IC, pathrate relies on the interarrivals between successive bursts. Without significant cross-traffic, the dispersion between the first packet of two consecutive bursts is equal to BL/C , where B is the length of the first burst, L is the packet size, and C is the path capacity. The train length should be sufficiently long so that at least two bursts

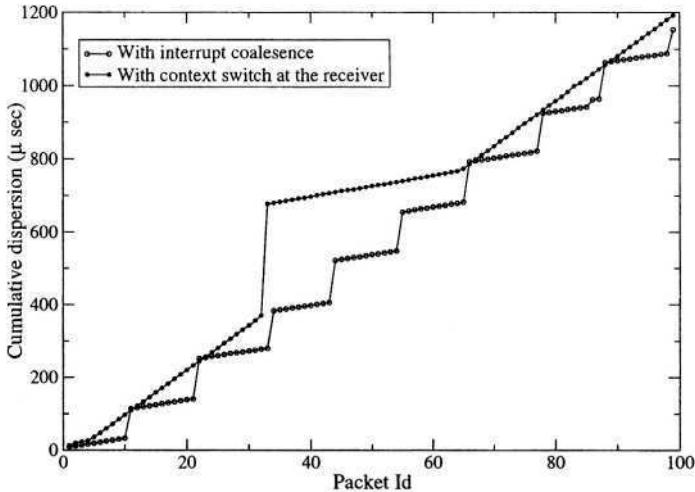


Fig. 3. The signatures of context switching and interrupt coalescence in the cumulative dispersion of a packet train.

are observed in each received train. Pathrate sends multiple packet trains and finally reports the median of the resulting capacity estimates.

3.2 Available Bandwidth Estimation

Available bandwidth estimation tools examine the relationship between the *send rate* R_s and *receive rate* R_r of periodic packet streams in order to measure the end-to-end available bandwidth A [3,11,12,13]. The main idea is that when $R_s > A$, then $R_r < R_s$; otherwise, $R_r = R_s$. Furthermore, when $R_s > A$, the queue of the tight link builds up, and so packet i experiences a larger queuing delay than packet $i-1$. Consequently, when $R_s > A$, the One-Way Delay (OWD) of successive probing packets is expected to show an increasing trend. On the other hand, if $R_s < A$, the OWDs of probing packets will be roughly equal. Pathload [5] and pathchirp [14] estimate the available bandwidth of a path by analyzing OWD variations.

Figure 4 shows the OWDs of a periodic 100-packet Ethernet MTU stream with and without IC. The sender timestamps and transmits packets at 1.5Gbps. The available bandwidth in the path is about 940 Mbps. The OWDs of successive packets, without IC, are clearly increasing showing that the probing rate is greater than the available bandwidth. However, in the presence of IC, packets are buffered at the NIC until the interrupt timer expires. Then, the packets are delivered back-to-back to the kernel, and then to the application. We refer to all successive packets delivered with a single interrupt as a *burst*. Among the packets of a burst, the first experiences the largest queueing delay at the NIC, while the last observes the minimum (or zero) queueing delay at the NIC. Notice

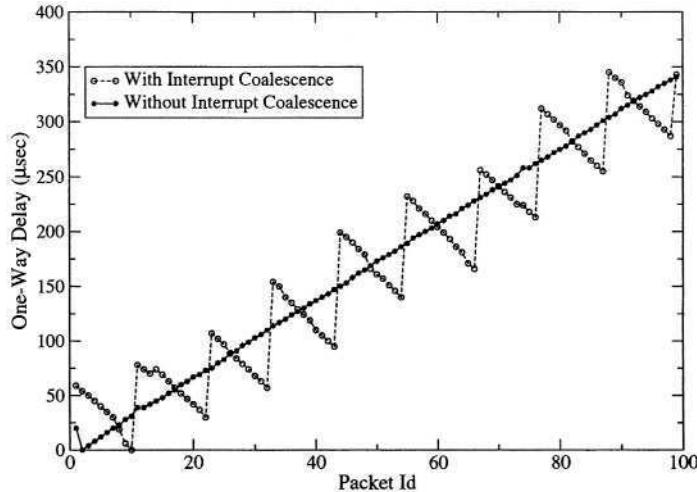


Fig. 4. OWDs in 100-packet train with and without IC.

that the increasing OWD trend of the packets within a burst has been destroyed due to IC.

Suppose that s_k is the send time of the k 'th packet in a certain burst, while t is when the interrupt is generated. Also, let r be the latency to transfer a packet from the NIC to user space, and g be the period between successive packets at the sender. Then, the OWD of the k 'th packet will be given by

$$\begin{aligned} d_k &= t + k * r - s_k \\ &= t + k * r - (s_1 + k * g) \end{aligned}$$

and so the relative OWD of packet $k + 1$ will be

$$d_{k+1} - d_k = r - g \quad (1)$$

From Equation (1), we see that if $g > r$, the successive OWDs in the same burst are decreasing. In the experiment of Figure 4, we have that $g = 8\mu\text{s}$, $r \approx 3\mu\text{s}$, and so there is a clear decreasing trend in the OWDs of each burst.

Note that tools that only use packet pairs, such as [12,14], or short packet streams (less than about 10 packets in our GigE cards) will fail to detect IC, since none of the packet pairs, or short streams, shows a correct OWD trend. On the other hand, notice that, although the OWD trend is destroyed in short timescales, the overall increasing trend in the OWDs is still preserved. Based on this observation, we have included an algorithm in pathload to detect IC and measure available bandwidth even in the presence of IC.

Estimation with IC: Pathload uses the same technique as pathrate (see §3.1), to detect whether the receiver NIC performs IC. When that is the case, pathload filters out the OWD measurements that have been affected the most from

IC. As explained in the previous paragraph, the queueing delay at the NIC is minimum (or zero) for the last packet of a burst. So, pathload rejects all OWD measurements *except the last of each burst*. From the remaining measurements, we can then examine whether an increasing OWD trend exists or not.

3.3 Passive Measurements

Passive monitors are often used to collect traces of packet headers and inter-arrivals. Several studies have focused on the traffic statistical characteristics (burstiness) in short timescales, by analyzing such packet traces. However, if the traces have been collected with commodity NICs that perform IC, then the packet interarrivals can be significantly altered. First, IC can make several packets appear as if they form a burst, even though that may not be the case. Second, IC can affect the correlation structure of packet interarrivals in short timescales.

We emphasize that special-purpose passive monitors typically timestamp each packet at the NIC, and so they avoid the negative effects of IC [15].

3.4 TCP Self-Clocking

Another negative effect of IC is that it can break TCP self-clocking [16]. Specifically, the TCP sender establishes its *self-clock*, i.e. determines how often it should be sending packets based on the dispersion of the received ACKs. To preserve the dispersion of ACKs at the sender, or the dispersion of data segments at the receiver, packets must be delivered to TCP as soon as they arrive at the NIC. IC, however, results in bursty delivery of data segments to the receiver, destroying the dispersion that those packets had in the network. The bursty arrival of segments at the receiver causes bursty transmission of ACKs, and subsequently bursty transmission of more data segments from the sender. The problem with such bursts is that TCP can experience significant losses in under-buffered network paths, even if the corresponding links are not saturated [17].

Figure 5 shows the CDF of ACK interarrivals in a 10-second TCP connection over a GigE path. Without IC, most ACKs arrive approximately every $24\mu\text{s}$, corresponding to the transmission time of two MTU packets at a GigE interface. This is because TCP uses delayed-ACKs, acknowledging every second packet. With IC, however, approximately 65% of the ACKs arrive with an erratic dispersion of less than $1\mu\text{s}$, as they are delivered to the kernel with a single interrupt. These “batched” ACKs trigger the transmission of long bursts of data packets from the sender. Note that the rest of the ACKs, even though non-batched, they still have a dispersion that does not correspond to the true capacity of the path, misleading the TCP sender about the actual rate that this path can sustain.

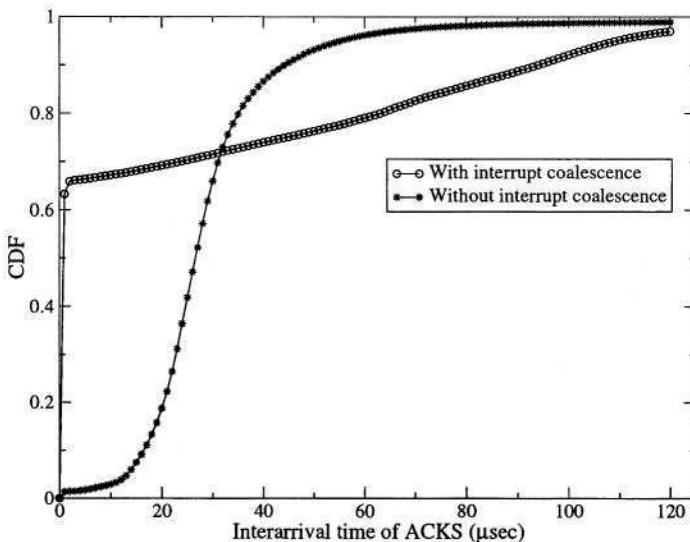


Fig. 5. CDF of ACK interarrivals in a 10-second TCP connection over a GigE path.

4 Conclusions

IC is used to reduce the interrupt processing overhead and becomes necessary when the minimum packet interarrival latency becomes comparable to the per-packet interrupt processing overhead. Unfortunately, IC can affect active network measurement tools that use closely spaced packet pairs, trains, or streams. In particular, capacity and available bandwidth estimation techniques can provide incorrect results if they ignore IC. On the positive side, we developed a simple algorithm to detect the presence of IC, and to filter out the erroneous dispersion or one-way delay measurements caused by IC.

References

1. Mogul, J.C., Ramakrishnan, K.K.: Eliminating receive livelock in an interrupt-driven kernel. *ACM Transactions on Computer Systems* **15** (1997) 217–252
2. Jin, G., Tierney, B.L.: System Capability Effects on Algorithms for Network Bandwidth Measurement. In: Proceedings of ACM Internet Measurement Conference. (2003)
3. Prasad, R.S., Murray, M., Dovrolis, C., Claffy, K.: Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. *IEEE Network* (2003)
4. Dovrolis, C., Ramanathan, P., Moore, D.: Packet Dispersion Techniques and Capacity Estimation. *IEEE/ACM Transactions on Networking* (2004)
5. Jain, M., Dovrolis, C.: End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. *IEEE/ACM Transactions on Networking* **11** (2003) 537–549

6. Intel: Interrupt Moderation Using Intel Gigabit Ethernet Controllers.
<http://www.intel.com/design/network/applnotes/ap450.pdf> (2003)
7. Syskconnect: SK-NET GE Gigabit Ethernet Server Adapter.
http://www.syskconnect.com/syskconnect/technology/SK-NET_GE.PDF (2003)
8. Intel: Intel Gigabit Ethernet Driver.
<http://sourceforge.net/projects/e1000> (2004)
9. Syskconnect: SysKconnect Gigabit Ethernet Driver.
<http://www.syskconnect.com/syskconnect/support/driver/ge.htm> (2003)
10. Pathrate and Pathload. <http://www.pathrate.org/> (2004)
11. Melander, B., Bjorkman, M., Gunningberg, P.: A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks. In: IEEE Global Internet Symposium. (2000)
12. Pasztor, A.: Accurate Active Measurement in the Internet and its Applications. PhD thesis, The University of Melbourne (2003)
13. Hu, N., Steenkiste, P.: Evaluation and Characterization of Available Bandwidth Probing Techniques. IEEE Journal on Selected Areas in Communications **21** (2003) 879–894
14. Ribeiro, V., Riedi, R., Baraniuk, R., Navratil, J., Cottrell, L.: pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In: Proceedings of Passive and Active Measurements (PAM) workshop. (2003)
15. Endace: Endace Measurement Systems. <http://dag.cs.waikato.ac.nz/> (2004)
16. Jacobson, V.: Congestion Avoidance and Control. In: Proceedings of ACM SIGCOMM. (1988) 314–329
17. Zee, M., Mikuc, M., Zagar, M.: Estimating the Impact of Interrupt Coalescing Delays on Steady State TCP Throughput. (2002)

Measurement Approaches to Evaluate Performance Optimizations for Wide-Area Wireless Networks

Rajiv Chakravorty¹, Julian Chesterfield¹, Pablo Rodriguez², and Suman Banerjee³

¹ University of Cambridge Computer Laboratory, Cambridge CB3 0FD, UK

{Rajiv.Chakravorty,Julian.Chesterfield}@cl.cam.ac.uk

² Microsoft Research, Cambridge CB3 0FD, UK

pablo@microsoft.com

³ University of Wisconsin, Madison WI 53706, USA

suman@cs.wisc.edu

Abstract. We present measurement approaches to evaluate performance optimizations, employed at different layers of the protocol stack, to enhance application performance over wide-area wireless networks (WWANs). Applications running over WWAN cellular environments (e.g web browsing) are significantly affected by the vagaries of the cellular wireless links. Much of the prior research has focussed on variety of isolated performance optimizations and their measurements over wired and wireless environments. In this paper we introduce experiment-based measurement approaches to benchmark application performance using optimizations performed at individual layers of the protocol stack.

These measurement initiatives are aimed at: (1) performing an accurate benchmark of application performance over commercially deployed WWAN environments, (2) characterizing the impact of a wide selection of optimization techniques applied at different layers of the protocol stack, and (3) quantifying the interdependencies between the different optimization techniques and providing measurement initiatives for future experimentation to obtain consistent and repeatable application benchmarks in WWAN environments.

1 Introduction

All over the world wide-area wireless networks (WWANs) are being upgraded to support 2.5G and 3G mobile data services. Unfortunately, application performance over WWANs is severely impacted by problems of high and variable round trip times, fluctuating bandwidths, frequent link outages, burst losses, etc. [1]. As a consequence, the end-user experience in such environments is significantly different from the relatively stable indoor wireless environments, e.g. 802.11b based Wireless LANs (WLANS).

In this paper we consider measurement approaches and performance benchmarks over WWANs from a end-user perspective. Our performance study explores questions like:

- What measurement approaches can yield reproducible and repeatable experiments over WWANs?
- What factors contribute to the poor application performance (web) over WWANs?
- What different optimizations can be applied at individual layers of the protocol stack and what is the benefit available from each?

To answer these questions, we conduct an empirical performance study involving real WWAN networks and applications. Our approach significantly differs from all previous approaches and work conducted over WWANs. While research in this direction has investigated large-scale performance study of end-to-end TCP flows [5] and also its cross-layer interaction with the link-layer [3,4], the approach taken in our paper is different in several ways. First, we precisely quantify the causes of poor application performance and quantify real *user experience* over WWANs. Here we accurately measure the different components that contribute to the latencies during web downloads for a range of popular websites (ranked in www.100hot.com). Second, we introduce *virtual web hosting* as an important construct to perform repeatable web browsing experiments over WWANs. Third, we benchmark all standard web browsers, protocols, and techniques with respect to their performance. Finally, we implement and study a wide selection of optimization techniques at different layers and their cross layer interactions on application performance.

Our paper is laid out as follows. The next section describes our experimental WWAN infrastructure and elaborates on our methodology to conduct repeatable and reproducible experiments over WWANs. In section 3, we discuss some of our empirical findings for optimizations applied at different layers of the protocol stack. Section 4 discusses our results while the last section concludes the paper.

2 Testbed Infrastructure and Methodology

We focussed our experimental evaluation on web-browsing performance over a WWAN testbed. For this, we used a commercial GPRS-based WWAN network in our experiments as shown in figure 1. In this testbed, the mobile terminal (MT), e.g. a laptop, connects to the GPRS network through a GPRS-enabled interface – a PCMCIA GPRS card or a phone. In our experiments the MT (or mobile client) downloaded web content over the WWAN link from different content locations: (1) directly from the real web servers, e.g. CNN, Yahoo, and (2) virtually hosted web-servers (explained later in this section) that were located in our laboratory.

To study the different optimization techniques at different layers of the protocol stack and their overall impact on application (web) performance, our experiments also required us to implement optimization-specific proxies. Based on the use of proxies, our experiments were classified into three modes:

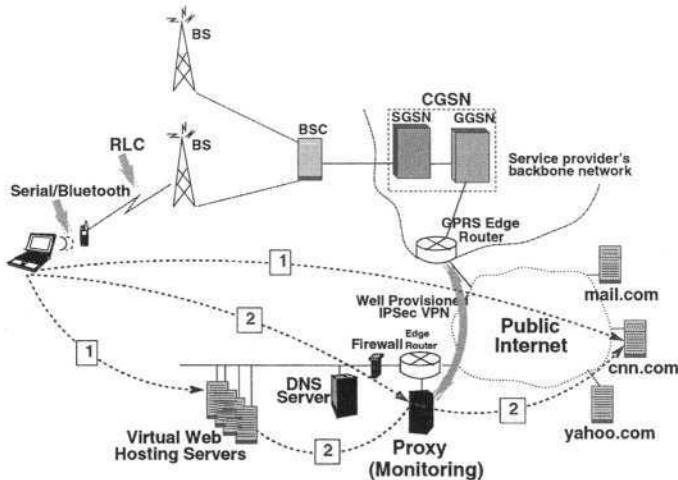


Fig. 1. WWAN Architecture and Testbed. In these experiments, we placed the proxy in our laboratory and then use a well provisioned IPSec VPN to ‘back haul’ GPRS traffic to it directly from the cellular provider’s network. In the proxy-based experiments, the mobile client connects to the web servers through this proxy (Label 2 in Figure 1).

- **No Proxy Mode:** In this case the client directly connected to the server and the experiments did not require any intervening proxy. These optimizations are the easiest to deploy.
- **Transparent Proxy Mode:** This mode is used for those experiments where the client need not be aware of the existence of a proxy and the cellular provider’s network *transparently* guides the client’s connections through a proxy as necessary. Transparent proxy solutions are also easy to deploy.
- **Explicit Proxy Mode:** This mode was used in experiments which require the mobile client to be aware of the proxy in the network (in this case called the ‘server-side’ proxy). This requires either (a) explicit browser configuration or (b) software update at the mobile client to make it appropriately interact with the server-side proxy. The software update is like a client-side proxy and hence we refer to this approach as a dual-proxy solution.

Furthermore, in our experiments we have used *virtual web hosting* to emulate real web downloads. Virtual web hosting is an important construct to perform repeatable web browsing experiments over WWAN links involving typically fast-changing websites.

Why Virtual Web Hosting? Contents of popular websites change very frequently (e.g. in CNN content changes within minutes). If real web-download experiments were to be conducted over low-bandwidth WWAN links involving such web-sites, then different download attempts may notice significant differences in the downloaded content structure and volume. In other words, the

total time to perform each set of experiments for every individual website was much higher than the time it takes for the web-page content to change. Hence it would not have been feasible for us to make meaningful comparisons performed directly using real websites. To avoid this problem we implemented a *virtual web hosting* system in our laboratory, where we statically replicated the contents of the popular websites into a set of web servers (hosted in our laboratory) that were made publicly accessible for the mobile client. Thus a mobile client can access the virtually hosted webpages using WWAN networks just as they would from the real servers *in a repeatable and reproducible fashion*.

Replicating Distributed Web Content for WWANs. Web downloads of popular websites such as www.cnn.com access a number of distinct domains spread across multiple CDN servers, e.g., Akamai, to download content (see figure 2). This access pattern significantly affects the download performance over WWAN links for a number of reasons, including (1) high number of DNS look-ups, (2) number of TCP connections opened by the client to these different servers, etc. To emulate this aspect of web downloads in the virtual web hosting setup, it was necessary to faithfully replicate the distributed web content and its overall structure. For each server in the original website, we assigned a separate web server in our laboratory to “virtually” host the corresponding content. The domain names of these virtual-hosting servers were constructed from their original domain names by pre-pending the corresponding CDN server domain names. These modified domain names were made available to the DNS. Additionally we updated the URLs pointing to the embedded content to reflect the new domain names. Thus, in a virtual web hosting experiment when a mobile client attempts to download a webpage, it would have to appropriately resolve different domain names for the different content servers similar to the case of a real web download.

Our experiments performed using virtual web hosting replicate the key components of the web browsing performance that any WWAN user would experience with actual web servers. However, there exists few differences between overall performance observed using the real web servers and virtual web hosting scenario. We have observed that the mean download latencies are lower (by about 5-10%) for the virtual-hosting system. This is primarily due to the (1) absence of dynamically generated content, (2) difference in server workload and processing times in the virtual-hosting case. We emphasize that none of the above performance differences change the qualitative nature of the results when comparing the different optimization techniques.

3 Experiences with Performance Optimizations

Our experimental evaluation is focused on web-browsing performance over a WWAN network. We have experimented with different standard web-browsers available (e.g. Mozilla, Internet Explorer, Netscape). Though there are minor variations in their implementations, we observed that their performance is roug-

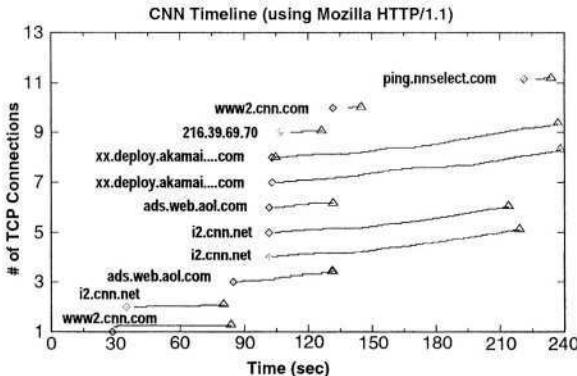


Fig. 2. Timeline for an example web download over WWAN networks, using Mozilla/HTTP/1.1. The web content is spread over 6 servers and multiple connections are opened by the browser to these servers. As the HTTP/1.1 default behavior dictates, only two simultaneous TCP connections are opened to a specific server. Each small rise in the lines indicates a separate GET request made using that specific connection.

Table 1. Web Download latencies (using Mozilla/HTTP/1.1) and other characteristics for 4 popular websites and their content distribution. During experiments, mobile host was stationary and reasonably good link conditions (e.g. typical C/I > 15dB). HTTP 1.1 achieves *abysmally* low throughput over WWANs.

Website	Download latency (sec)		No. of Dom.	Emb. Objects (Size in KB)				T'put(Kb/s)
	WWAN-Real	WWAN-Virtual		Count	Sum	Avg.	Max.	
mail	43.3 (5.5)	34.5 (3.4)	4	11	36.7	3.3	11.0	8.5
yahoo	38.8 (4.1)	35.0 (3.1)	6	16	60.3	3.8	36.0	13.8
amazon	102.3 (9.8)	76.4 (7.7)	3	42	91.9	2.2	46.8	9.6
cnn	204.0 (17.6)	196.3 (12.4)	6	67	186.8	2.8	22.3	7.6

hly similar. Our results show that the default configuration parameters of most browsers (typically chosen to work well in wired networks or wireless LANs) perform poorly in WWAN environments. This is surprising in the context of prior work [4], which showed that TCP, the underlying transport protocol used by HTTP, makes efficient use of the WWAN wireless link. Our results also show that individual TCP connections are relatively efficient over these wireless links. However, the HTTP protocol needs to be suitably adapted to improve its performance over WWAN environments.

In order to precisely benchmark web performance, we have used the Mozilla browser version 1.4. In its default setting Mozilla opens upto 8 simultaneous TCP connections per web-server using HTTP 1.0 and upto 2 TCP connections using HTTP/1.1, Mozilla also supports proposed experimental features in HTTP/1.1, e.g. pipelining.

Table 2. Data throughputs achieved for ftp-downloads over WWAN wireless links using a single TCP connection. TCP achieves good throughput for larger files.

File Size (KB)	FTP-throughput (Kbps)
1	13.2 (1.5)
5	18.1 (0.9)
10	18.8 (2.1)
50	29.7 (3.3)
100	30.5 (3.2)

3.1 Performance Benchmarks

We conducted experiments for a number of different websites and we briefly summarize four of them in Table 1. These four websites were chosen based on the diversity of their characteristics, content types, content volumes, and number of servers used. The download latencies of the different websites have significant variability due to the diversity in content and the multiplicity of servers. The table also indicates the overall data throughput achieved in downloading these websites. We can observe that the overall throughput is significantly low. It varies between only 7.5 Kbps to 17 Kbps for different websites, even though the ideal downlink data-rate is 39.6 Kbps. We can contrast the performance of this web download to ftp-like data transfers presented in Table 2. In this table we present the throughput achieved when we downloaded a single file (of different sizes) over the same WWAN wireless link.

The throughput achieved in such file transfer experiments were significantly higher than the web downloads. For example the web download throughput for amazon.com with a total content size of 91.9 KB was 9.6 Kbps, while the download of a single 50 or 100 KB file was around 30 Kbps! The high file transfer data throughput confirms prior observations made by Ludwig et. al. [4] that TCP performs quite well over GSM-based wireless links. This implies that there are significant inefficiencies in the web download mechanisms and carefully applied optimizations can significantly improve the performance.

3.2 Performance Optimizations

We have examined the performance of a wide-selection of optimization techniques that have been proposed at the different layers of the protocol stack — application, session, transport, and link. As discussed in Section 2 some of these optimization techniques relied on a transparent or explicit proxy that was located in our laboratory. In this section we will discuss the benefits observed by each of these techniques, except for the explicit dual-proxy techniques in most cases. The dual-proxy techniques works with very different assumptions of deployment and hence it is not possible to make a fair comparison of these techniques with the no-proxy or single-proxy techniques. Therefore, we will only comment on the benefits of the schemes individually and their combined

effects in the summary of results (Section 4). We now discuss performance optimizations.

Application layer Optimizations. For application layer optimizations, we quantified the benefits of schemes like HTTP pipelining, extended caching, delta encoding, and dynamic content compression.

Dynamic Data Compression. We implemented dynamic content compression using an application-level proxy operating in the transparent as well as the explicit dual-proxy mode. From our experiments, we have observed that the content in the different websites are very compressible. However, the benefits of compression on application performance may not be as substantial (except for the case of Yahoo). This apparent anomalous behavior is due to the typical object size distribution of some webpages. Here we observe that most of the objects in the webpages can be small, e.g. nearly 60% of the objects in a CNN snapshot were less than 1 KB (typically 1 TCP segment, assuming 1460 byte payloads of IP packets). Any amount of compression would clearly not change the number of segments below one. Therefore the overheads of issuing individual GET requests for these objects sequentially over the two TCP connections dominates the transfer time of these objects and hence the improvement in data transfer latency due to compression will be minimal in these cases. In contrast, for web sites where the distribution of object sizes is skewed towards larger values (e.g. Yahoo) the impact on download latencies is higher.

HTTP Pipelining. We evaluated performance of the HTTP 1.1 protocol. The default persistent HTTP/1.1 protocol gets each of these small objects sequentially over its two TCP connections, and waits numerous times between the completion of each GET request and the beginning of the next. In contrast, HTTP pipelining allows many GET requests to be issued simultaneously by the mobile client and hence the objects are fetched without any intervening gaps. From our experiments, we see that HTTP pipelining provides between 35% to 56% benefits for the different websites. HTTP pipelining is an experimental technique in the HTTP/1.1 standard and we found that, unfortunately, most browsers do not enable this feature by default.

CHK-based Caching/Delta Compression. We also investigated performance of extended CHK-based caching and delta coding for different web-sites. Our experiments show that such techniques on average improves real web-browsing experience by about 3-6% for fast-changing web-sites.

Session level Optimizations. We performed a detailed study of performance enhancement schemes like (1) the impact of multiple simultaneous transport connections as typical in standard web browsers, (2) impact of DNS look-ups on web downloads [2], and, (3) parse-and-push technique.

Varying TCP Connections. We investigated an alternative session layer technique to optimally choose the number of simultaneous TCP connections opened by the client to the server. We found that for a base capacity of the GPRS handset (39.6 Kbps in our case) increasing the number of TCP connections (from 2

to 6) leads to significant improvement in the user experience (i.e. for CNN the download latency reduces from 196.3 seconds to 123.0 seconds).

DNS Boosting. DNS-boosting achieves the same effect as URL re-writing (as in Content Distribution Networks) by intelligently manipulating DNS queries from the client. Specific details of this scheme is available in [2]. Note that this can significantly benefit performance in two ways: (1) by avoiding extra DNS Lookups and (2) by reducing the number of TCP connections opened by a web browser. We implemented this technique as a proxy and performed download experiments for the different websites. By eliminating the DNS lookups for the transparent proxy, we achieve another 5-9% improvement in the download latency. The net improvements due to the session and application techniques are between 53-65%.

Parse-n-Push. Parse-and-push is a session-level, explicit, dual-proxy scheme that emulates deterministic content pushing towards the mobile client, when the wireless downlink would have been otherwise left idle. While supporting parse-and-push mechanism requires explicit client-side software update, the scheme helps to improve overall utilization of the link. Our experiments have shown that Parse-and-push provides an additional 5%-12% improvement in the web download latency for the popular websites.

Transport layer Optimizations. We evaluated the performance of standard TCP, a recently proposed link-adapted variant suited for WWAN environments (TCP-WWAN), and a customized UDP based transport (UDP-GPRS) solution.

Using two different proxies, we quantified the additional benefits of using link-adapted TCP and custom UDP based solution. In these experiments, we apply the application-level optimizations (full compression) and session-level optimizations. We have observed that using TCP-WWAN (transparently deployed) achieves between 5-13% additional benefits for the different websites. UDP-GPRS custom protocol (dual-proxy approach) leverages its specific knowledge of the wireless link characteristics to improve the download performance further (between 7-14% for the different websites).

Link layer Optimizations. Using trace-based simulations, we have studied the interaction between link-layer retransmissions (ARQ) and forward error correction (FEC) schemes in WWAN environments. We have investigated mechanisms that allow the RLC to dynamically choose the encoding schemes in conjunction with the ability to enable or disable ARQ, and the impact of such mechanisms on applications. Performing actual experimentation for this study was difficult since we had no control on the encoding schemes used by the Base Station to transmit data to the mobile client. At the mobile client we only had the flexibility to enable or disable ARQ, and the ability to disable FECs. Therefore, we performed trace-based simulations to study the data performance for various applications over a wide range of channel conditions and encoding choices.

Our study confirms that for each different channel condition there is an optimal value of FEC that leads to the least download latency. For example a

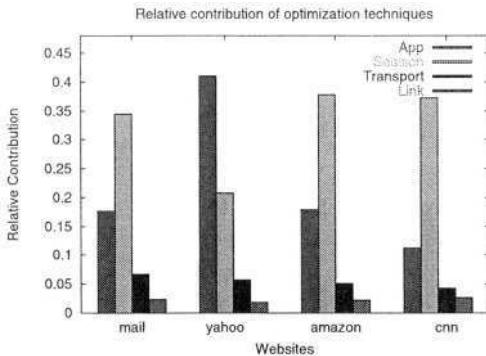


Fig. 3. Relative contribution of optimizations for 4 popular web-sites.

moderately poor channel, with an error rate of 0.9% on the GPRS channel, 5-6% FEC is the optimal choice to minimize download times. The amount of required FEC for such optimal performance increases with increase in channel error rates. This suggests that the RLC should continuously monitor the channel conditions and *dynamically* choose the amount of FEC to be applied on reliable data transfers across the wireless links.

4 Summary of Results

From the experiments conducted, we classified the performance optimizations into two classes — those that require re-configuration or software update in the mobile client, i.e. uses an explicit proxy and those that have no such requirements. Here we assumed a reasonably good wireless link (error less than 0.2%) where dynamic (adaptive) FECs provide a latency improvement of about 5%. This value is derived from our link-layer trace-based simulations to improve performance.

In Figure 3, we only plot the relative contribution of the schemes that require no client-side reconfiguration when all optimizations *are applied simultaneously*. For example, in Amazon application, session, transport, and link layer optimization techniques contribute 17.9%, 37.8%, 5.1%, and 2.2% respectively. The improvement provided by all the techniques applied simultaneously were the sum of these values, 63.0%, which brought the download latency for Amazon from 76.4 seconds to 29.3 seconds. ***In general, we can observe that application and session layer techniques have a dominating effect in improving the web performance.*** They lead to 48-61% web performance improvements for our example websites. Thus our work demonstrates that the application and session-level mechanisms currently deployed for web browsing applications make poor use of the relatively efficient lower layers. Employing appropriate optimizations at these layers (as described in this paper) can help bridging this performance gap observed between the upper and lower layers. Our results show the

benefits to be somewhat higher when client-side reconfiguration/software update is applied.

Note that transport, and link layers optimizations typically provide an additional 5-10% performance improvement (considering reasonably good link conditions), which is still significant for web downloads over WWAN links.

5 Conclusions and Ongoing Work

Preliminary results from our comparative performance study of different optimization techniques reveals the following: (1) There is a significant mismatch in the performance of default HTTP protocols and its underlying transport mechanism, TCP. Unlike wireline networks, standard web browsers are unable to exploit even the meagre resources of the WWAN links. (2) Significant performance benefits can be realized by suitable optimizations implemented at the application and session layers. Commercial web servers and browsers should implement the HTTP-pipelining scheme, which provides noticeable benefits to end-user performance. (3) Inspite of significant compressibility of web content, dynamic data compression techniques do not provide commensurate performance benefits. (4) Custom protocols, explicitly designed for WWAN environments, present significant performance benefits at the transport layer. However, in many cases the deployment of such schemes can be expensive for the service providers.

In our ongoing work, we are conducting more thorough experiments including range of other popular web-sites to obtain even more accurate web browsing benchmarks. We are also investigating other novel approaches for benchmarking application performance across realistic web server workloads and in presence of dynamically changing web content. We plan to extend this study for other WWANs e.g. UMTS, CDMA 2000.

References

1. R. Chakravorty and I. Pratt.: "Performance Issues with General Packet Radio Service", *Journal of Communications and Networks (JCN)*, Vol. 4, No. 2, December 2002.
2. P. Rodriguez and S. Mukherjee and S. Rangarajan.: "Session-level techniques to Improve Web Browsing Performance over Wide-Area Wireless Links", *Proc. of the World Wide Web (WWW) Conference, 2004* (to appear).
3. M. Meyer.: "TCP Performance over GPRS", *Proc. of IEEE WCNC 1999*.
4. R. Ludwig, et al.: "Multi-Layer Tracing of TCP over a Reliable Wireless Link", *Proc. of ACM SIGMETRICS 1999*.
5. P. Benko, et al.: "A Large-scale, Passive Analysis of End-to-End TCP Performance over GPRS", *Proc. of the IEEE INFOCOM 2004* (to appear).

Measuring BGP Pass-Through Times

Anja Feldmann¹, Hongwei Kong², Olaf Maennel¹, and Alexander Tudor³

¹ Technische Universität München, Germany {anja,olafm}@net.in.tum.de

² Agilent Labs, Beijing, China hong-wei_kong@agilent.com

³ Agilent Labs, Palo Alto, USA alex_tudor@agilent.com

Abstract. Fast routing convergence is a key requirement for services that rely on stringent QoS. Yet experience has shown that the standard inter-domain routing protocol, BGP4, takes, at times, more than one hour to converge. Previous work has focused on exploring if this stems from protocol interactions, timers, etc. In comparison only marginal attention has been payed to quantify the impact of individual router delays on the overall delay. Salient factors, such as CPU load, number of BGP peers, etc., may help explain unusually high delays and as a consequence BGP convergence times. This paper presents a methodology for studying the relationship between BGP pass-through times and a number of operationally important variables, along with some initial results. Our results suggest that while pass-through delays under normal conditions are rather small, under certain conditions, they can be a major contributing factor to slow convergence.

1 Introduction

Even though BGP has been studied extensively within the last few years (see Tim Griffin's Web page [1] of references as a starting point) we still do not have a good understanding on how and where signaling delays occur. Unexplainable large convergence times have been reported [2]. It is unclear whether they are attributable to protocol interactions [3], implementation idiosyncrasies [4], hardware limitations or other factors [5].

While one may attribute convergence times greater than 30 minutes to route flap damping [6] and those that are less than 2 – 3 minutes to multiples of the MRAI timer [7] the wide variability of in-between times is puzzling [8]. An unknown component of the delay is the contribution of each router along the path of the BGP update. A detailed exploration of the delay of each router and how it relates to factors, such as CPU load, number of BGP peers, etc., can help explain these observations.

We propose to explore pass-through times of BGP updates using a black-box testing approach in a controlled environment with appropriate instrumentation. To this end, we have setup a test framework that consists of:

Device under test (DUT): a router.

Load framework: that can be used to impose a specific, pre-defined load on the DUT. In our case it consists of several PCs and an Agilent router tester.

BGP workloads can be generated by the PCs as well as the router tester. The router tester is used to generate controlled rate data traffic. Tests are repeated with both PC as well as router tester generated BGP workloads.

Instrumentation framework: that allows us to measure not just the pass-through times of BGP updates but also the load imposed by the load framework. It consists of several packet-level monitors, periodic router queries and the router tester.

The testbed has wider applicability. For example, it can be used to explore other router measures, such as line card FIB convergence.

Our methodology goes beyond RFC compliance tests and benchmarks, e.g., IETF’s BMWG [9], in that we do not consider pass-through times in isolation. Rather, we investigate the correlation between BGP pass-through time and router load using a variety of stressors. BGP load is affected by such variables as (1) number of peers, (2) routing table size, and (3) BGP update rate. Non-BGP related tasks are many. We do not attempt to account for all of them individually. Instead, we impose a specific background traffic load which causes the `ip_input` task’s CPU usage to increase. To measure the CPU load we periodically sample the cumulative route processor’s load. With controlled experiments we can then use previously obtained CPU load data to infer the BGP portion of the CPU load. Our initial results are about establishing a baseline for pass-through times. We also explore some aspects of stress response. A stress situation arises when the imposed load reaches the limits of DUT.

Additional parameters that may effect pass-through time include the configured I/O queue length for BGP processes, ACLs’ complexity and hit ratio, BGP policy settings such as route-maps, peer-groups, filter-lists and/or communities, as well as AS prepending, etc. These are beyond the scope of this paper.

Parameter sensitivity tests and reasonable configurations were used to reduce the otherwise many experiments, given the large number of parameter value permutations.

Our experiments show that in general the DUT handles BGP stress well, which is in line with recent findings [10]. Yet the per hop BGP processing delays can be significant. Updates are only processed every 200ms even when the MRAI timer is inactive. Activating the MRAI timer adds further delay components causing higher delays occur with increasing MRAI timer values. DUT targeted traffic, even at low data rates, drastically impacts CPU load and accordingly pass-through delays. We notice that update the rate is not nearly as significant a factor for causing delays as is the number of peers. Yet high update rates occurring concurrently on multiple peers, as is happening with after router reboots, can cause problems.

The rest of the paper is structured as follows. In Section 2 we describe our methodology for measuring pass-through times and imposing a controlled router CPU load. Next, in Section 3, we describe in detail the configuration and tools that constitute our test framework. We then describe our experiments and report their results in Section 4. Section 5 concludes our paper and comments on the practical learnings of this work as applied to current operational practice and future routing protocol design.

2 Test Methodology

Three topics need detailed explanation: measuring pass-through time, separating router processing delay from MRAI timer delay, and imposing a controllable load on the DUT.

2.1 Measuring Pass-Through Times

There are three common approaches to network measurement: passively observing regular traffic, actively evaluating injecting traffic at end points within the injecting application, and passively measuring actively injected traffic. Since we operate in a testbed the first option is not applicable. Accordingly, we use specifically designed updates as active probes together with a monitoring BGP session. The timings of the probes and the resulting response updates generated by the DUT and directed to the monitoring session are passively measured using dedicated and synchronized packet capture.

Using special updates gives us the ability to impose patterns with certain characteristics, such as ensuring that the DUT will relay it to the monitoring session. The alternative approach is to replay captured BGP update traces. While this may provide useful background traffic it suffers from several shortcomings. First the pass-through time of a router depends on the settings of several BGP specific parameters such as the value of the MRAI timer. In order to distinguish the delay due to this timer from the delay due to the router we need certain update patterns which may or may not be present in regular BGP traces. Second, not all incoming BGP updates trigger an outgoing BGP update. Therefore it is hard to tell which BGP updates are discarded or are combined into other updates. Furthermore the amount of work associated with each BGP update will vary depending on its content with respect to the router's configuration.

The simplest form of a BGP update probe pattern is comprised of a single new prefix. Since the prefix is new, the update has to be propagated to all neighbors and the monitor session, policy permitting. The drawback is that the routing table size grows ad infinitum and that the available memory becomes an unintended co-variable. The table size can be controlled via explicit or implicit withdrawals. We use implicit ones since BGP withdrawal processing differs from update processing and the results would have to be separated. Implicit withdrawals on the other hand are indistinguishable from other updates. Each time a probe update for a prefix is to be sent we randomly choose an AS path length that differs from the previous path length. This ensures that the update is propagated to the monitor session and that the quality of the path improves or deteriorates with the same probability. The BGP update rate for both probes and traces can be controlled. In summary probe patters are convenient for active measurements while replaying BGP traces is appropriate for generating a realistic load on a router.

Pass-through times are not constant. They vary with the makeup of the updates and the background load caused by other factors. Accordingly, we obtain sample values by measuring the time difference between in-bound (into the

DUT) probe injections and out-bound (onto the monitoring BGP session) update propagation. To avoid time synchronization errors the packet monitors are dedicated, line rate capable, capture only cards with highly accurate, synchronized clocks. Furthermore, all other BGP sessions are terminated on the same machine, either the PC or the router tester.

Throughout this paper we use 10,000 prefixes from the 96/8 range as probe prefixes. The central part of the experiments last for 15 minutes and our probing rate is a rather low 1 update a second. This guarantees that the TCP throughput will never be problematic. We started by using 10 probing sessions but realized that interactions with a periodic timer limited the accuracy of the estimates. The periodicity of the timer in question is $200ms$ which with 10 probes per second gave us only 2 samples per timer. To increase the rate of samples per timer to 10 we increased the number of probe sessions to 50. Each probe session uses a different random offset within the second for sending its probe. This ensures that the probing is done at exponentially spaced but fixed intervals within the second. A histogram of the resulting pass-through times is plotted in Figure 1. Interestingly, the pass-through times vary from $2.4ms$ to about $200ms$ with some ranging up to $400ms$. The average pass-through time is $101ms$. The even distribution in the range of $2ms$ to $200ms$ indicates some kind of timer. Indeed, closer inspection reveals that the router limits the update processing to 5 times a second. This should result in a theoretical upper bound on the pass-through times of $200ms$. Yet some of the updates are held back for one update processing cycle. This results in pass-through time greater than $210ms$ for 1.3% of the probes.

To determine how the pass-through time compares with the one-way packet delays we also measured the one-way delay experienced by IP packets of three typical packet sizes (64, 576, 1500 bytes) that were sent during the same experiment, see Figure 1 for the 1500 byte packets. The average delays are, $0.028ms$, $0.092ms$ and $0.205ms$, hence, significantly shorter. This shows that a BGP update is delayed 11 times longer in the best case and 500 times longer on average than a simple IP packet. While this might seem significant, the total accumulated delay, at $100ms$ a hop along a 20 hop router path, would be rather small at under 2 seconds. This indicates that we have to consider additional factors.

2.2 MRAI Timer Delay

The purpose of the Min-Route Advertisement Interval timer [11] (MRAI) is to limit the number of updates for each prefix/session for a peer to one every x seconds. A typical value for x is 28 seconds. If the timer fires at time $t - x$ and t then all updates received and processed within this interval of size x are batched and sent shortly after time t . Based on this observation an upper bound for the pass-through time can be derived, even when the MRAI timer is active: for each MRAI timer interval consider those probes with minimal pass-through delay. A lower bound is derivable from the probe with largest pass-through delay within the interval. This probe arrived too late to be included in the previous MRAI timer interval.

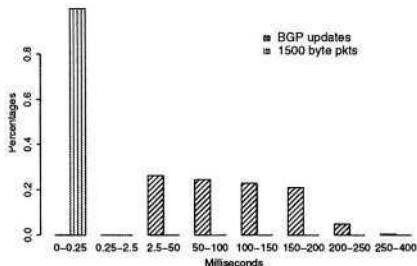


Fig. 1. Histogram of pass-through times together with one-way packet delays for typical packet sizes 64, 576, and 1500.

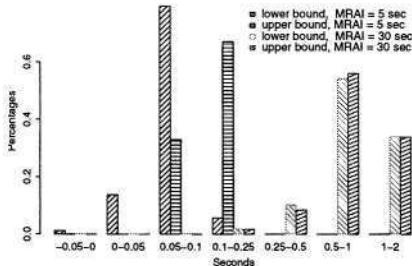


Fig. 2. Histogram of upper and lower bounds on pass-through times for MRAI values of 5 and 30 seconds.

Figure 2 shows the histogram of the pass-through times for two different MRAI timer values: 5s and the default Cisco value, which is roughly 30 seconds. Note that with the MRAI timer in place the minimal measured pass-through times are 71ms and 122ms. On the other hand the maximum values for the lower bounds are 0.121 and 1.72 seconds! This indicates that an update might have to wait a significant amount of time before it is processed even if it reaches the router at a good moment. This is especially the case for larger MRAI values where the average pass-through time increases from 109ms for the 5s MRAI timer to 883ms for the default MRAI value. Note that each experiment is run for the same duration. Accordingly the number of samples for the pass-through time decreases as the MRAI value increases.

Overall it seems that even for small MRAI values the timer interactions between MRAI and the BGP update processing timer increases the minimum pass-through time significantly. As the MRAI value is increased the minimum pass-through time also increases and will clearly dominate any link delays. Furthermore, inspection of the probes on the monitoring session reveals that the order of the update probes is not maintained. This means that a probe that was sent 10 seconds later than another might be observed earlier on the monitoring session.

2.3 Controlled Background CPU Load

Imposing a controllable background CPU load on the DUT is necessary in order to study how it responds to stress. The goal is to identify a set of tasks that generate a constant CPU load independent of BGP. This is difficult as it implies generating an input load that is uniformly served by a task running at a uniform priority. This is rarely the case. In Cisco IOS the BGP processes (and any routing tasks) have higher scheduling priority than almost everything else targeted at the CPU. The IOS ip_input task is a high priority process whose CPU use is related to the rate of a packet stream directed to the DUT's main IP address.

Another problem is measuring the CPU load. The CPU load of a Cisco router can be queried in two ways: via a command at the telnet interface or via an SNMP query. Unfortunately, the default priorities of both telnet and SNMP are lower than those of BGP and the packet processing tasks. Accordingly, for calibration only, we raised the priority of SNMP task and then measured the CPU load both via the command line as well as via SNMP for 5 rates: 2k, 5k, 10k, 15k pkt/s. Both estimates aligned quite well with the only problem that the command line interface did not deliver any values under high loads due to starvation. Figure 3 shows a histogram of the CPU load. 2k packets impose almost no load. 5k is already significant. 10k is almost critical while 15k is well beyond critical. Note that a 15k packet rate corresponds to a bit rate of 0.36 Mbits, which is rather modest for a high speed interface on a high end router. This should encourage providers to filter internal destination addresses on *all* incoming connections.

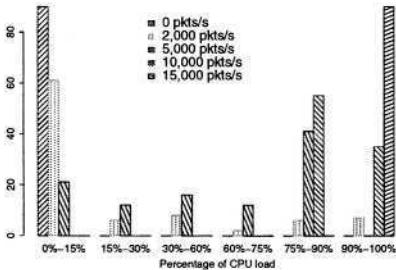


Fig. 3. Histogram of CPU load estimates for packet rates of 2k, 5k, 10k and 15k directed to the router IP.

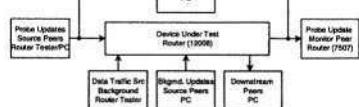


Fig. 4. Test-bed setup for router testing.

3 Test Framework

The testbed shown in Figure 4 illustrates its functional building blocks. The physical layout is more complex and not shown here for the sake clarity and brevity.

The device under test (DUT) is a Cisco 12008 GSR equipped with: 256MB memory, 512KB of L2 cache, 200MHz GRP CPU, three Gigabit SX and 8 Fast Ethernet interfaces. It runs IOS version 12.0(26)S. An Agilent RT900 router tester is used for selective experiment calibration and to generate data traffic. BGP updates, probes and background, are generated by a PC. The monitoring peer runs on a Cisco 7507. Probe update traffic from the PC into the DUT is captured by Endace DAG cards [12]. Outgoing probe update traffic from the DUT to the monitoring peer is also captured by an Endace DAG card. All cards are synchronized.

The DUT is subjected to three traffic types: BGP update probes, BGP background activity updates and non-routed data traffic directed to the DUT. Probe updates are used to compute DUT pass-through times. We create a BGP activity noise floor by generating separate update streams, called background updates, that are in turn propagated to multiple downstream peers. Data traffic is used to indirectly control the CPU load and hence the time allotted to BGP processing.

DAG generated time-stamps are used to compute the DUT pass-through time. We use tethereal to decode and reconstruct the BGP TCP sessions from the capture files. To ease the configuration and setup of each experiment various scripts automatically configure the PCs, the router tester, and the routers, then start the experiments and after it is done start the evaluation. Unless specified otherwise each experiment lasts for 15 minutes actual time but the evaluation is not started for another 15 in order to retrieve all updates.

4 Pass-Through Times

Section 2 introduces our methodology for measuring pass-through times and shows how to impose a background load on the DUT. In this section we explore how pass-through times change as the demand on the router increases. Due to the large number of parameters we cannot test all combinations. Rather, we perform a number of tests to explore the variables to which pass-through times are sensitive, including the background CPU load, the number of sessions in combination with the BGP update rate, and the complexity of the BGP table in combination with the BGP update rate.

More precisely in a first step we combine BGP pass-through probes with the background CPU load. Next we increase the CPU load by adding 100/250 additional BGP sessions and a total of 500 BGP updates a second. This experiment uses a regular pattern of updates similar to the probes. Based on this calibration of our expectation we explore the load that is imposed by actual measured BGP tables. The next two experiments differ in that one uses small BGP tables containing between 15,000 - 30,000 prefixes while the other uses large BGP tables containing between 110,000 - 130,000 updates. Due to the memory requirements of this table the number of additional sessions is reduced to 2. This provides us with a setup to explore different BGP update rates: as fast as possible (resembles BGP session resets), 200 updates and 20 updates a second.

4.1 Pass-Through Times vs. Background CPU Load

This set of experiments is designed to show how the background CPU load influences the BGP pass-through delays. Accordingly we combine the approach for measuring BGP pass-through delays via active probes with that of imposing a controlled background CPU load via a controlled packet stream directed to the DUT's IP address, see Section 2. We use a packet stream of 0, 2k, and 10k packets as the first two impose no additional or just minimal load while the latter is already almost critical.

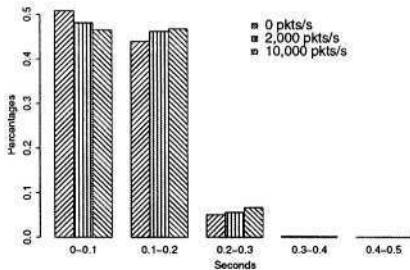


Fig. 5. Histogram of pass-through times subject to different levels of background traffic (0, 2k, 10k pkts/second).

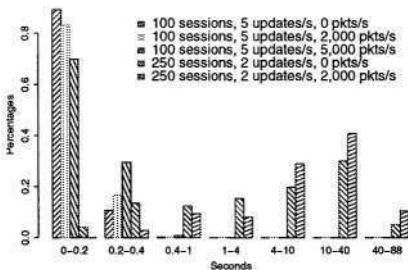


Fig. 6. Histogram of pass-through times subject to different # of sessions (100/200) and background traffic (0, 2k)).

The histogram of the resulting pass-through times is shown in Figure 5. While the differences may at first appear minor the CPU load nevertheless has an impact. It causes a delayed invocation of the BGP update processing task which is reflected in the increase of the number of updates with a pass-through time larger than 210ms. With no additional load only 1.3% of the updates are in this category. With 2k packets this increases to 2.15% and for 10k packets to 3.73%. Note that a probe rate of 50 updates a second coupled with 5 invocations of the BGP update processing task every second should create delays longer than 200ms for at most 10% of the probes. Overall we conclude that the increased CPU load delays the invocation of the BGP update processing task and therefore increases the pass-through delays. Yet, due to the timer, the delay increase is on average rather small: from 101ms to 106ms to 110ms.

4.2 Pass-Through Times vs. Number of Sessions

This set of experiments is designed to explore if the number of sessions has an impact on the BGP pass-through times. So far our load mix consisted of the active BGP probes and the packets which cause a background CPU load. Next we add additional BGP sessions (100/250) and 500 updates a second to this mix. The simplest additional sessions are similar to our probe sessions with the exception that we increase the update rates to 2 and 5 updates per second respectively.

Figure 6 shows a histogram of the resulting BGP pass-through times. Interestingly adding 100 sessions poses no significant problem to the router. Yet adding 250 sessions causes way too much load on the router even without background traffic. Note that Cisco recommends to keep the number of sessions for this specific router below 150. Adding to the 100 session experiment a background CPU load of 2k (5k) increases the CPU load from a 5 minute average of roughly 67% to 83% and then to 95%. That CPU loads are not summable is an indication for the possibility of saving some CPU by delaying the processing of the BGP updates. The additional BGP sessions increase the average pass-

through times to $116ms$. The CPU load is then responsible for the increase to $130ms$ and respectively $160ms$. The increase of the percentage of BGP probes that take longer than $200ms$ is even more dramatic: first from 1.3% to 7.4% and then with the packet load to 12.5% and 25.9%. Still the maximum pass-through times are reasonable small at less than $800ms$.

The further increase of the number of sessions to 250 causes a multitude of problematic effects. First the router is no longer capable of processing the updates so that it can send TCP acknowledgments on time. Accordingly the number of TCP retransmissions increases from almost none, less than 0.15%, to 2.5% of the BGP probe updates. Second the number of probes propagated to the monitoring sessions is drastically reduced. With 250 sessions the router does not propagate updates for 39.8% of the probes. This problem is aggravated (49.2%) by adding $2k$ packets of background traffic. While this reduces the number of samples of the pass-through times their values are now in a different class with average pass-through times of $9,987ms$ and $15,820ms$. The pass-through times increase by several orders of magnitude.

4.3 Pass-Through Times vs. BGP Table Size and Update Rate

So far all tests consisted of either probe updates or artificial patterns. Accordingly we now replace these artificial BGP sessions with actual BGP updates. For this purpose we have selected two sets of two BGP routing tables dumps from Ripe [13], one containing 37,847/15,471 entries and the other containing 128,753/113,403 entries. Note that routing tables generally differ in terms of their address space overlap, their size and their update rate. In this preliminary study we did not use actual updates or synthetic updates [14] with similar characteristics. Rather, since we want to study the impact of the BGP update rate on the pass-through times for different table sizes, we opted for some regular pattern. The first pattern, called “full-speed”, corresponds to continuous session resets and is realized by repeatedly sending the content of the BGP table to the router as fast as possible. The second pattern, called “100 updates/sec”, is similar but limits the rate of updates to 100 BGP updates a second. The third pattern, called “10 updates/sec”, further reduces the rate of updates to 10 BGP updates a second. As it is well known that session resets impose a large load on the router one may expect larger pass-through times. As one hopes that the router can perform session resets at a rate of 100 updates a second the second pattern should be simpler and not impose quite such a large load. The 10 updates a second can be expected to impose even less load than our update probes and therefore should provide us with a base line.

Figure 7 and 8 show the histograms of the pass-through times for experiments with the two small tables, Figure 7, and the two larger tables, Figure 8. As expected the pass-through times for “10 updates/sec” is with an average of $111ms$ for the small table only slightly increased. The impact of the large table is visible in its average of $127ms$. For the small table the full speed update rate is significantly higher than 100 updates/sec and imposes a CPU load of 88% to 60%. This difference in terms of update rate is not as large for the full table. Here the full patter generates a CPU load of 100% as one would hope for. For

the small table the average pass-through times increase significantly from 147ms to 181ms. If this may not seem like much there is huge danger hiding here, the one of missing BGP keep-alives. In both “100 updates/sec” experiments and the “full-speed” experiment for the large table the router did not manage to send its keep-alive in time. Therefore these experiments terminated prematurely. Overall the maximum pass-through time in the small table experiments are reasonable with a maximum of less than 710ms and only 35% greater than 210ms. For the more realistic cases with the large tables this changes. Here the maximum pass-through times increase to 2.8 seconds and the percentages larger than 210ms increases to 76%.

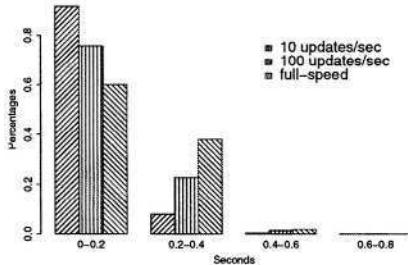


Fig. 7. Histogram of pass-through times as update rate increases (small table, 2 sessions).

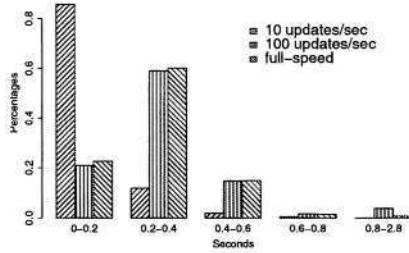


Fig. 8. Histogram of pass-through times as update rate increases (large table, 2 sessions).

5 Summary

Our results show that it is possible to determine the pass-through times using a black box testing approach. In general we found that BGP pass-through times are rather small with average delays well less than 150ms. Yet there are situations where large BGP convergence times may not just stem from protocol related parameters, such as MRAI and route flap damping. The pass-through time plays a role as well.

Even when the MRAI timer is disabled and the router is otherwise idle, periodic BGP update processing every 200ms can add as much as 400ms to the propagation time. Increasing MRAI values appear to trigger timer interactions between the periodic processing of updates and the timer itself which causes progressively larger delays. For example, when using the default MRAI timer value even the average estimate increases to 883ms but more importantly the lower bound estimation can yield values for up to 8 seconds. This indicates that there is an additional penalty for enabling MRAI beyond the MRAI delay itself. Furthermore we have observed out of order arrival of updates which suggests that

using multiple prefixes for load balancing or fail-over may not always function as expected. This bears more detailed verification.

Low packet rate data traffic targeted at the DUT can impose a critical load on the router and in extreme cases this can add several seconds to BGP processing delay. These results reinforce the importance of filtering traffic directed to the infrastructure.

As expected, increasing the update rate does have an effect on processing time, but it is not nearly as significant as adding new peers. Worth noting is that concurrent frequent updates on multiple peers may cause problems. For example, 53 peers generating 150 updates per second can cause the router to miss sending a KEEPALIVE in time, thus resulting in a session reset.

Overall the in general small pass-through times indicate that the current generation of routers may enable us to rethink some of the timer designs/artifacts in the current BGP setup. Yet care is needed to not trigger the extreme situations outlined above. Furthermore additional analysis is needed to better understand the parameters affecting BGP processing rate, such as FIB updates, line card CPU loads, BGP update contents and other configuration related parameters already mentioned above.

References

1. T. Griffin, "Interdomain Routing Links." <http://www.cambridge.intel-research.net/~tgriffin/interdomain/>.
2. C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet routing convergence," in *Proc. ACM SIGCOMM*, 2000.
3. T. G. Griffin and G. Wilfong, "An analysis of BGP convergence properties," in *Proc. ACM SIGCOMM*, 1999.
4. C. Labovitz, "Scalability of the Internet backbone routing infrastructure," in *PhD Thesis, University of Michigan*, 1999.
5. D. Wetherall, R. Mahajan, and T. Anderson, "Understanding BGP misconfigurations," in *Proc. ACM SIGCOMM*, 2002.
6. Z. M. Mao, G. Varghese, R. Govindan, and R. Katz, "Route flap damping exacerbates Internet routing convergence," in *Proc. ACM SIGCOMM*, 2002.
7. T. Griffin and B. J. Premore, "An experimental analysis of BGP convergence time," in *Proc. International Conference on Network Protocols*, 2001.
8. Z. M. Mao, R. Bush, T. Griffin, and M. Roughan, "BGP beacons," in *Proc. Internet Measurement Conference*, 2003.
9. H. Berkowitz, E. Davies, S. Hares, P. Krishnaswamy, and M. Lepp, "Terminology for benchmarking bgp device convergence in the control plane," 2003. Internet Draft ([draft-ietf-bmwg-conterm-05.txt](#)).
10. S. Agarwal, C.-N. Chuah, S. Bhattacharyya, and C. Diot, "Impact of BGP dynamics on router CPU utilization," in *Proc. Passive and Active Measurement (PAM)*, 2004.
11. Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," 1995. RFC 1771.
12. "ENDACE measurement systems." <http://www.endace.com/>.
13. RIPE's Routing Information Service Raw Data Page, <http://data.ris.ripe.net/>.
14. O. Maennel and A. Feldmann, "Realistic bgp traffic for test labs," in *Proc. ACM SIGCOMM*, 2002.

Impact of BGP Dynamics on Router CPU Utilization

Sharad Agarwal¹, Chen-Nee Chuah², Supratik Bhattacharyya³, and Christophe Diot⁴

¹ University of California, Berkeley, USA, sagarwal@cs.berkeley.edu

² University of California, Davis, USA, chuah@ece.ucdavis.edu

³ Sprint ATL, Burlingame, USA, supratik@sprintlabs.com

⁴ Intel Research, Cambridge, UK, christophe.diot@intel.com

1 Introduction

The Internet is an interconnection of separately administered networks called Autonomous Systems or ASes. To reach entities outside the AS, the inter-domain routing protocol used today is the Border Gateway Protocol or BGP [1]. It has been approximately 15 years since BGP was deployed on the Internet. The number of ASes participating in BGP has grown to over 16,000 today. However, this growth has been super-linear during the past few years [2]. With this sudden growth there has been concern in the research community about how well BGP is scaling. In particular, it has been noted that there is significant growth in the volume of BGP route announcements (or route flapping) [3] and in the number of BGP route entries in the routers of various ASes [2].

For every BGP routing update that is received by a router, several tasks need to be performed [4]. First, the appropriate RIB-in (routing information base) needs to be updated. Ingress filtering, as defined in the router's configuration, has to be applied to the route announcement. If it is not filtered out, the route undergoes the BGP route selection rules and it is compared against other routes. If it is selected, then it is added to the BGP routing table and the appropriate forwarding table entry is updated. Egress filtering then needs to be applied for every BGP peer (except the one that sent the original announcement). New BGP announcements need to be generated and then added to the appropriate RIB-out queues.

These actions can increase the load on the router CPU. Long periods of high router CPU utilization are undesirable due to two main reasons. High utilization can potentially increase the amount of time a router spends processing a routing change, thereby increasing route convergence time. High route convergence times can cause packet loss by increasing the window of time during which the route for a particular destination is unavailable. Further, high router CPU utilization can disrupt other tasks, such as other protocol processing, keep alive message processing and in extreme cases, can cause the router to crash.

In this work, we answer the question “Do BGP routing table changes cause an increase in average router CPU utilization in the Sprint IP network?”. Sprint operates a “tier-1” IP network that connects to over 2,000 other ASes. Thus, we believe that it is a suitable point for studying the impact of BGP on average router CPU utilization. There has been prior work [5,6] in analyzing BGP protocol behavior during worm propagation. To the best of our knowledge, there has been no prior published work on analyzing the relationship between BGP protocol behavior and router CPU utilization. We examine

BGP data from multiple routers in the network. We correlate this with SNMP data on CPU utilization for about 200 routers on different days inside Sprint. The findings of our work are:

- On average, BGP processes consume the majority of router CPU cycles. For short periods of time, we observe very high router CPU utilization due to BGP processes.
- We find that during normal network operation, there is some correlation between short increases in the number of BGP routing table changes and increases in average router CPU utilization and vice versa, at the time granularity of 5 minutes. However, over all the instances we observed, the impact was negligible.
- We find that during the SQL Slammer worm attack in January 2003, there was a tremendous increase in the amount of BGP changes that lasted for several hours and there was a correlation with average router CPU utilization. However, we find that the increase in utilization during this time was under 20% for almost all routers.

There are some limitations of our work:

- Since the Sprint IP network consists purely of Cisco routers, we do not measure how other router architectures react to BGP protocol behavior.
- We do not benchmark router CPU performance in a laboratory testbed. Our emphasis is on actual performance in an operational environment.
- We do not attempt to create a detailed model of how the router CPU should react to different kinds and volumes of BGP messages. Such a model will be heavily influenced by the operating system source code, which we do not have access to.

2 Analysis Data

To understand how BGP routing table changes impact average router CPU utilization, we need to analyze three kinds of data from the operational network. We now describe the routers that we access, the interactive session data, the SNMP (Simple Network Management Protocol) data and the BGP data that we analyze.

2.1 Routers

The Sprint network (AS 1239) consists of over 600 routers, all of which are Cisco routers. We had access to data from 196 routers, the majority of which are Cisco GSR 12000 series and Cisco 7500 series with VIP interfaces. They all have either about 256 MB or 512 MB of processor memory. The route processor on each of these routers is a 200 Mhz MIPS R5000 family processor. The BGP routing protocol runs as part of the operating system. On older routers such as the Cisco 1600, 2500, 4500, 4700, 7200, and 7500, there is a single processor that runs both the operating system (IOS) and packet switching [7]. On newer architectures, such as the Cisco 7500 – VIP and GSR-12000, the main processor runs IOS and interface card processors perform distributed packet switching. On these routers, the route processor runs all the routing protocols, builds the forwarding, CEF (Cisco Express Forwarding) and adjacency tables and distributes them to the line cards.

There are typically four BGP processes in Cisco IOS¹. The “BGP Open” process handles opening BGP sessions with other routers. It runs rarely. The “BGP Scanner” process checks the reachability of every route in the BGP table and performs route dampening. It will run once a minute and the size of the routing table will determine how long it takes to complete. The “BGP Router” process receives and sends announcements and calculates the best BGP path. It runs every second. The “BGP I/O” process handles the processing and queueing involved in receiving and sending BGP messages. The frequency of execution of this process will be related to the frequency of BGP updates.

2.2 Interactive Session Data

All the routers in our study allow command line interface (CLI) access via secure shell (SSH). Upon logging into the router, we can issue commands to query the state of the router. We issued the “show process cpu” command to all routers during the study. This command lists all the processes in IOS, along with the CPU utilization of each process [7]. Both the percentage utilization of the CPU (over the last 5 seconds, 1 minute and 5 minutes) and the total number of CPU milliseconds consumed since the last boot of IOS is reported. This data provides an instantaneous snapshot of what processes are currently loading the CPU and which processes have consumed the most CPU resources since boot up. A sample output is below.

```
CPU utilization for five seconds: 99%/2%; one minute: 18%; five minutes: 15%
PID Runtime(ms) Invoked uSecs 5Sec 1Min 5Min TTY Process
 1      3756 1242536      3 0.00% 0.00% 0.00% 0 Load Meter
 2      192    35       5485 0.00% 0.02% 0.03% 2 SSH Process
...
142    22517688 86537321      260 0.40% 0.39% 0.44% 0 BGP Router
143    29989784 29380359      1020 0.24% 0.69% 0.70% 0 BGP I/O
144    720698256 4712209     152943 95.48% 14.07% 11.12% 0 BGP Scanner
...
```

2.3 SNMP Data

Due to the limitation of how frequently we can collect CLI data, we also collect SNMP data. The SNMP protocol allows for a data collection machine to query certain SNMP counters on these routers and store the values in a database. We query the 1 minute exponentially-decayed moving average of the CPU busy percentage as defined in SNMP MIB 1.3.6.1.4.1.9.2.1.57. We query and store this value once every 5 minutes from each one of the 196 routers that we have access to. The network operators cited concerns about affecting the router CPU load if we were to poll this counter more frequently. This data provides us with the CPU utilization at a relatively large time scale granularity and hence does not identify the process responsible for high load. We have collected this data for as long as 2.5 years for some routers.

2.4 BGP Routing Data

In order to know if high CPU utilization is caused by a large number of BGP messages, we also analyze BGP data. We collect iBGP data from over 150 routers in the Sprint

¹ <http://www.cisco.com/warp/public/459/highcpu-bgp.html>

network, all of which we also collect SNMP data from. We collect BGP data using the GNU Zebra² routing software. We connect to about 150 routers which comprise the iBGP route reflector mesh inside the Sprint network. Each PoP announces BGP routes that originate from it to the data collector. We also connect as route reflector clients to two routers in two different PoPs.

3 Results

3.1 Short Time Scale Behavior

We first address the impact of the BGP routing protocol on router CPU utilization in short time scales. We analyze interactive session data here using the “show process cpu” command on routers in the Sprint network. This command was executed during normal network operation when no significant outage or abnormal behavior occurred. Across all 196 routers that we have access to, we see one of two cases when we execute the command. The common case is when the CPU is lightly loaded and no process is consuming a significant percentage of CPU load. In other cases, either the “BGP Scanner” or the “BGP Router” process consumes a significant percentage (sometimes over 95%) of CPU load in the 5 second average, but not in the longer term averages. This indicates that for very short time periods, BGP processes can contribute to high load on a router CPU. Aggregate statistics on how often this short time scale behavior manifests itself are difficult to produce since they are highly dependent on the collection methodology. Since polling techniques may have a lower priority than other routing processes in the operating system, aggregate results for such short time scales may be unreliable. However, for the remainder of this section, we focus on longer time scale behavior which does not suffer from this problem.

3.2 Aggregate Behavior

The main focus of this work is the impact of BGP dynamics on time scales that are long enough to have the potential to increase route convergence times and impact router stability. The first issue to address here is how much the overall contribution of the BGP processes is to the CPU cycles consumed by the router operating system.

The interactive session data shows the number of CPU cycles that each process has consumed since the router was booted. If we add the values for the three BGP processes and compare that to the sum of all the processes, we know the percentage of CPU cycles that the BGP protocol has consumed. On 20 February 2003, we collected a single snapshot of “show process cpu” data from all 196 routers that we have access to. On this day, no significant outage or abnormal network behavior occurred. Over 85% of the routers had been up for over 10 weeks when we collected the snapshots, which we consider to be long enough for a reliable reading. We calculate the percentage of CPU cycles that BGP processes consume and plot the histogram in Figure 1. We see that for the majority of routers, BGP processes consume over 60% of CPU cycles. Most of the routers below 60% were being phased out of the network and thus had few or no

² <http://www.zebra.org>

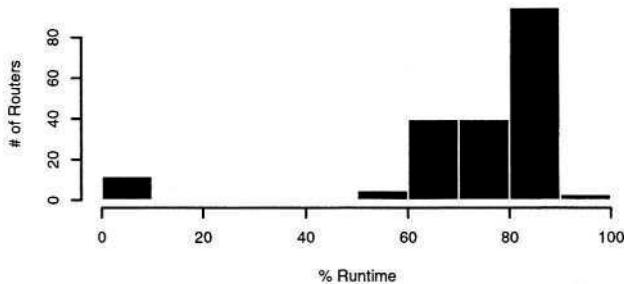


Fig. 1. CPU Utilization due to BGP Processes

BGP adjacencies. This histogram was similar across other days that we collected data on. We find that the BGP processes consume the majority of CPU cycles during router uptime. Given how much resource this protocol consumes, there is significant potential for protocol dynamics to directly impact router CPU load.

We now consider how frequently high CPU load occurs in operational routers over a 2.5 year period. During this time, a CPU load value is reported every 5 minutes via SNMP. Across the routers that we have data for over 2.5 years, roughly 0.6% of the 5 minute samples are missing. This may be due to router reboots and / or losses in SNMP data collection. We find that typically in less than 1% of these samples the CPU load was above 50%. For the vast majority of time across these routers, the CPU load was below 50%.

In Figure 2, we only consider time periods when the CPU load was above 50%. We transform the data into the number of consecutive time intervals with high CPU load and plot the cumulative sum of the sorted data. This shows that of the high CPU load occurrences, the majority of them occur for short time periods, but there are some that occur for long periods of time. This behavior is typical compared to the other routers for which we have data. These graphs are over very long periods of time, during which

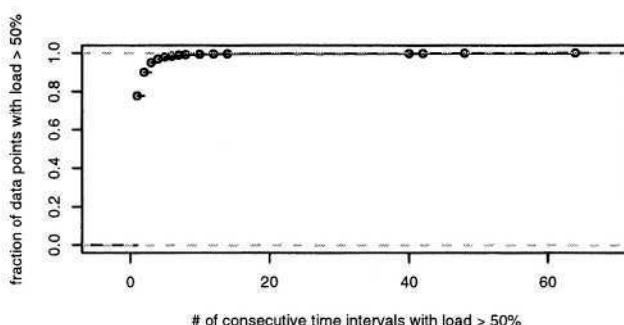


Fig. 2. Cumulative Sum of # of Time Intervals with CPU Load > 50% (Sept. 2000 to Feb. 2003)

abnormal network conditions may have occurred to cause the long durations of high load. We now examine typical network conditions and abnormal network conditions separately, and specifically consider the impact by BGP.

3.3 Typical Network Conditions

We begin by considering a typical week (10-17 February 2003) when no significant network event occurred. We examine if variations in the rate of BGP changes impact the average CPU utilization. In Figure 3, we show the number of BGP routing table changes at a router in the Sprint network. The graph is similar for other routers in the network; this one is picked arbitrarily. Each point in the graph represents the total number of changes to the BGP table during a 5 minute period. We see that on average, there are about 600 routing table changes every 5 minutes, but spikes of much higher rate of change occur. One such spike consisted of over 30,000 changes, which we denote as “Event A”. However, these spikes do not last a long time.

During this same time period, we plot the CPU load in percentage for the same router in Figure 4. Each point shows the percentage of CPU cycles that were consumed by the operating system (the remaining cycles are idle). This value is the 1 minute exponentially-decayed moving average of the CPU load and there is a point every 5 minutes. We see that the load is typically around 25%, and in one case exceeded 45% (which we denote as “Event B”).

Comparing Figure 3 to Figure 4 shows little correlation. There is very little cross correlation between the two time series over the whole week. The CCF (cross correlation function) magnitude is less than 0.1. In Figure 5, we show the cross correlation between the two time series for a two hour period around “Event A”. We see that even during this short but significant increase in the number of BGP events, there is only a small correlation with the CPU load (a maximum CCF of about 0.3). In Figure 6, we focus on “Event B” where there was a significant increase in the CPU load. While there is some correlation here (a maximum CCF of about 0.6), when we check Figure 3 around “Event B”, we do not see a very large increase compared to normal activity throughout the week.

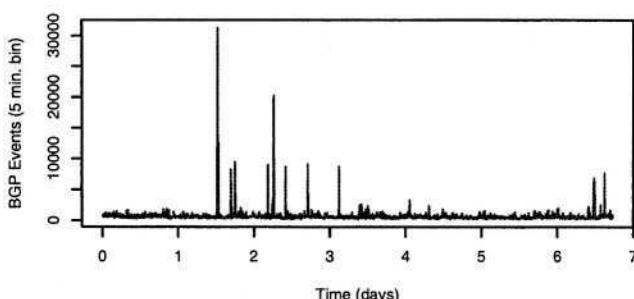


Fig. 3. iBGP Updates During a Typical Week (10-17 Feb. 2003)

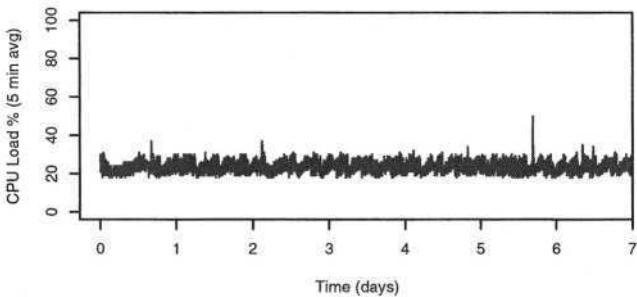


Fig. 4. CPU Load During a Typical Week (10-17 Feb. 2003)

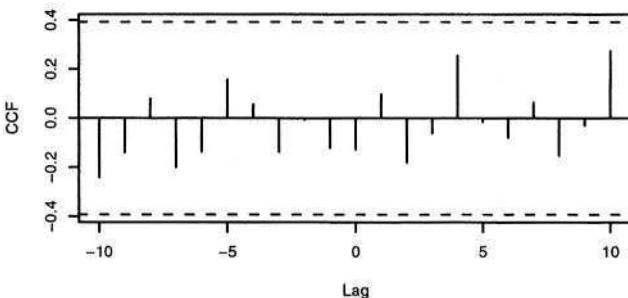


Fig. 5. CCF for 2 Hours Around Event A

This behavior we observe is typical across other routers and during other time periods. On average, the cross correlation is below 0.15. In some instances, for two hour periods around specific cases of above average CPU utilization or high BGP activity, the cross correlation is around 0.5. However, in none of these instances have we observed *both* high (significantly above average) CPU load and high BGP activity.

3.4 Abnormal Network Conditions

We now focus on abnormal network conditions. Around 05:30 UTC on 25 January 2003, the Sapphire/Slammer SQL worm attacked various end hosts on the Internet. While routers were not targeted, the additional traffic generated by the attack caused various links on the Internet to get saturated. This caused router adjacencies to be lost due to congestion, resulting in a withdrawal of BGP routes. These withdrawals propagated across the Internet. Upon withdrawal of these routes, congestion would no longer occur on these links and BGP sessions would be restored, causing BGP routes to be re-added. This cycle repeated until filters were applied to drop the attack traffic. As a result, in Figure 7, we see a significant amount of BGP traffic. While the peak value of about 20,000 route changes in a 5 minute window is not significant compared to the peaks in Figure 3, the length of time is. This large amount of activity took many hours to abate,

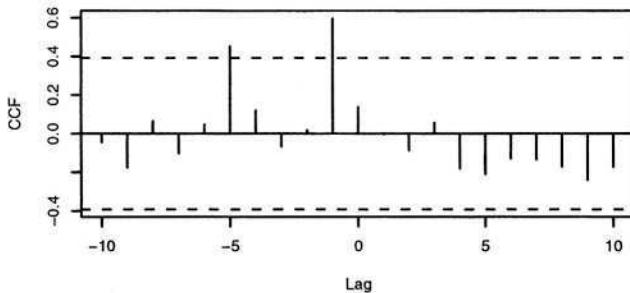


Fig. 6. CCF for 2 Hours Around Event B

instead of just a several minute spike. While this graph shows the number of changes at a particular router in the Sprint network, all routers that we have BGP data from experienced similar activity.

We correlate this time series with the CPU load percentage for the same router. Across the 65 hour period, there is a correlation of 0.5 around time lag of 0 to 5 minutes. When we focus on 12 hours before and 12 hours after the start of the attack, we see a stronger correlation of 0.6. For a period of 30 minutes before and 30 minutes after the event in Figure 8, a maximum correlation of 0.7 is observed. We plot the maximum correlation between BGP changes and CPU load across all 196 routers that we have access to as a histogram in Figure 9. We see that most routers had a correlation of over 0.5 during this abnormal event.

However, even with a strong correlation, we need to consider the magnitude of the impact on the router CPU load. In Figure 10, we show a histogram of the increase in CPU load during the 65 hour period that we consider across the 196 routers. The value that we show is the difference between the lowest CPU load and highest CPU load that each router experienced during the 3 day period. We see that despite the strong correlation, for most routers, there was less than a 20% increase in the router CPU load. In Figure 11

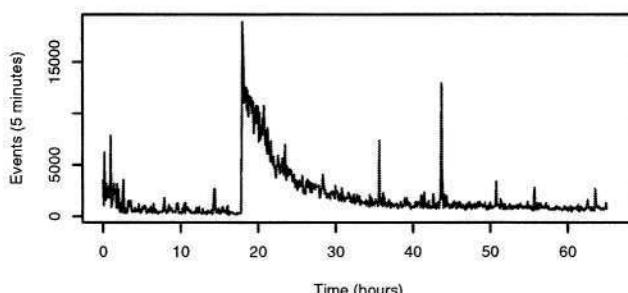


Fig. 7. iBGP Updates During the SQL Worm Attack (24-26 Jan. 2003)

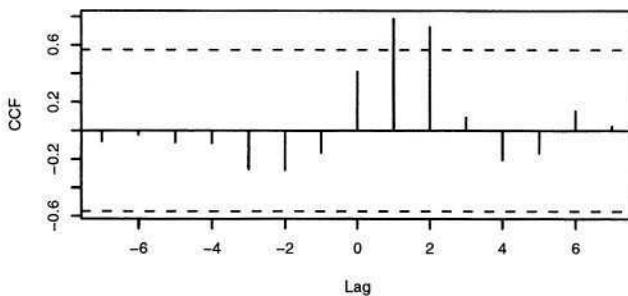


Fig. 8. CCF During the SQL Worm Attack (24-26 Jan. 2003), One Hour Around Event

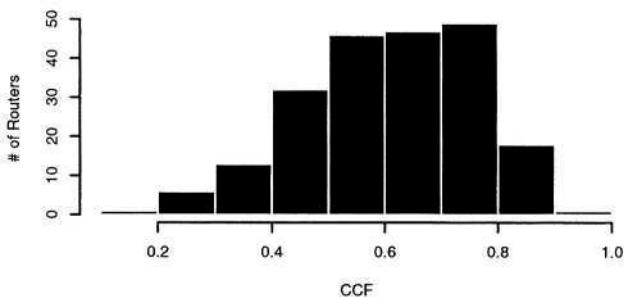


Fig. 9. Max. CCF During the SQL Worm Attack (24-26 Jan. 2003), 1 Hour Around Event

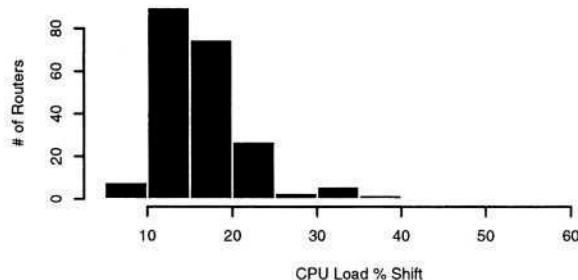


Fig. 10. CPU Utilization Increase During the SQL Worm Attack (24-26 Jan. 2003)

we show the histogram of the highest CPU utilization experienced by each router at any time during the 3 day period. We see that in most cases, the maximum load was below 50%. A few outliers above 50% exist in the data set, but manual inspection revealed that these few routers underwent scheduled maintenance during the increase in CPU load.

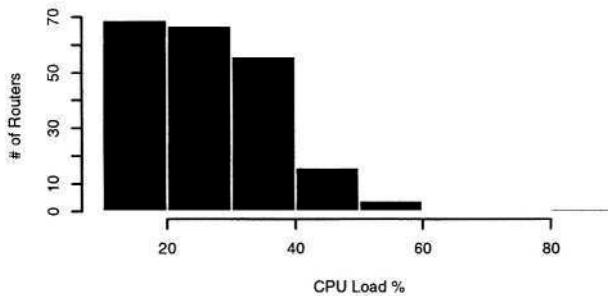


Fig. 11. Maximum CPU Utilization During the SQL Worm Attack (24-26 Jan. 2003)

4 Conclusions

Router CPU load is a significant concern in the operation of an ISP. Long periods of high CPU load can increase route convergence times, and has some correlation with router instability. The BGP routing protocol has a potential for significantly impacting CPU load due to the nature of the protocol. Several actions need to be taken upon receiving a route announcement, and on average more route announcements are seen by an ISP today than ever before. In this study, we examine the impact of BGP activity on operational routers in the Sprint IP network.

We find that on average, BGP processes tend to consume over 60% of a router's non-idle CPU cycles. During short time scales (5 seconds), we have observed BGP processes contributing almost 100% CPU load. During longer time scales (1 – 5 minutes), we see a weaker correlation. During normal network operation, we find that there is some correlation between increased BGP activity and router CPU load, but the impact is small. During an abnormal network event that lasted over 10 hours, we find a correlation. However, the increase in CPU load was under 20% for most routers.

BGP processes tend to run frequently for very short intervals, during which the CPU can reach the maximum utilization. Due to this, BGP consumes the majority of CPU cycles over a very long period of time (weeks). The quantity of BGP messages received during a particular cycle can increase the CPU load. However, this increase is not consistently large enough to cause concern about the operation of the router. A possibility is that certain kinds of BGP messages may increase the load more than others. However, without a detailed understanding of the specific implementation of BGP in these routers, we cannot speculate on such specific behavior.

Thus we conclude that during normal operation, CPU load is not significantly impacted by BGP activity in the time scale of minutes. Short term impact in the time scale of seconds is not likely to significantly impact convergence times or router stability. During abnormal events of the magnitude of the SQL Slammer worm, router CPU is not likely to increase significantly.

References

1. Stewart, J.W.: BGP4: Inter-Domain Routing in the Internet. Addison-Wesley (1998)
2. Huston, G.: Analyzing the Internet's BGP Routing Table. Cisco Internet Protocol Journal (2001)
3. Bu, T., Gao, L., Towsley, D.: On routing table growth. In: Proc. IEEE Global Internet Symposium. (2002)
4. Halabi, S., McPherson, D.: Internet Routing Architectures. Second edn. Cisco Press (2001)
5. Cowie, J., Ogielski, A., Premore, B., Yuan, Y.: Global Routing Instabilities during Code Red II and Nimda Worm Propagation. Draft paper (2001)
6. Wang,L.,Zhao,X.,Pei,D.,Bush,R.,Massey,D.,Mankin, A., Wu, S.F.,Zhang,L.: Observation and analysis of BGP behavior under stress. In: Proc. Internet Measurement Workshop. (2002)
7. Bollapragada, V., Murphy, C., White, R.: Inside Cisco IOS Software Architecture. Cisco Press (2000)

Correlating Internet Performance Changes and Route Changes to Assist in Trouble-Shooting from an End-User Perspective

Connie Logg, Jiri Navratil, and Les Cottrell

Stanford Linear Accelerator Center, 2575 Sand Hill Road, Menlo Park, CA 94025
{cal,jiri,Cottrell}@slac.Stanford.edu,

Abstract.¹ With the growth of world wide data intensive scientific collaborations, there is a need to transfer large amounts of data to and from data repositories around the world. To effectively enable such transfers, high speed, predictable networks are needed. In turn, these require performance monitoring to ensure their quality. One tool/infrastructure that has been successfully used for the analysis and monitoring of critical paths is IEPM-BW [1]. Based on experience gained from the *achievable* throughput monitoring in IEPM-BW, we developed ABwE [2], a tool to enable quick (< 1 second), low impact (40 packets) measurements of *available* bandwidth. Using ABwE we have been able to quickly detect significant changes in available bandwidth on production links up to 1Gbps, and report this information to the appropriate Network Operations Centers (NOCs) for repair, as such changes are often associated with route changes. This paper discusses this set of tools and their effectiveness together with examples of their utilization.

1 Introduction

With the growth of world wide data intensive scientific collaborations, there is a need to transfer large amounts of data to and from data repositories and collaborator sites around the world. To effectively enable such transfers, high speed, efficient, predictable networks are needed. In turn, these require continual performance monitoring to ensure optimal network performance for the applications to run. One tool/infrastructure that has been successfully used for the analysis and monitoring of critical paths (i.e. paths for which optimal performance is required) is IEPM-BW [1]. Based on experience gained from the achievable throughput monitoring in IEPM-BW, we developed ABwE [2], a tool to facilitate quick (< 1 second), low impact (40 packets) measurements of available bandwidth. Using ABwE we have been able to quickly (within minutes) visually identify significant changes in available bandwidth on production links with up to 1Gbps bottlenecks. Investigating such changes, in particular degradations, we have found, not surprisingly, that many can be associated

¹ This work is supported by U.S. DOE Contract No. DE-AC03-76SF00515.

with route changes. Once such a significant performance change is discovered, the main problem for the end-user (e.g. network administrator at an end-site) is to: gather relevant information to identify the magnitude of the change; the time(s) it occurred; identify the before and after routes; see if the change affects multiple paths; discover common points of change in the paths; identify the probable relevant Internet Service Providers; and report this information to the appropriate Network Operations Centers (NOCs). In our experience once the above has been done, the NOCs are fairly quick in responding with the cause of the change and often a fix. We have therefore developed a set of tools to facilitate the above process. The tools measure traceroutes at regular intervals, record them in an archive, and provide tools to enable simple visualization, navigation and integration with other tools such as ABwE and IEPM-BW, and a topology display. This presentation will present this set of tools and discuss their effectiveness together with examples of their utilization.

2 History and Methodology

In November 2001, a monitoring host with a 1 GE interface was set up at SLAC. Remote hosts (35-45) at collaborating sites around the world were chosen as target hosts for network performance tests. Accounts with SSH [3] access were set up on these remote hosts to provide for communicating with them. Several tools were evaluated and eventually PING, IPERF[4] (TCP/UDP transfer tool), BBFTP [5] (file transfer tool), BBCP [6] (another file transfer tool), QIPERF [7], and GridFTP [8] were selected. Currently at regular intervals PING, TCP transfers (using IPERF), file transfer tools BBFTP and GridFTP, and ABwE measurements are run. Note that the ABwE measurements are made in both the forward and reverse directions. In addition, forward *and* reverse trace-routes are run approximately every 10-12 minutes between SLAC and all the remote hosts. The results of these tests and the trace-routes are analyzed to: identify unique routes and assign route numbers to them for each remote host; identify significant route changes; and to turn the data into more useful formats (web browsable for users, text format to embed in email to Internet Service Providers, log format for debugging etc.). The data is stored in a data base with the measurement time and other test parameters for access by Web Services [9], MonALISA [10], and other visualization and analysis tools.

3 Visualization

The simplest visualization technique involves time series graphs of the ping minimum and average Round Trip Times (RTTs), the results of the achievable and available bandwidth test results, the file transfer throughputs, and indicators on the time series graphs which denote when the route to or from a node has changed. This allows for visual correlation of significant changes in RTT, available and achievable bandwidth, and route changes in one or both directions. Fig. 1 is an example of this. Fig. 1 type

graphs can be very dense, however there are options available with them to individually plot the components and to vary the time scale.

As can be seen in Fig. 1, route changes frequently do not cause throughput changes, but throughput changes often correlate with route changes. Table 1 presents a summary of visually identifiable route and throughput changes for 32 nodes monitored via IEPM-BW for the period 11/28/03 – 2/2/04. Often (see Table 1) a change in throughput is associated with a route change. This suggests that the first thing to check for when evaluating throughput changes would be a route change.

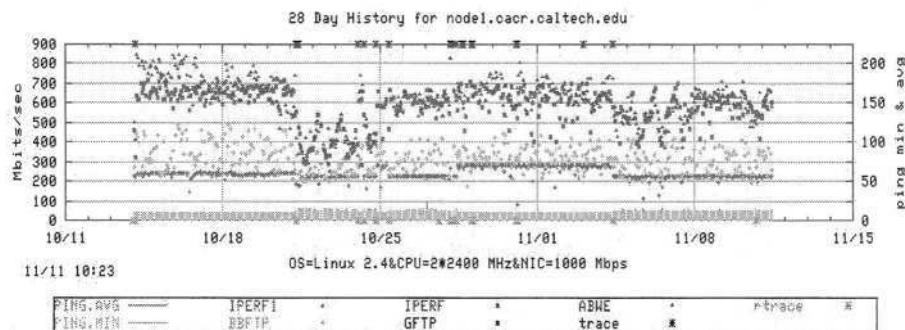


Fig. 1. Time series plot with route changes indicated. The asterisks along the top axis indicate the forward traceroute changes. The asterisks along the bottom axis indicate reverse route changes. Note the correspondence between throughput changes and forward route changes

Table 1. Summary of Route and Throughput Changes for 11/28/03 - 2/2/04

Location (# nodes)	# route changes	# with thruput increase	# with thruput decrease	# thruput changes	# thruput change with route	# thruput change w/o route
Europe(8)	370	2	4	10	6	4
Canada & U.S. (21)	1206	24	25	71	49	22 ²
Japan (3)	142	2	2	9	4	5

We also observe, not unexpectedly since many of the routes to remote hosts share subroutes, that a route change to one host often corresponds with changes in the routes to other hosts. A web accessible route daily summary page (Fig. 2) is created and updated throughout the day. At the top of the page are links to “Yesterday’s Summary”, today’s “Reverse Traceroute Summary”, and the directory containing the

² Note that 9 of these throughput changes are regular variations on Friday nights due to regularly scheduled large data transfers on one of the target networks.

historical traceroute summaries. Under those links is the traceroute summary table which provides “at a glance” visualization of traceroute change patterns. This facilitates the observation of synchronized route changes for multiple hosts in the cases that a common subroute changes for some of them.

[Yesterday's Summary](#) | [Reverse Traceroute Summary](#) | [Directory of Historical Traceroutes](#)

Checking a box for a node(s) and an hour(s) and pressing SUBMIT will provide topology maps (Fig. 3) of the selected

NODE \ Hour	00	01	02	03	04	05	06	07	08
<input type="checkbox"/> node1.caltech.edu* R Sum Log* [file]	189
<input type="checkbox"/> node1.cesnet.cz* R Sum Log* [file]	35
<input type="checkbox"/> node1.clrc.ac.uk* R Sum Log*	91
<input type="checkbox"/> node1.dl.ac.uk* R Sum Log* [file]	97
<input type="checkbox"/> node1.ece.rice.edu* R Sum Log* [file]	198
<input type="checkbox"/> node1.fnal.gov* R Sum Log* [file]	8
<input type="checkbox"/> node1.in2p3.fr* R Sum Log* [file]	31
<input type="checkbox"/> node1.indiana.edu* R Sum Log*	181
<input type="checkbox"/> node1.internet2.edu* R Sum Log* [file]	232
<input type="checkbox"/> node1.ip.apan.net* R Sum Log* [file]	189
<input type="checkbox"/> node1.kek.jp* R Sum Log* [file]	131
<input type="checkbox"/> node1.tl1.net* R Sum Log* [file]	0

Fig. 2. Screen shot of part of a traceroute summary web page with summary table

To facilitate further investigation of changes, there are highlighted links in this table that allow one to: view all the traceroutes for a selected remote host (as a color coded web table accessible by clicking on the nodename); access text suitable for attaching to trouble reports ([Sum](#)); review the log files ([LOG*](#)); review the route numbers (“[R](#)”) seen for a given host together with when last seen; view the available bandwidth time-series for the last 48 hours ([\[file\]](#)); and to select times and remote hosts for which one wishes to view topology maps.

In Fig. 2 for hours “07” and “08”, it can be seen that there were multiple route changes to European nodes in the same time frame. Each entry (there can be multiple for each box representing an hour) provides a dot to denote that the route has not changed from the previous measurement. If the route has changed, the new route number is displayed. The first measurement for each day is displayed with its route number. This very compact format enables one to visually identify if several routes changed at similar times, (i.e. route numbers appear in one or two columns for multiple hosts (rows)), and whether the changes occur at multiple times and/or revert back to the original routes.

Note that the table has check boxes before the nodes and above the hour columns at the top of the table. By checking boxes for nodes and hours, and then clicking on “SUBMIT Topology request” the viewer can generate a topographical map of the routes.

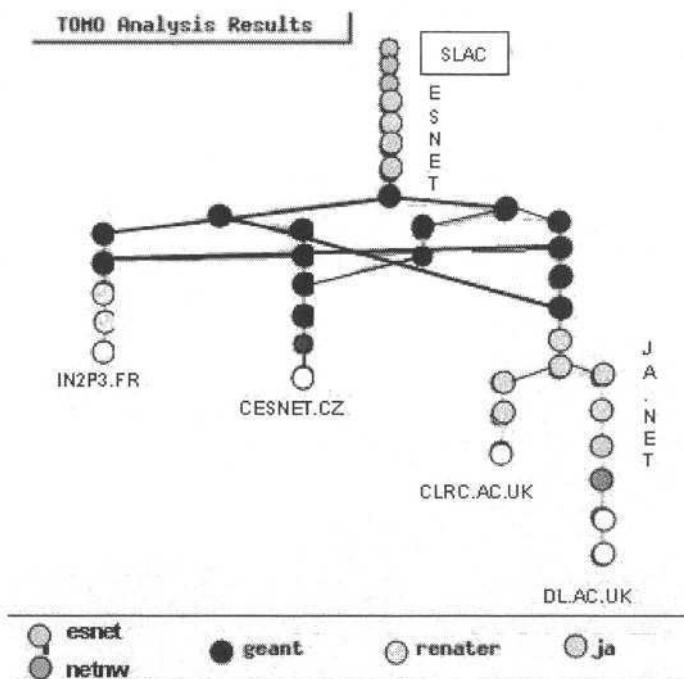


Fig. 3. Topology visualization of routes from SLAC to 4 European sites between 07:00 and 09:00 Jan 15'04

Fig. 3 is an example of such a topology map showing routes from SLAC between 07:00 and 09:00 on Jan. 15 2004 to European sites in France, the Czech Republic, and 2 sites in the UK. This corresponds to the route changes seen in Fig. 2, and the multiple routes used can be clearly seen. The topology maps display the hop routers colored by ISP, provide the router and end host names by “mouse over”, and provide the ability to zoom in to help disentangle more complex sets of routes.

4 Example of the Visualization of Achievable and Available Bandwidth Changes, and Route Changes

The measurements of achievable and available bandwidth use very different techniques. We measure achievable bandwidth via an IPERF TCP memory to memory transfer between two hosts over many seconds (usually 10 seconds). IPERF is network intensive as it sends as much TCP data as possible for the duration of the measurement. Thus we make IPERF measurements every 90 minutes. We measure available bandwidth using ABwE in less than a second by sending 20 UDP packet pairs, and measuring the time dispersal of the inter-packet delays upon arrival at the remote host. We repeat the ABwE measurements every minute. Fig. 4 is the graph of

the IPERF bandwidth data points and the ABwE measurements from SLAC to Caltech for a 24 hour period on October 9, 2003. ABwE provides estimates of the current bottleneck capacity (the top line in Fig. 4), and the cross-traffic (the bottom line in Fig. 4) and the available bandwidth (middle line). The available bandwidth is calculated by subtracting the cross-traffic from the bottleneck capacity. The IPERF measurements are displayed as black dots on the graph. The two measurements are performed independently from two different hosts at SLAC. Note the corresponding drop observed by the two bandwidth measurements at about 14:00. To understand what has happened here, it is only necessary to look at the traceroute history for this day.

Fig. 5 is a snapshot of the traceroute table for Caltech corresponding to Fig. 4. Note the change from route #105 to route #110 at 14:00, and back again to route #105 about 17:00. By selecting those hours and the Caltech node and submitting the topology request, we can get a graph (Fig. 6) of the topology change responsible for the bandwidth changes.

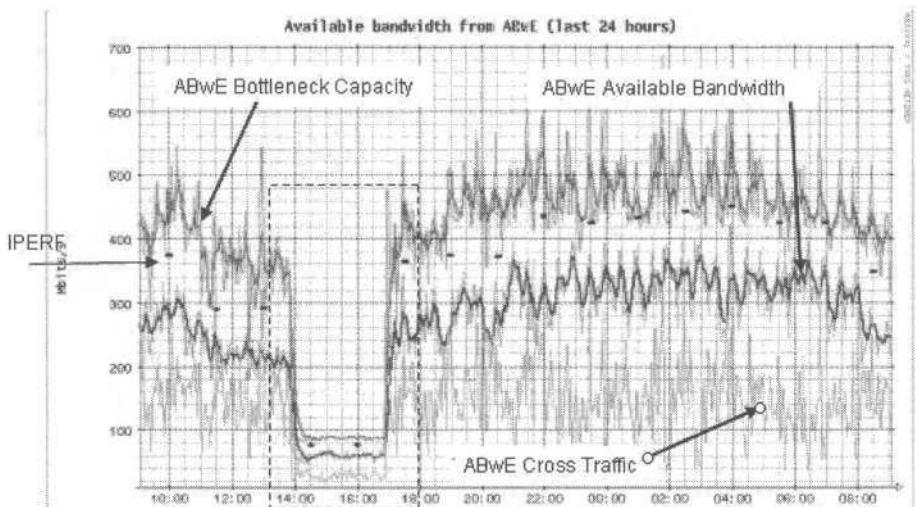


Fig. 4. Plot of ABwE (available bandwidth) measurements and corresponding IPERF (achievable bandwidth) measurements

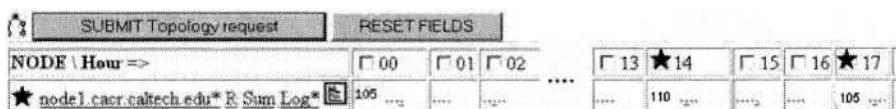


Fig. 5. Snapshot of traceroute summary entry for Caltech at 14:00 and 17:00

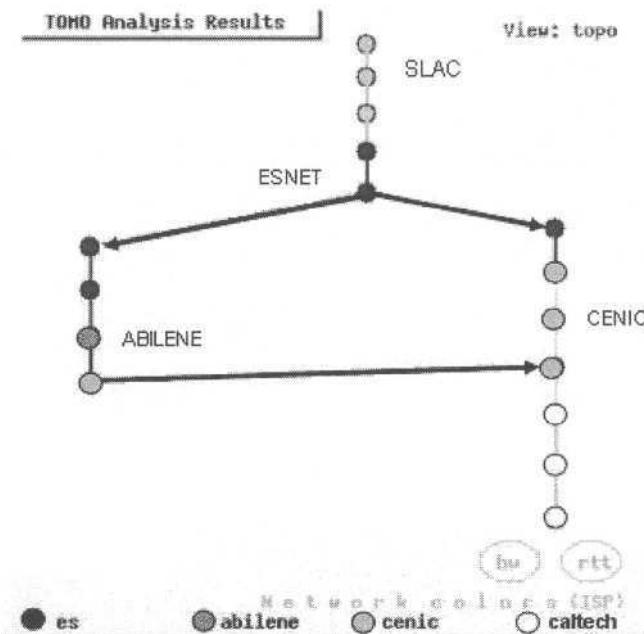


Fig. 6. Graphical traceroute display. Note hop to ABILENE on the way to CENIC

5 Challenges in Traceroute Analysis

Analyzing traceroute data to identify unique routes and to detect “significant” changes in routes can be tricky. At many hops there can be non-responsive router responses (see hop “12” in Fig. 7) one or more times, where no information is returned.

```
traceroute to NODE1-GIG.NSLABS.UFL.EDU (xx.yy.160.3)
1  SLAC-RTR1  0.164 ms
2  SLAC-RTR2  0.368 ms
3  i2-gateway.stanford.edu (192.68.191.83)  0.288 ms
4  STAN.POS.calren2.NET (171.64.1.213)  0.369 ms
5  SUNV--STAN.POS.calren2.net (198.32.249.73)  0.626 ms
6  Abilene--QSV.POS.calren2.net (198.32.249.162)  0.959 ms
7  kscyng-snvang.abilene.ucaid.edu (198.32.8.103)  36.145 ms
8  iplsng-kscyng.abilene.ucaid.edu (198.32.8.80)  53.343 ms
9  atla-iplsng.abilene.ucaid.edu (198.32.8.78)  60.448 ms
10 a713.c12008.atla.abilene.ucaid.edu (192.80.53.46)  71.304 ms
11 ssrb-ewan-gsr-g20.ns.ufl.edu (128.227.254.122)  71.323 ms
12 *
13 nslab-bpop-rsm-v222.nslabs.ufl.edu (128.227.74.130)  76.356
```

Fig. 7. Traceroute output

This can happen for a variety of reasons due to the configuration of the router at that hop and/or the type of software that it is running. There can be multiple non-responses for the same hop. There can be hop changes within a network provider's domain. In many cases these are transparent and no effect on the throughput is seen, while at other times these can be significant. Looking at the trace route output unfortunately does not solve the problem. In our processing of traceroutes to detect significant routing changes, we basically ignore non-responders. However route changes, even within a network provider's domain are considered significant. In the topology graphs responsive and non-responsive hops are displayed.

6 Challenges in Identifying Throughput Changes

Ideally we would like to automate the detection of throughput changes and compare them automatically to the traceroute changes. We have mentioned the problem with identifying significant traceroute changes. Identifying throughput changes is also tricky. One has to set "thresholds" of change to use in order to pick out throughput changes. These thresholds must vary according to the throughput level. On a gigabit link which handles high volume data transfers, a drop of 100-200 megabits (10%-40%) may simply be the result of a long/large data transfer, and may not be significant. On a 100 megabit link, a drop of 50 megabits (50%) may very well be significant, or again it may mean that there is a sustained high volume data transfer occurring.

The frequency of the data points also needs to be taken into consideration, to avoid false "alerts" which are worse than no alerts, ideally one wants to identify a "sustained" drop before alerting. If the data points are one hour apart, it may take several hours to have enough data to be sure it is a sustained drop. If the data points are once a minute, 10 minutes or slightly more may be adequate. Even with one minute samples, the identification may be difficult. In some cases we have seen the drop happen gradually over a day or two, and thus the percent change threshold is never exceeded.

7 Utilization

This set of tools has been in production use at SLAC for several months. It has already been successfully used in several problem incidents and is being enhanced as a consequence of its use. We will report on specific examples illustrating how the tools have been used to identify and pin-point performance problems in various networks. In some of these cases the problem went unidentified for several days or even weeks in one case, but once identified and reported, the problem was fixed in hours. With the large number of important collaborating sites, it is impractical to manually review all the performance graphs for all the paths and detect problems quickly. This identifies the need to automate the reliable detection of significant changes.

8 Future Plans

Work is in progress to automate the identification of significant changes, and to automatically assist in gathering the associated relevant information (e.g. current, and trace routes before and after a performance change, time and magnitude of the change, topology map, and time series plots of the performance changes). This will be gathered into email and a web page and sent to the local network administrator. We expect to report on progress with this automation by the time of the conference. We are also analyzing the ratio of significant performance problems caused by route changes and vice-versa, and the duration of significant performance degradations, and will also report on this.

References

1. *Experiences and Results from a New High Performance Network and Application Monitoring Toolkit*, Les Cottrell, Connie Logg, I-Heng Mei, SLAC-PUB-9641, published at PAM 2003, April 2003.
2. *ABwE: A Practical Approach to Available Bandwidth Estimation*, Jiri Navratil, Les Cottrell, SLAC-PUB-9622, published at PAM 2003.
3. SSH: <http://www.ssh.com/solutions/government/secureshell.html>
4. IPERF: <http://dast.nlanr.net/Projects/Iperf/>
5. BBFTP: <http://doc.in2p3.fr/bbftp/>
6. BBCP: <http://www.slac.stanford.edu/~abh/bbcp/>
7. QIPERF: http://www-iepm.slac.stanford.edu/bw/iperf_res.html
8. GridFTP: <http://www.globus.org/datagrid/gridftp.html>
9. Web Services: <http://www.w3.org/2002/ws/>
10. MonALISA: <http://monalisa.cacr.caltech.edu/>

This page intentionally left blank

Author Index

- Agarwal, Sharad 278
Ammar, Mostafa 22
- Banerjee, Suman 73, 257
Bar, Sagy 53
Bergamini, Andrea 215
Beverly, Robert 158
Bhattacharyya, Supratik 278
Biersack, Ernst W. 1, 215
Broido, Andre 113
Brownlee, Nevil 147
Brunskill, James 205
- Calyam, Prasad 137
Carson, Mark 103
Chakravorty, Rajiv 257
Chesterfield, Julian 257
Chuah, Chen-Nee 278
claffy, kc 113, 147
Claypool, Mark 227
Cottrell, Les 289
Crovella, Mark 63
- Dhamdhere, Amogh 22
Diot, Christophe 278
Dovrolis, Constantinos 93, 247
Duarte, Otto Carlos M.B. 43
- Fdida, Serge 43
Felber, P.A. 1
Feldmann, Anja 267
Fomenkov, Marina 147
- Gao, Ruomei 113
Garcés-Erice, L. 1
Glick, Madeleine 195
Goebel, Vera 215
Gonen, Mira 53
Griffin, Timothy G. 73
- Hall, Mark 205
Hamra, A. Al 1
Hohn, Nicolas 126
Hyun, Young 113
- Ibrahim, Sami 83
- Izal, M. 1
- Jain, Manish 247
James, Laura 195
Jiang, Hao 93
- Kalden, Roger 83
Karbhari, Pradnya 22
Kaur, Jasleen 33
Kinicki, Robert 227
Kong, Hongwei 267
- Li, Mingzhe 227
Logg, Connie 289
Lorier, Perry 205
- Maennel, Olaf 267
Mandrawa, Weiping 137
McGregor, Anthony 205
Mieghem, Piet Van 237
Montesino-Pouzols, Federico 175
Moore, Andrew 195
- Navratil, Jiri 289
Nguyen, Hung X. 185
Nichols, James 227
- Papagiannaki, Konstantina 126
Pias, Marcelo 73
Plagemann, Thomas 215
Prasad, Ravi 247
- Raj, Himanshu 22
Rewaskar, Sushant 33
Rezende, José F. de 43
Riley, George F. 22, 168
Rodriguez, Pablo 257
- Santay, Darrin 103
Schopis, Paul 137
Simpson Jr., Charles Robert 168
Sridharan, Mukundan 137
- Tang, Liying 63
Thiran, Patrick 185
Tolu, Giacomo 215
Tudor, Alexander 267
Tutschku, Kurt 12

- | | | | |
|-------------------------|--------|----------------|-----|
| Urvoy-Keller, Guillaume | 1, 215 | Wu, Huahui | 227 |
| Veitch, Darryl | 126 | Zegura, Ellen | 22 |
| Wessels, Duane | 147 | Zhou, Xiaoming | 237 |
| Wool, Avishai | 53 | Ziviani, Artur | 43 |