MIT Technical Report MIT-CSAIL-TR-945, April 2004

M&M: A Passive Toolkit for Measuring, Correlating, and Tracking Path Characteristics

Sachin Katti MIT CSAIL
Dina Katabi MIT CSAIL
Eddie Kohler UCLA/ICIR
Jacob Strauss MIT CSAIL

Abstract

This paper presents *M&M*, a passive measurement toolkit suitable for large-scale studies of Internet path characteristics. The `multiQ` tool uses equally-spaced mode gaps in TCP flows' packet interarrival time distributions to detect multiple bottleneck capacities and their relative order. Unlike previous tools, `multiQ` can discover up to three bottlenecks from the `tcpdump` trace of a single flow, and can work with acknowledgment as well as data interarrivals. We also describe the `mystery` tool, a simple TCP loss event, packet loss, and RTT analyzer designed to work in concert with `multiQ`. The M&M toolkit can *measure* simple path properties; *correlate* different types of measurement of the same path, producing new kinds of results; and because M&M is passive, it can use publicly-available traces to *track* the value of a measurement over multiple years.

We validate our tools in depth using the RON overlay network [4], which provides more than 400 heterogeneous Internet paths and detailed information about their characteristics. We compare `multiQ` with Nettimer and Pathrate, two other capacity measurement tools, in the first wide-area, real-world validation of capacity measurement techniques. Each tool accurately discovers minimum capacities (85% of measurements are within 10% of the true value); `multiQ` additionally discovers multiple bottlenecks and their orderings. We also use our toolkit to perform several measurement studies using a reservoir of 375 million traced packets spanning the last two years. Among the results of these studies are that bottleneck capacity on our traced links has gone up by around an order of magnitude from 2002 to 2004, and that differences in levels of statistical multiplexing on 10 Mb/s and 100 Mb/s bottleneck links result in flows over those links having similar fair-share bandwidths.

1 Introduction

A *mental model of the network* is the set of significant assumptions about the network made in the course of a piece of research. For example, we may assume that congestion only happens at the edge of the network; that the level of statistical multiplexing on the bottleneck link is low; that there is no congestion on the reverse path; that the distribution of flow sizes is heavy-tailed; and so forth. These assumptions must arise from a good understanding of the current state of the network, or how it may be expected to behave in future. Research based on erroneous assumptions has little to say about how the actual network should evolve [13].

How, then, can we create a useful description of the current Internet? The best answer is to measure those properties important for a given research question, in the widest range of expected conditions, and extract from the results any parameters you need.

But this kind of measurement presents a different set of problems than measurement for application use. For example, while an application might only care about the available bandwidth on a path, a good simulation scenario for evaluating transport protocol effects needs to know the characteristics of cross traffic and the capacities of *all* the bottleneck links (not just the tightest bottleneck). Application measurements often use active probe traffic, which becomes difficult on very large scales because of probe overhead and the need to avoid perturbing the very characteristics being measured. Furthermore, active measurements cannot be run in the past, making it difficult to see how the Internet has evolved over time. We believe, therefore, that developing a comprehensive set of *accurate, passive, trace-based* measurement tools is essential for creating more faithful representations of the Internet, and evaluating the ones we already use.

This paper presents *M&M*, a suite of passive measurement tools suitable for constructing transport-centric descriptions of the Internet. The M&M tools can extract, from passive TCP traces, broad and deep information about the capacities of multiple bottlenecks traversed by TCP flows, and the losses and RTT changes those flows experience. Combining the tools' output is easy, and can produce higher-level information about, for example, levels of statistical multiplexing—information important for transport-level mental models. We validate the tools in real-world conditions, and apply them to large, diverse traces in several example measurement studies. The tools (particularly `multiQ`) and their validation are our main contribution, but the studies also produce interesting results: for example, 10 Mb/s and 100 Mb/s bottleneck links both have significant levels of statistical multiplexing, and very similar ranges of loss rates.

The M&M suite consists of a novel passive capacity measurement tool, `multiQ`, and a multi-function TCP analyzer, `mystery`. Both tools analyze medium-to-long TCP flows contained in trace files.

`multiQ` uses packet interarrivals to investigate questions about the capacities along a path. Its basic insight is that packet interarrival times, shown as a distribution, demonstrate *equally-spaced mode gaps* caused by intervening cross traffic packets on bottleneck links in the path. `multiQ` is both passive and precise. Unlike earlier capacity-measurement work [29, 21, 8, 23, 2], it can passively discover capacities from sender-side ack packets, as well as from receiver-side data packets; and uniquely for passive tools, it can discover the capacities and relative order of up to three bottleneck links along a path.

Term	Definition
Significant flow	A TCP flow that achieves an average packet rate $>$ 10 pps ($\approx$ 1 pkt/RTT), contains at least 50 packets, and has an MTU of 1500 bytes. (The vast majority of medium-to-long data flows have this MTU.)
Bottleneck	Link where traffic faces queuing
Capacity	The maximum rate at which packets can be transmitted by a link
Narrow link	The link with the smallest capacity along a path
Tight link	The link with minimum available bandwidth along a path
Cross-traffic burst	Traffic intervening between two consecutive packets of a traced flow
Path capacity	Capacity of the narrowest link on that path

Table 1—Definitions of the terms used in this paper.

`mystery` reports loss events, lost packets, and fine-grained semi-RTT measurements throughout the length of each flow. Its techniques aren't fundamentally new, although incremental changes improve its results for difficult traces; but where previous tools have used basic measurements as a means to an end, such as the characterization of factors limiting flow performance [39, 30] (a useful and complementary approach), `mystery` concentrates on fine-grained, accurate measurements of basic properties. These measurements easily combine with each other, and with `multiQ`'s results.

Section 7, which validates `multiQ`, presents the first wide-scale Internet evaluation of recent advancements in capacity measurement. Using over 10,000 experiments on 400 heterogeneous Internet paths with known likely capacities, we evaluate `multiQ`'s accuracy and compare it with Nettimer [21], another passive capacity measurement tool, and Pathrate [11], an active tool. Our results confirm that link capacity measurement tools are mature and accurate; more than 85% of their measurements are within 10% of their correct value. With sender side traces consisting mainly of acks, `multiQ` is still correct in 70% of the estimates, and it can accurately and automatically report non-minimum-capacity bottlenecks 64% of the time. We also discover several cases where the active and passive tools detect differences in traffic limit behavior.

We close the paper with four quick, large-scale (375 million-packet) measurement studies of 258 diverse NLANR traces taken over the past two years. The M&M suite makes it easy to summarize important properties from these traces, including the distribution of bottleneck link capacities (which has increased markedly over the last two years), the levels of statistical multiplexing on bottlenecks (there is a wide range on both small- and large-capacity bottlenecks), and loss event rates for packets with different minimum-capacity bottlenecks.

Table 1 defines several important terms used throughout the paper.

2 Related Work

Much of the substantial literature on Internet measurements is complementary to our approach. Prior work, particularly on extracting properties from passive traces, would combine naturally with results from the M&M tools; we have observed that the power of a suite of tools is greater than the sum of its parts, and look forward to integrating other measurements into our framework.

Internet measurements can be divided into two classes, active and passive. Active measurements send probe traffic along a studied path to induce a network reaction that reflects the state of the path, where passive measurements extract information from packet traces or data flows that have already traversed the studied path. Active measurements are usually more powerful because the investigator can control the timing and the sending rate of the probes, but the extra load generated by probes can be undesirable, and active measurements cannot be executed on paths not controlled or accessible to the measurement tool.

Our work is particularly related to prior work on capacity measurements and tight link discovery. Capacity measurement is already a mature field with many relatively accurate tools. Currently, Nettimer [21] is the main passive tool for discovering path capacity. Our work builds on the insight gained from Nettimer, but achieves higher accuracy and can discover multiple bottleneck capacities. Further, our tool can discover bottleneck capacities from sender side traces or receiver side traces, whereas Nettimer requires the receiver side trace to achieve any accuracy. Jiang and Dovrolis [18] describe a passive method of capacity estimation based on histogram modes.

There are many active tools for measuring path capacity. Some of these tools try to find the capacities of all links along the path [29, 23]. Others, such as Pathrate, focus on the minimum capacity of a path [10]. The accuracy and the amount of generated traffic vary considerably from one tool to another. In Section 7, we evaluate `multiQ` alongside Nettimer and Pathrate.

Prior work that detects *tight* links—non-minimum-capacity bottlenecks—has all been active to our knowledge [2, 23]. There are also tools for discovering the available bandwidth along a path [15, 25, 37, 33, 34, 14], which all actively probe the network.

Shifting focus from tools to the underlying techniques, much prior work used packet inter-arrival times to estimate link capacities. Keshav proposed the concept of "Packet Pair" for use with Fair Queuing [19]. This refers to sending two back-to-back packets and computing the bottleneck capacity as the packet size divided by the pair dispersion at the receiver side. Packet pair is at the heart of many capacity and available bandwidth estimation methods, including ours.

Cross traffic can cause errors in packet pair-based capacity estimates. In particular, Paxson observed that the distribution of packet-pair capacity measurements is multi-modal [32], and Dovrolis et al [11] show that the true capacity is a local mode of the distribution, often different from its global mode. Many researchers have noted that some of the modes in the inter-arrival distribution may be created by secondary bottlenecks or post-narrow links [11, 20, 28]. Various mechanisms to filter out the cross traffic effects were proposed, such as using the minimum dispersion in a bunch of packet pairs, using the global mode in the dispersion distribution [21, 18], and using variable size packet pairs [11]. This paper complements the above prior work, but takes the opposite tactic—rather than filtering out the impact of cross traffic, we leverage the useful structure in the packet dispersion distribution created by cross-traffic to detect the capacities of multiple bottlenecks.

Prior work to `mystery` includes tools for measuring TCP characteristics such as RTT, loss rates and loss characterization. The T-RAT tool [39] is closest in spirit to our goal; it uses passive traces—sometimes more restricted than `mystery` can cope with—to classify TCP flows based on the main factors limiting their rates. `tcpanaly` [30] automatically analyzes TCP behavior from packet traces, and focuses on finding implementation anomalies. Jiang and Dovrolis [17] present a technique for passive estimation of RTTs from traces. The `tcpeval` tool for critical path analysis [6] detects various causes of transfer delay. Balakrishnan et al [5] used TCP traces at a WWW server to reproduce the evolution of several TCP state variables. Allman [3] presents algorithms for estimation of correct values of retransmission timeout settings and available bandwidths, aiming to optimize a connection's usage of the network as it begins. Lu and Li [22] present a passive half-RTT estimator exactly complementary to ours: where `mystery` matches data packets to the acks they cause, Lu and Li match acks to the data packets they liberate.

Finally, our work greatly benefits from CAIDA and NLANR's efforts to collect packet traces and analyze Internet traffic [7, 27].

3 Capacity Estimation with EMG

We begin by explaining the operation of our capacity-estimation tool, `multiQ`, and its underlying basis: the equally-spaced mode gaps (EMGs) induced by cross traffic on packet interarrival time distributions.

4 Architecture

4.1 Motivation

Current practices for detecting intrusions employ firewalls or IDS (Intrusion Detection Systems). Firewalls are choke points for network traffic whcih filter traffic according to the security policies of the organisation. IDS are passive monitors of traffic which either look for specific signatures (misuse detection) or deviations from normal traffic (anomaly detection). Snort and Bro are examples of systems which employ misuse detection. Current NIDS systems suffer from two drawbacks - high rate of false alarms and perspective from single vantage point which limits their ability to detect distributed attacks. The perspective from a single vantage point limits the ability to detect intrusions quickly and consequently increases the reaction time. Further alerts which might not be classified as malicious by a single IDS could be seen to be part of a larger distributed attack if a a global view was available.

Presently the above concerns are addressed by aggregating all alerts at a centralized location. Manual operators then go thorugh these logs to analyze and classify intrusions. Due to the volume of alerts involved and the high false positive rate, this is an ardous process resulting in a large detection time. Consequently an attack continues unabated during this time lag between commencement of attack and detection by which time it might have caused unrecoverable damage. For example Shannon and Moore (Infocom paper) show that reaction times on the order of 30 minutes are necessary to contain worm outbreaks. Clearly there is a need for an effective, automated and real-time system which can detect attacks and initiate appropriate defence strategies.

A promising approach to alleviate the above shortcomings is to correlation of alerts from different IDS systems i.e. a Distributed Intrusion Detection System (DIDS). In this envirnoment IDS systems exchange information with each other offering the tantalizing benefit of near global knowledege at the cost of communicating with peers. However several critical problems need to be adressed before intrusion detection information can be safely distributed among cooperating peers. Exchanging alert data in a full mesh with all participating peers is prohibitively expensive given the sheer volume from each IDS. Consequently scheduling algorithms to limit the group size to a managaeable size are necessary. Even among small groups there is a need to be economical in the amount of information exchanged. Clearly exchanging offending packet payloads or high volume signature information is expensive. Hence compressed yet reasonably descriptive means of communication are imperative. Another important concern is that of privacy - if we have IDS systems from different organisations exchanging information, mechanisms for safely exhanging sensitive data without compromising the cooperating peers need to be evolved.

The assumptions we make about the environment we operate in are important considerations. Our motivation to build such a system came out of the difficulties experienced in using a deployed IDS system in a Tier 1 ISP. These IDS sensors were placed at different locations in the network, ranging from borders with other ISPs to internal assets. Each IDS system was autonomous and reported all iots alerts to a centralized system where the detection was done manually. The sheer volume of the alerts generated at each sensor was overhelming (*i will put some numbers here*), which resulted in delays in detection and consequently containment and defence. The obvious benefits of distributed analyses and consequent reduction in the number of false positives were the main motivations for this system. Hence we envision an architecture where a ste of IDSes are deployed over a large ISP, which communicate with each other. Since all the IDS sensors belong to the same organisation, we do not consider issues of privacy in this paper.

4.2 Our Approach

A distributed system for scalable intrusion detection is not unlike other large distributed information sharing systems. Hence it must satisfy the same set of requirements:

- Availability - the detection system must be resilient to failures as well as attacks againt the system itself
- Scalability - The system should be scalable as the number of sensors in the system grows.

- Distributed Failure modes - There should not ne single points of failures in the system and responsibilities and consequently vulnerabilities should be distributed

- Heterogeneity - A distributed IDS system must be able to include different IDS systems

4.3 Scalable Correlation Mechanisms

In building a scalable technique, the first major hurdle to be passed is the communication cost. Clearly every IDS system cant talk to every other sensor, the resulting communication overhead would be unmanageable. Also the benefits of such a system are questionable - the time taken to communicate with everyone and evolve a consensus on a suspected intrusion would be comparable to manual detection. Hence we take a step back and ask the question - Is it necessary for all the sensors to talk to each other for detecting an intrusion?

Sophisticated approaches to the above problem take two differnt tacks. The first approach creates mappings from particular alerts (IP addresses, type of alerts, ports) to specific sensors. These sensors are then responsible for those kind of attacks. With all alert data of a a particular nature being collected at one node, all processing can be done there without the need for communicating with other nodes. But nodes become a special case of the centralized model: they are single points of failure corresponding to the IP address being hashed. Collecting raw information might also not be attractive due to issues of trust, nodes might be uncomfortable to place raw alert information at a single node.

Essentially what is necessary is a way to generate optimal groups of sensors which need to communicate with each other when all of them are seeing a particular attack. This optimal schedule should allow any node to correctly guess which other subset of sensors it should talk to in order to ascertain the nature of an attack. In order to mimic this behavior the (approximately) correct nodes must talk to each other at (approximately) the right time. One way of accomplishing this is to pick groups at random and change groups at a fast rate. Howvere a more robust model than the random approach provides some control over the rate at which groups are formed and change. We evolve such an algorithm and explain it in the following paragraphs.

Consider a typical distributed attack - a distributed denial of service attack. DDOS attacks have proliferated over the past few years and the evidence [cite CERT warnings] is that many more are on the way. Staging such an attack requires several steps - an attacker compromises a number of machines, using well known software vulnerabilities. Backdoors for later access are installed in each such machine. Sophisticated automated tools are available which make it very simple for an attacker to achieve the above. Finally an attacker configures all the compromised machines to launch a targeted attack at a specifc resource over the Internet. The resource could be a popular website, a service (ex DNS) or the routing fabric of the Internet. The compromised machines are organised in a hierachy to hide the identity of the true attacker. Spoofing source addresses is a common technique to prevent traceback.

But any distributed attack contains some common properties which offer points of localisation for a detection system. For the DDOS attack above, if a particular website is being targeted, the destination subnet is a good starting point for commonality among alerts. It is likely that many sensors are going to generate alerts which correspond to the same destination subnet when such an attack takes place, given the distributed nature of the attack. It is also likely that sensors close to the destination subnet will be seeing more of the attack traffic than other sensors. Hence if a sensor detects suspicious activity aimed at a particular destination subnet, at a good first approximation forming a group consisting of sensors which are going to see traffic to the particular destination will help in detecting the veracity of its suspicions.

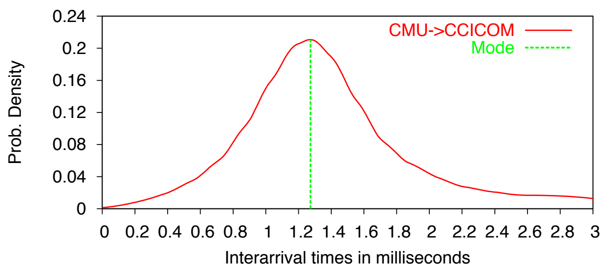
The essential idea hidden above is that of using hints in the attack for determining the set of sensors which could possibly be seeing the same attack. The hints are properties which will be common to all alerts corresponding to the same attack. In the above DDOS example, it was the destination subnet being attacked. If a particular resource in the Internet is being attacked, it would be the commonality among the resource being attacked (for example DNS servers). During the early parts of a worm outbreak it is the identity of the source subnets which will be common to all the alerts being generated. If a worm has reached a significant level of contamination, the signature of thw worm will be the common property across sensors. In Section evaluation we show that there are many dimensions along which an attack shares properties across sensors.

4.4 Taxonomy

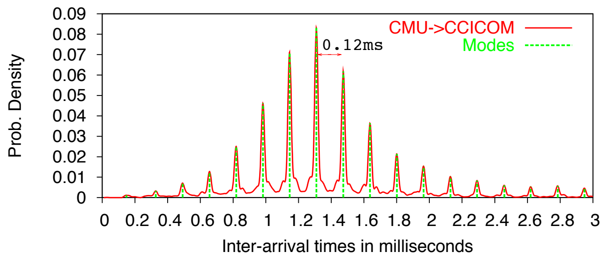
For a sensor to construct the set of sensors which are likely to see the same kind of attack without talking to them, it needs information about every other IDS in the system. This infomration should contain a sketch of the nature of traffic each sensor has encountered in the past and expects to in the future. This implies a certain assumption of stationarity in the nature of attacks at each sensor which we show to be true in the susequent section. Hence every IDS contains a profile of every other sensor in the system. The profile is a summary of the source subnets a sensor sees attacks from, the destination subnets it protects and also sees attacks for, the nature of signatures of attacks it encounters, the traffic profile classified according to port, protocol etc. When a sensor sees an alert, it identifies the property and consults the profiles of other sensors to determine the group of sensors each IDS stores a profile of every other IDS in the system.

Clearly the property along which to localise depends on the nature of the alert. For the DDOS example above it was the destination subnet, for a worm it would be the source subnet or the signature and so on. This underlines the need for classifying alerts into a taxonomy which lets us localise the property which is likely to be common among the distributed set of alerts. The essential intuition is for each sensor to classify an alert according to the taxonomy, it picks the property which is likely to be common among alerts of that kind and consult the subset of sensors which are known to see traffic satisfying the particular property of that class of alerts.

The taxonomy above is intended to be a starting point and extendable and hence not comprehensive. Each leaf node in the



(a) Main mode at around 1.2ms shows the 10Mb/s CMU link



(b) Gaps of 0.12ms show the 100Mb/s CCICOM link

Figure 1—The data from Figure 4b at two different resolutions.

tree above corresponds to a distinct type of attack for which a unique common property can be found in all alerts belonging to that specific attack in progress. The sensor classifies every alert it deems to be suspicious according to the taxonomy and then identifies the set of sensors whose profiles indicate the likelihood of seeing the same attack. These sensors then communicate with each other and exchange alert information to determine the validity of their suspicions.

4.5 Communication Mechanisms

Alerts grow rapidly given the substantial traffic on each sensor, hence even among small groups it is necessary to encode and represent alerts in a compact manner. Drawing from lessons learned in the deployment of web proxies we propose Bloom filters as efficient mechanisms to exchange compact information about alerts. Bloom filters are compact bit vectors and serve as probabilistic data structures. Entries are indexed by k independent hashes of the data (in our case the alert information, which could be IP addresses, ports, alert types etc). The bit index pointed toby the hash opf the data is set to 1. This is repeated multiple times for resiliency and redundancy. In order to determine membership of a particular data item the same process is executed, the data item is hashed and the corresponding indices are checked. If those bits are set, it implies membership.

Bloom filters therefore offer the following advantages

- Efficient representation of alert information. A Bloom filter around 10k bits in size is still able to verify tens of thousands of entries

- The tradeoff is the amount of compression and the false positive ratio, but Bloom filters do not create false negatives. The false positive ratio can be reduced by appropriate tuning and using redundancy by multiple alert lists.

```
 1. Compute flow interarrivals from trace file
 2. Set $scale$ := 10 $\mu$s
 3. While $scale < 10{,}000\mu$s:
 4.     Compute kernel PDF estimate with width = $scale$
 5.     Find the modes
 6.     If there's only one mode, at $M$:
 7.         Output a capacity of (1500*8/$M$) Mb/s
 8.         Exit
 9.     Compute the mode gaps
10.     Compute the PDF of the gaps
11.     Set $G$ := the tallest mode in the gap PDF
12.     If the probability in $G > 0.5$:
13.         Output a capacity of (1500*8/$G$) Mb/s
14.     Increment $scale$
```

Figure 2—Pseudocode for `multiQ`.

5 MultiQ: Automating EMG

The `multiQ` passive bottleneck detection tool automates the EMG capacity detection technique. It takes as input a `tcpdump` trace, and automatically discovers and estimates the capacity of the bottlenecks traversed by particular flows specified by the user.

Automating multiple bottleneck discovery is tricky because it requires interpreting the visual image of the interarrival PDF to extract the relevant information and ignore the noise. To do this, `multiQ` analyzes the interarrival PDF at a *progression of resolutions* corresponding to a known set of common link speeds. To demonstrate this, Figure 1 plots the CMU-to-CCICOM data from Figure **??**b at two different resolutions. At the lower resolution, we see one large mode in the distribution, which corresponds to the upstream lower-capacity bottleneck. As we increase the resolution, the large mode becomes fractured into smaller spikes corresponding to the higher-capacity bottleneck. The envelope traced by the peaks of the smaller spikes follows the original broader mode.

The procedure works as follows. At each resolution, starting with the highest resolution, `multiQ` constructs a kernel density estimate of the PDF and scans it for statistically-significant modes. The gaps between these modes are computed. Then, `multiQ` finds the probability distribution of the gaps themselves. A mode in the gap's PDF corresponds to a highly repeated gap length—the hallmark of a congested link. If `multiQ` finds a significantly dominant mode in the gap distribution at the current resolution, it decides that mode represents the transmission time of 1500 bytes on some bottleneck, and outputs that bottleneck's capacity. If there is no dominant gap at the current resolution, `multiQ` decreases the resolution and repeats the procedure. Figure 2 shows this procedure in pseudocode.

A few details are worth discussing. First, since we are looking at the interarrival PDF at different resolutions, we need to use a kernel PDF estimator to detect the modes—the flat bins of a histogram would prevent precise mode estimation. Second, modes are identified as local maxima in the density estimate that have statistically significant dips.[1] Finally, when `multiQ` analyzes ack inter-arrival PDFs, it uses a slightly different pro-

[1]A significant dip [35] is defined as one in which the dips on either side of a local maximum drop by more than the standard deviation of the kernel density

cedure to deal with the first mode in the PDF: a large spike close to zero is a sign of compressed acks and should be ignored, whereas a spike located at twice as much as the repeated gap in the PDF is a sign of delayed acks and corresponds to the transmission time of 3000 bytes on the bottleneck link.

5.1 Limitations

EMG estimation is more robust on receiver-side data-packet traces than sender-side ack traces. When run on ack traces, the current version of `multiQ` does not try to discover bottlenecks whose capacity is higher than 155 Mb/s.

Our method relies on the cross-traffic burst structure, which depends on the packet size distribution. If 1500 bytes stops being the dominant large-packet mode, our technique will fail. Fortunately, this distribution appears to be changing towards further emphasis of the 40-byte and 1500-byte modes; for instance, compare the 1998 and 2001 packet size distributions in Claffy's papers [9, 36].

6 MYSTERY

The `mystery` tool investigates the network characteristics of loss event rate, packet loss rate, and RTT variability. The loss event detector works at either the sender or receiver side, and only requires access to the data packets. The lost packet detector and the ack correspondence detector (which measures RTT variability) are designed for the sender side—they work at the receiver side, but produce uninteresting results—and require access to both data and acks. These techniques are not fundamentally new; loss event detection, for example, goes back at least to `tcpanaly` [30]. `mystery` differs from earlier work in the granularity of its results. Other tools report anomalies or broadly classify TCP flow behavior [39]; mistakes in fine-grained measurements, such as RTTs, may be acceptable as long they don't affect the broad result. `mystery` complements this work by providing good-quality raw data, such as ack correspondences. It doesn't, however, contain any of the higher-level intelligence built into the other tools.

`mystery` operates on `tcpdump`, NLANR, or other format traces containing one or more TCP flows. Its output is in XML format. Section 7.7 presents a validation.

6.1 Loss Events

The loss event detector reports all loss events in the trace, where a loss event begins with a lost packet and ends when the sender retransmits that packet. A loss event may contain more than one lost packet; modern TCP implementations halve their congestion windows once per loss event, rather than once per packet loss. `mystery`'s loss event detector behaves similarly to those in T-RAT and other tools [39, 16]: It detects a new loss event every time it sees a reordered or retransmitted packet whose original transmission was not part of a previous loss event.

estimate at the local maximum. The standard deviation is given by

$$StdDev(g(x)) = \sqrt{g(x) \times R(K)/nh}, \quad (1)$$

where $g(x)$ is the estimate at point x, $R(K)$ is the roughness of the kernel function, n is the number of points, and h is the kernel's width.

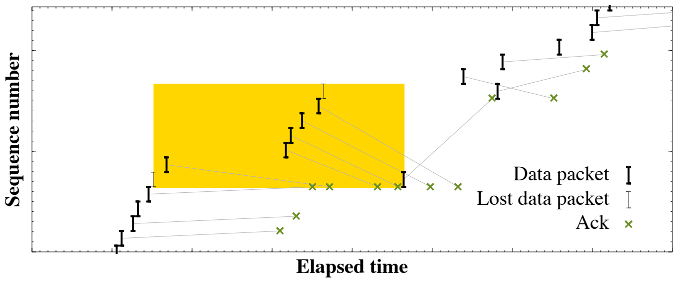


Figure 3—Time-sequence plot showing a loss event (shaded box), lost packets (thin I-beams), and ack correspondences (lines between data packets and acks).

An incremental improvement in `mystery`'s loss event detector is the use of ack timing to distinguish false retransmissions from true loss events. A loss event is *false* if the original "lost" packet was actually received. To our knowledge, previous tools detect a false retransmission when the relevant ack arrives strictly before the retransmission. `mystery` takes the flow's *minimum ack delay* into account. The min-ack-delay equals the minimum time difference between any data packet in the trace and its corresponding acknowledgment; a loss event is false if the delay between the retransmission and the ack is much smaller than this. (Since min-ack-delay measures the minimum time it takes for an ack to arrive, any ack sent quicker than this must have corresponded to the original "lost" packet.)

The loss event detector cannot detect events all of whose packets were dropped upstream of the trace point, and false loss events can only be distinguished when acks are available in the trace. In our validation experiment (Section 7.7), `mystery` finds 5776 loss events in 155 traces, 99 of which are labeled false. Manual trace examination indicates the main causes of false loss events are reordering, bad RTT estimates, and confusion caused by earlier loss events.

6.2 Lost Packets

The lost packet detector uses ack information to decide which packets in a loss event were actually lost. Aside from its independent interest, we found lost packet detection necessary to obtain good ack correspondences.

The lost packet detector again uses the obvious algorithm plus some incremental improvements. It is based on TCP's cumulative ack, which indicates the delivery of every preceding sequence number. When a new ack a arrives, `mystery` moves backwards over the data packets. Each data packet p with last sequence number $\leq a$ is marked *unless* other packets covering p's sequence numbers have already been marked. Once the whole trace is processed, any unmarked packets are identified as lost. An improvement is to avoid marking packets that must have arrived after the ack was sent, again using min-ack-delay. We also needed special handling for TCPs that don't implement Fast Recovery: two or more candidates covering the same sequence numbers may need to be marked.

This algorithm behaves independently of the number of duplicate acks. One might expect us to count duplicates instead, since each dup-ack generally indicates that another packet has been received; but reordering, interference from prior retransmissions, and lost acks make it more robust to ignore duplicate acks. SACK and DSACK information would be valuable, and if these options were ubiquitous, the loss detector would become trivial.

The lost packet detector can incorrectly identify packets as lost if the RTT grows significantly over the connection's lifetime, or if acks are dropped.

6.3 Ack Correspondence

The ack correspondence tool generates a mapping AC from ack packets to data packets, where $AC(a)$ equals the data packet that caused a to be sent. The last sequence number on $AC(a)$ will not equal a's ack number if there was loss or reordering. Ack correspondence is complementary to, but easier than, data correspondence [22], which determines the data packets that were liberated by each ack. An ack correspondence mapping expresses properties of the TCP session, such as whether the receiver delays acks; but we're mainly interested in it for sender-side traces, where a complete mapping provides fine-grained measurements of the round-trip time throughout the connection's life. Existing passive RTT measurements look mostly at the initial portion of the connection [17].

Given an ack packet a, the ack correspondence algorithm chooses as $AC(a)$ the earliest data packet that could plausibly work. Heuristics used to determine plausibility include:

- The delay between $AC(a)$ and a must be at least $0.8 \times$ min-ack-delay.
- $AC(a)$ cannot be a lost packet (we use the packet loss detector here), and no data packet corresponds to more than one ack.
- Keep track of ack-highwater, the maximum sequence number that we believe was received. If a acknowledges more than ack-highwater, then a was not sent in response to a retransmission, and $AC(a)$'s last sequence number should equal a's ack number.
- If a isn't a duplicate, then it was sent in response to new data at the top of the window, or a successful retransmission. In either case, $AC(a)$'s last sequence number can't be greater than a's ack number.
- If a is a duplicate, then there was loss or reordering. Skip any data packet whose initial sequence number equals a's ack number.

Section 7.7 validates the ack correspondence detector on 155 diverse traces.

7 VALIDATION

We evaluate the accuracy of `multiQ` using 10,000 experiments over 400 diverse Internet paths from the RON overlay network, and compare it both with known topology information and with two other capacity measurement tools, Pathrate and Nettimer. Our results show the following:

- When measuring minimum-capacity bottlenecks, `multiQ` is as accurate as Pathrate, an active measurement tool; 85% of its measurements are within 10% of the true value. Nettimer is equally accurate if operated with both sender and receiver traces, but its accuracy goes down to 74% with only receiver side traces and 10% with only sender side traces.

- On sender side traces, which consist mainly of acks, 70% of `multiQ`'s measurements are within 20% of their correct value.

- As for tight links (i.e. non-minimum capacity links), `multiQ` automatically detects 64% of them, misses 21% (though a human could detect them visually on an interarrival PDF), and mislabels 15%.

- The average error of both `multiQ` and Nettimer is highly independent of flow size for flows larger than 50 packets.

- We also validate `mystery` using 155 diverse paths from RON. When run at the sender side (the hard case), its error rate for lost packets is under 1% for more than 80% of the paths we tested, and under 10% for all paths. Ack correspondence is slightly less reliable.

7.1 Experimental Methodology

Ideally, we would like to have information about all the capacities and loss rates along a large number of heterogeneous paths that form a representative cross section of the network. This is inherently difficult on the Internet, of course, but we have tried to evaluate our tools on as representative a network as possible. We use the RON overlay network [4], whose 22 geographically-distributed nodes have a diverse set of access links, ranging from DSL to 100 Mb/s connections,[2] and ISPs on both the commercial Internet and Internet2. RON has 462 heterogeneous paths, 25% of which use Internet2. We therefore have good reason to believe that these paths' characteristics are representative of what we would encounter on the Internet.

We compare the capacity tools' estimates for each RON path against that path's "true" bottleneck capacity. A fair amount of legwork was required to determine these values. We contacted each node's hosting site and obtained a list of all their access links and the capacities of the local networks to which the nodes are connected. For multi-homed nodes, we learned the access capacities of each upstream ISP. RON nodes not on Internet2 have low-speed access links ranging from DSL to 10 Mb/s; paths terminating at one of these nodes are unlikely to encounter a lower-capacity link on the Internet backbone. For RON nodes in Internet2, we additionally obtained information about *all* Internet2 links on the relevant paths. On top of this, we used a wealth of information obtained from the RON overlay operator about path characteristics over the last 3 years.

To verify the consistency of these "true" capacities, we ran all three capacity measurement tools and a number of `ttcp` and UDP flows of varying rates on each path. If a path's results pointed out an inconsistency—for example, if `ttcp` or UDP obtained more bandwidth than the "true" capacity—then we eliminated the path from our experiments. Only 57 out of a total of 462 paths needed to be eliminated.

[2] 9 nodes have 100 Mb/s uplinks, 6 have 10 Mb/s, 3 have T1, and 4 have DSL.

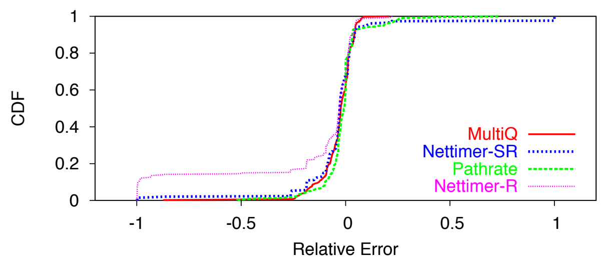


Figure 4—Comparison of the accuracy of MultiQ, Nettimer and Pathrate. Graphs show the CDF of the relative error.

Source	Destination	Capacity estimate (Mb/s)		
		multiQ	**Nettimer**	**Pathrate**
jfk1-gblx	speakeasy	1.354	1.366	99
nyu		1.353	1.361	98.5
cornell		1.392	1.358	9.55
gr		1.354	1.362	99.5
cmu		1.354	1.36	9.65
jfk1-gblx	cybermesa	10.519	11.89	.998
nyu		10.563	10.514	.9985
cornell		8.134	8.1	.997
gr		8.134	8.139	.9985
cmu		8.13	8.121	.996

Table 2—Estimate differences between Pathrate and the other tools (see § 7.3).

7.2 Timestamp Errors

An important source of possible error is the timestamps we get from `tcpdump`. Our tools work on single passive traces, so we don't need to worry about calibrating timestamps from multiple sites [31]; only errors in time *differences* are relevant. These errors may arise from fluctuations in the time it takes to go from an on-the-wire packet delivery to the network interrupt handler, which timestamps the packet on `tcpdump`'s behalf.

We analyzed a data set that contains both DAG hardware timestamps and `tcpdump` timestamps collected at RIPE [38]. Although `tcpdump` timestamps can differ from DAG hardware timestamps by 20 μs, the errors in the timestamps of consecutive packets are highly correlated. Hence, compared to interarrival times calculated from the DAG timestamps, the errors in interarrivals of successive packets computed from `tcpdump` timestamps are only a few μs. Such small errors should not affect our results.

7.3 Minimum Capacity Estimation

We now turn to an evaluation of `multiQ`'s minimum capacity estimation. We compute the relative error of `multiQ`'s estimates compared with the "true" minimum capacities, and compare that relative error with two other capacity measurement tools—Pathrate, which is active, and Nettimer, which is passive. We find that `multiQ` is very precise.

We tried to ensure that the three tools encountered the same path characteristics, such as loss rate and delay, by running the tools immediately after one another on each path. We first conduct a 2 minute run of `ttcp` and collect traces at both endpoints. These traces serve as data sets for `multiQ` and Nettimer. Immediately thereafter, we run Pathrate on the same path and compute its estimate; we use the average of Pathrate's high and low estimates. This procedure is repeated five times, and we report the average of those 5 trials. Finally, the same set of experiments is run both at day and night, to compensate for any traffic fluctuations due to the time of the day. In total, we performed more than 10000 experiments.

We plot the *relative error* ξ for each capacity estimate C_e, which is defined as

$$\xi = \frac{C_e - C_t}{C_t} \qquad (2)$$

where C_t is the path's "true" capacity.

Figure 4 shows the cumulative distribution function (CDF) of the relative errors of `multiQ`, Nettimer, and Pathrate estimates on RON's 405 paths. Nettimer has two lines: Nettimer-SR uses traces from both sides, while Nettimer-R uses only receiver-side traces. `multiQ` also uses only receiver-side traces. Ideally, the CDF should be a step function at "0", meaning that all experiments reported the "true" capacity. A negative relative error means that the tool has underestimated the capacity, whereas a positive relative error means that the tool has overestimated it.

Our results show that minimum capacity measurements are relatively accurate. On 85% of the paths, `multiQ`, Pathrate, and Nettimer-SR all report estimates within 10% of the "true" value. When Nettimer is given only the receiver-side trace, however, only 74% of its estimates are within 10% of the actual values. All three methods are biased towards underestimating the capacity.

Next, we look more closely at the errors exhibited by each tool. `multiQ` errors are caused mainly by over-smoothing in the iterative procedure for discovering mode gaps, which flattens the modes and prevents accurate computation of the gaps. Pathrate's logs indicate that its errors happen when the interarrival's distribution exhibits many modes. Though the correct bottleneck capacity is usually one of the modes discovered by Pathrate, the tool picks a different mode as the bottleneck capacity. When Nettimer made errors, we found that often the path has low RTT (< 16 ms). The tool mistakes the RTT mode in the inter-arrival PDF for the transmission time over the bottleneck. The effect is most pronounced when Nettimer is operating with only traces at the receiver side; when it has both traces, we theorize that it can estimate the RTT and eliminate the corresponding mode.

Our experiments show that different tools can disagree on the capacity of a particular path, but can all be correct. We noticed that, on some paths, the Pathrate estimate differs substantially from the Nettimer and `multiQ` estimates. In particular, Pathrate repeatedly reports capacities of 100 Mb/s for paths going to the speakeasy RON node and 1 Mb/s for paths going to cybermesa, while Nettimer and `multiQ` estimate them as 1.5 Mb/s and 10 Mb/s (Table 2). Further investigation revealed that the differences are due to the flows being rate limited. Speakeasy rate-limits TCP traffic to 1.5 Mb/s, which is the Nettimer and `multiQ` estimate. UDP flows are not limited, so Pathrate, which sends UDP packets, sees a link of 100Mb/s. In contrast, the cybermesa access link capacity of 10 Mb/s is correctly estimated by Nettimer and `multiQ`. Pathrate's relatively long trains of back-to-back packets, however, trigger cybermesa's leaky bucket rate limit; they exceed the maximum

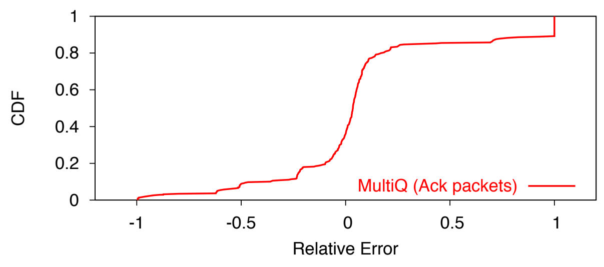


Figure 5—The accuracy of capacity estimates based on ack interarrivals.

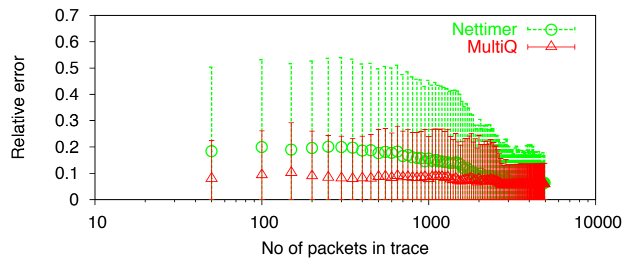


Figure 6—The relative error of MultiQ and Nettimer as a function of the traced flow size. Both average error and deviation are lower in the case of MultiQ.

burst size of the leaky bucket and becomes limited by the token rate, which is 1 Mb/s. TCP windows stay smaller than the bucket size, and so its packets are spaced by the actual link. This information has been confirmed by the owner sites.

7.4 Minimum Capacity Estimation Using Acks

Unlike existing tools, `multiQ` can obtain a reasonable capacity estimate exclusively using a *sender*-side trace, using the interarrival times of ack packets. Figure 5 shows the relative error of `multiQ`'s sender-side ack estimation, compared with its receiver-side data-packet estimation; the data comes from the experiments described in § 7.1. Since acks contain information about both forward and reverse links, we define the true capacity C_t for sender-side `multiQ` measurements as the minimum of the forward and reverse paths' capacities. Sender-side ack interarrivals produce lower-quality results than receiver-side data packet interarrivals, but still, 70% of the measurements are within 20% of the "true" value. Unlike receiver-side `multiQ`, the errors on sender-side `multiQ` tend towards overestimation.

7.5 Relative Error and Flow Size

We would expect capacity estimate error to be dependent on the amount of data available: more data should mean a better estimate. In this section, we quantify this effect.

Figure 6 plots the absolute value of the relative error of Nettimer-SR and `multiQ`'s estimates, as a function of the number of packets in the traced flow. We use the traces generated for § 7.1, truncated to various lengths; the relative errors are averaged over the whole set of RON paths. The bars show one standard deviation away from the average error. `multiQ`'s error is lower than Nettimer's for smaller numbers of packets. In fact, `multiQ`'s average error does not depend much on the number of packets, but the error variance decreases substantially as the number of traced packets increases. This means that there are particular flows in the data set that were hard to analyze and required a large number of packets for correct estimation. Also, the average error and error variance converge to nonzero values as the number of packets increases. This means that there are certain very noisy paths which neither `multiQ` nor Nettimer can correctly analyze, regardless of the number of traced packets.

Pathrate, on the other hand, is active. On our tests, it uses an average of 1317 probe packets, with a standard deviation of 1888 packets; but since it uses probes of varying sizes, a better metric is the amount of traffic it sends: 1.75 MB on average, with a standard deviation of 2.56 MB. The large standard deviation indicates that Pathrate uses far more traffic on paths that are hard to estimate.

7.6 Tight Links

This section evaluates `multiQ`'s ability to discover non-minimum-capacity bottlenecks, or *tight links*; as discussed above, `multiQ` can report up to three bottleneck capacities per flow. Unfortunately, we usually cannot say with confidence what the tight links along a path could be, and we can't correlate any results with other tools. To deal with this issue, we limit this test to Internet2 paths. Internet2 has a very low utilization (MRTG plots a maximum utilization $< 10\%$ [1]), so any observed queuing should be at the edges. Thus, for these paths we are reasonably confident that congestion happens at one or both access links, whose capacities we know. Also, because downstream narrow links tend to erase the effect of upstream bottlenecks (see § **??**), we limit this test to paths in which the downstream bottleneck capacity is larger than the upstream bottleneck capacity.

We run `ttcp` over each of these paths and log the packet arrival times at the receiver using `tcpdump`. The experiment is repeated multiple times during both peak and off-peak hours. We run `multiQ` on the resulting traces and record the various link capacities which are output. Each of these estimates could be a link on the path. We say that a tight link on a path is correctly estimated if one of the non-minimum-capacity estimates from `multiQ` is within 20% of the actual tight link capacity. All other estimates for that path are considered to be incorrect. If only the minimum capacity is found for a path, the answer for that path is logged as "not estimated". Tables 3 and 4 summarize the results: 64% of the experiments reported a tight link present on the path, 15% reported an invalid tight link (a bottleneck that differed from the correct value by more than 20%), and the remainder only reported the minimum bottleneck. The experiments that correctly found a tight link had an average relative error of 0.156.

7.7 Lost Packets and Ack Correspondence

To validate `mystery`, we used 155 pairs of traces from the RON testbed, similar to those described in § 7.3. We run `mystery` on the sender-side trace (the hard case) and collect its main results—a set of lost data packets, and an ack correspondence mapping *AC*. These results can contain four kinds of mistakes: "lost" packets that were actually received; "deliv-

Result	Fraction
Correct	64%
Incorrect	15%
Not estimated	21%

Table 3—`multiQ` tight link estimates

Avg. Relative Error	Std. Deviation in Error
0.156	0.077

Table 4—Average relative error and standard deviation in the correctly estimated tight links.

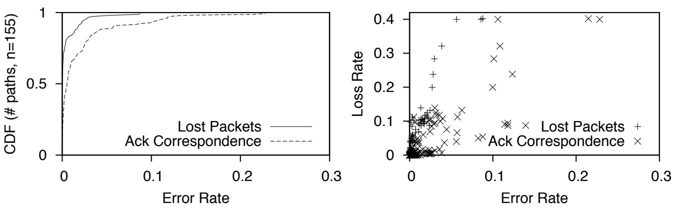


Figure 7—Error rates for `mystery`'s lost-packet and ack-correspondence detectors. On the left: error rate CDF; on the right: loss rate vs. error rate.

ered" packets that were actually lost; incorrect ack correspondences; and missing ack correspondences. All of these results are easy to check given the receiver-side trace. If we assume that all drops happen inside the network, then packets are delivered iff they show up in the receiver-side trace;[3] and ack correspondences is easy to determine at the receiver side, where acks show up in a few milliseconds rather than an RTT.

Figure 7 shows the results. Each graph has error rate as its X axis, where the error rate is the number of mistakes divided by the total number of events (data or ack packets sent). The lost packet detector is quite reliable, achieving 99% accuracy on 80% of the 155 paths; the ack correspondence detector is also reliable, but less so. Both error rates rise with the loss rate (right-hand graph), but the lost packet detector still achieves 90% accuracy on all paths. We investigated particular traces with high error rates, and found that many of the errors are impossible to fix without DSACK information or other explicit feedback. In particular, reverse-path losses cause problems for the tool. When the network drops the single ack sent in response to a packet, `mystery` cannot hope to detect that the packet was delivered.

8 Measurement Studies

We now turn to four `multiQ`- and `mystery`-based measurement studies of Internet path characteristics that could enable the construction of more realistic simulation scenarios. These studies are not intended to be complete; they are simply examples of results that are relatively easy to find using our measurement methodology and tools.

Several of these studies depend on the tools working together. This requirement points out another advantage of passive measurement: To combine the results of two active measurements, one might need to perform both measurements simultaneously, increasing measurement impact on the network; to combine the results of two passive measurements, you just run them both on the same trace.

[3]We do account for the very few packets that are dropped after the receiver trace point.

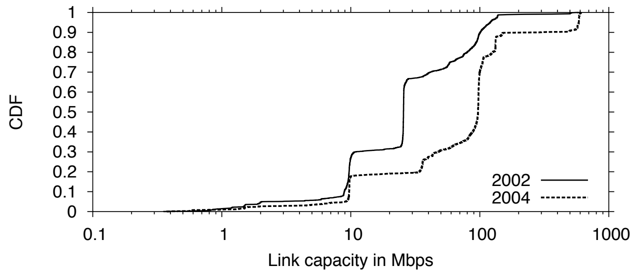


Figure 8—The empirical cumulative distribution of bottleneck capacity in the 2002 and 2004 NLANR datasets.

- **Evolution of bottleneck capacity.** We use `multiQ` to determine the bottleneck capacities in two large sets of NLANR traces [27], taken in 2002 and 2004.
- **Statistical multiplexing.** We estimate the level of statistical multiplexing on the NLANR traces' bottleneck links using `multiQ` (to measure capacity) and `mystery` (to measure throughput and RTT).
- **Loss and RTT.** `mystery` is used to plot how round-trip time changes around losses.
- **Loss and bottleneck capacity.** `mystery` calculates the loss event rate for packets in the NLANR traces; we plot this against bottleneck capacity calculated by `multiQ`.

The NLANR traces contain more than 375 million packets in 258 traces, collected on one OC-48, five OC-12, and fifteen OC-3 links. There are two sets of traces, one collected in 2002 and one in 2004. The traces contained over 50,000 significant flows. Although this data is not representative of all Internet traffic—for example, it all comes from within the US—it is large and diverse, and was collected at major connection points to the backbone.

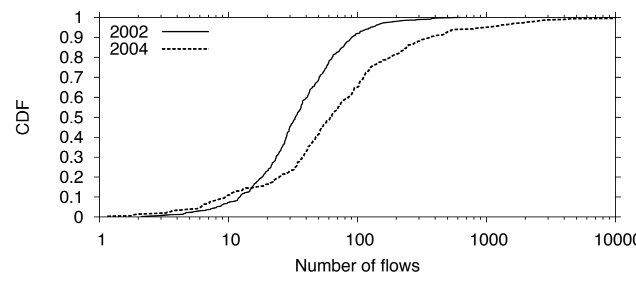
8.1 Bottleneck Capacity Distribution

We analyzed both the 2002 and 2004 NLANR trace sets using `multiQ`, extracting the bottleneck capacities experienced by every significant flow. Figure 8 demonstrates shows the shift in path capacity that occurred between the sets. In 2002, less than 20% of the significant flows were bottlenecked at a 100 Mb/s or higher capacity link. This number increased to 60% in 2004, showing a substantial and rapid growth in the capacity of bottleneck links. The highest bottleneck capacity that we identified in the 2002 data set is an OC-3 link. In contrast, the highest bottleneck capacity in the 2004 data set is an OC-12 link. Although this increase in bottleneck capacity is not uniformly distributed across all traces, it is impressive that the average bottleneck capacity has grown so much in a short period.

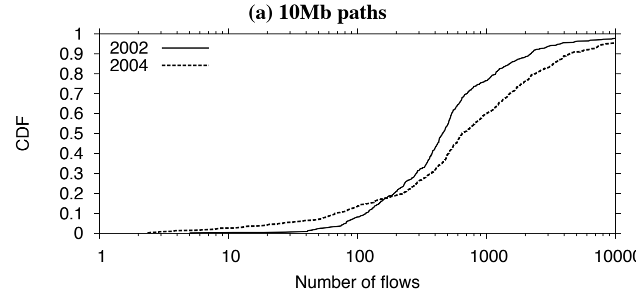
8.2 Statistical Multiplexing

Many published simulation scenarios assume low levels of statistical multiplexing on bottleneck links [13]. With `multiQ` and `mystery`, we can check this assumption.

We took the same NLANR traces from January, 2002 and 2004, and computed the level of statistical multiplexing for the



(a) 10Mb paths



(b) 100Mb paths

Figure 9—Distribution of statistical multiplexing on 10 and 100 Mb/s links in the 2002 and 2004 datasets.

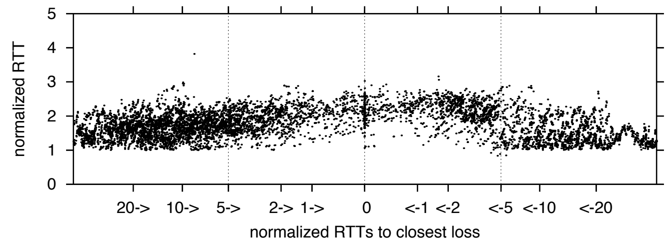


Figure 10—RTTs from a RON trace (from aros to ana1-gblx), plotted by time distance to the nearest lost packet.

two prevalent bottlenecks, the 10 Mb/s and the 100 Mb/s links. `multiQ` tells us the minimum-capacity bottleneck link; because this link is likely congested, we assume, as a first approximation, that the bottleneck capacity is distributed fairly among flows on that link. We then estimate the number of flows on a bottleneck as the ratio of the bottleneck's capacity to the throughput of the flow. Because TCP flows share a link in inverse proportion to their respective RTTs, we first normalize each flow's throughput with respect to the average RTT across all flows traversing the same bottleneck capacity. We used `multiQ` to determine the bottleneck capacity of each flow and `mystery` to compute its RTT. We did not calculate statistical multiplexing for traces with incomplete TCP header information.

Figure 9 shows CDFs of the level of statistical multiplexing on these paths. For the 10 Mb/s links, the median degree in the 2002 traces was 30, whereas it is 60 in the 2004 traces, corresponding to a fair share changing from 330 to 160 Kb/s. For the 100 Mb/s links, the median degree in 2002 was 450, and in 2004 was 650. The fair share bandwidth for these paths was somewhat lower than the 10 Mb/s links, decreasing from 220 Kb/s to 150 Kb/s. Contrary somewhat to conventional wisdom, we notice that the fair share bandwidth is not proportional to the bottleneck link capacity.

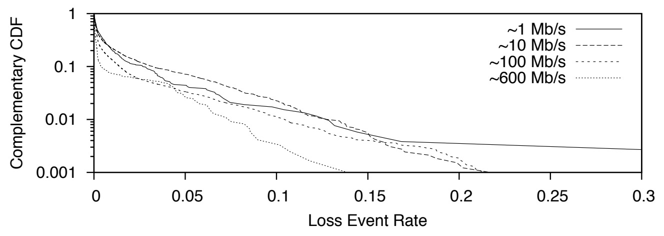


Figure 11—Complementary CDF of loss event rates for 13,627 significant flows from 2004 NLANR traces, divided into 4 bins by bottleneck capacity.

8.3 Losses and RTT

Together, `mystery`'s lost-packet and ack-correspondence detectors can produce plots that correlate RTT changes with losses. This has been an active area of research, motivated by the desire to deploy delay-based congestion control schemes; previous studies have depended on active probing [26] or on a limited set of RTT measurements, corresponding roughly to those that might be extracted on-line by a non-SACK TCP [24]. A `mystery`-based measurement offers both the relative ease of passive measurement, and a near-complete set of RTTs.

Figure 10 shows a representative graph taken from 155 runs over the RON traces described above; we show only one graph due to lack of space. As in prior work, little correlation between loss and delay is visible, even with `mystery`'s complete RTT information. More interesting are the differences between traces. For example, some traces show RTT *decreasing* before losses. Some cybermesa traces show no RTT variation whatsoever around losses, which might be explained by the leaky bucket rate limiter deployed there (§ 7.3).

8.4 Loss Rate and Bottleneck Bandwidth

Finally, Figure 11 shows a direct combination of results from `multiQ` and `mystery`: a plot of the loss event rates for flows differentiated by bottleneck capacity. We used `multiQ` to determine the bottleneck capacities of 15,000 significant flows from 2004 NLANR traces, and `mystery` to determine the loss event rate for each. We use TFRC's definition for loss event rate, namely the inverse of the average number of packets between loss events [12]; this is easily extracted from `mystery`'s output, a list of true loss events in the trace.

Loss events occur at all bottleneck capacities. Somewhat unexpectedly, the range of loss rates on 100 Mb/s-bottleneck flows is the same as for 10 Mb/s-bottleneck flows. Flows with 600 Mb/s bottleneck links still experience losses, but less so than flows with smaller bottlenecks.

9 Conclusions

We have presented the M&M set of passive tools for large-scale measurements and analysis of Internet path properties. The first tool, `multiQ`, is based on the insight that equally-spaced mode gaps (EMGs) in the packet interrarrival PDF correspond to the transmission time of 1500-byte packets on some congested link along the path. Uniquely to passive measurement tools, `multiQ` can discover the capacity of up to three bottlenecks and their relative location from a `tcpdump` trace of a flow. The second tool, `mystery`, detects several end-to-end parameters, such as loss rate and RTT. We calibrated these tools using extensive tests on 400 heterogeneous Internet paths.

To demonstrate the M&M tools in action, we applied them to a large collection of Internet traces containing over 375 million packets, investigating four properties of the network. Although these studies are not our main contribution, they produced interesting results—for example, that flows with 100 Mb/s bottleneck capacities achieve lower fair share bandwidth than flows with smaller capacities, due to higher levels of statistical multiplexing on the bottleneck links. The ease of creating these results given our tools, and their application to historical as well as current traces, show how M&M and tools like it can help achieve our goal: building and maintaining better mental models of the network.

For future work, we would like to use `multiQ` and `mystery` to address the following questions: "How many bottlenecks is a flow likely to encounter?" "When multiple queuing points exist, can one tell which among them is dropping the packets?" "Do published TCP equations accurately estimate the throughput obtained by real TCP flows?" Additionally, by running `multiQ` on both sender and receiver traces of the same flow, we would like to investigate whether bottlenecks on the reverse path are the same as those on the forward path. Most of `multiQ`'s errors identifying tight links are visually detectable by a human, indicating a potential for improved accuracy. Finally, we would like to integrate other measurement tools into the suite.

The M&M tools will be made publicly available under an open-source license by final publication.

REFERENCES

[1] Abilene. http://monon.uits.iupui.edu/.

[2] A. Akella, S. Seshan, and A. Shaikh. An Empirical Evaluation of Wide-Area Internet Bottlenecks. In *Proc. IMC*, Oct. 2003.

[3] M. Allman and V. Paxson. On Estimating End-to-End Network Path Properties. In *ACM SIGCOMM*, 1999.

[4] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Oct. 2001.

[5] H. Balakrishnan, V. Padmanabhan, and R. Katz. TCP Behavior of a Busy Internet Server: Analysis and Improvements. In *INFOCOM (1)*, 1998.

[6] P. Barford and M. Crovella. Critical path analysis of TCP transactions. In *SIGCOMM*, 2000.

[7] Cooperative Association for Internet Data Analysis (CAIDA). http://www.caida.org/.

[8] R. Carter and M. Crovella. Measuring Bottleneck link Speed in Packet-Switched Network. Technical Report TR-96-006, Boston University, Mar. 1996.

[9] K. Claffy, G. Miller, and K. Thompson. The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone, Apr. 1998. http://www.caida.org/outreach/resources/learn/packetsizes/.

[10] C. Dovrolis, P. Ramanathan, and D. Moore. Packet Dispersion Techniques and Capacity Estimation. *submitted to IEEE/ACM Transactions in Neworking*.

[11] C. Dovrolis, P. Ramanathan, and D. Moore. What do Packet Dispersion Techniques Measure? In *IEEE INFOCOM '01*, 2001.

[12] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proc. SIGCOMM*, Aug. 2000.

[13] S. Floyd and E. Kohler. Internet Research Needs Better Models. In *HotNets-I*, Oct. 2002.

[14] N. Hu and P. Steenkiste. Evaluation and Characterization of Available Bandwidth Techniques. *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, 2003.

[15] M. Jain and C. Dovrolis. Pathload: A Measurement Tool for End-to-End Available Bandwidth. In *Passive and Active Measurements*, March 2002.

[16] 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. In *Proc. IEEE INFOCOM*, Mar. 2003.

[17] H. Jiang and C. Dovrolis. Passive Estimation of TCP Round-Trip Times, 2002. To appear in ACM CCR.

[18] H. Jiang and C. Dovrolis. Source-Level IP Packet Bursts: Causes and Effects. In *Proc. IMC*, Oct. 2003.

[19] S. Keshav. A Control-Theoretic Approach to Flow Control. In *ACM SIGCOMM '88*, September 1991.

[20] K. Lai and M. Baker. Measuring Bandwidth. In *INFOCOM*, 1999.

[21] K. Lai and M. Baker. Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. In *Proc. USENIX*, 2001.

[22] G. Lu and X. Li. On the Correspondency between TCP Acknowledgement Packet and Data Packet. In *Proc. IMC*, Oct. 2003.

[23] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User Level Internet Path Diagnosis. In *Proc. ACM SOSP*, Oct. 2003.

[24] J. Martin, A. Nilsson, and I. Rhee. The Incremental Deployability of RTT-Based Congestion Avoidance for High Speed TCP Internet Connections. In *Measurement and Modeling of Computer Systems*, pages 134–144, 2000.

[25] B. Melander, M. Bjorkman, and P. Gunningberg. A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks. In *In Global Internet Symposium*, 2000.

[26] S. B. Moon, J. Kurose, P. Skelly, and D. Towsley. Correlation of Packet Delay and Loss in the Internet. Technical Report TR 98-11, Dept. of Computer Science, University of Massachusetts, Amherst, 1998.

[27] National Laboratory for Applied Network Research. http://pma.nlanr.net/.

[28] A. Pasztor and D. Veitch. The Packet Size Dependence of Packet Pair Methods. In *Proc. of 10th IWQoS*, 2003.

[29] pathchar. ftp://ee.lbl.gov/pathchar.tar.Z.

[30] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. In *ACM SIGCOMM*, pages 167–179, 1997.

[31] V. Paxson. On Calibrating Measurements of Packet Transit Times. In *Proc. SIGMETRICS 1998*, June 1998.

[32] V. E. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, 1997.

[33] V. J. Ribeiro, M. Coates, R. H. Riedi, S. Sarvotham, and R. G. Baraniuk. Multifractal Cross Traffic Estimation. In *Proc. of ITC Specialist Seminar on IP Traffic Measurement*, September 2000.

[34] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *Passive and Active Measurement Workshop*, 2003.

[35] D. Scott. *Multivariate Density Estimation*. John Wiley, 1992.

[36] C. Shannon, D. Moore, and K. Claffy. Beyond Folklore: Observations on Fragmented Traffic. In *IEEE/ACM Transactions on Networking*, Dec. 2002.

[37] J. Strauss, D. Katabi, and F. Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *Proc. IMC*, Oct. 2003.

[38] H. Uijiterwall and M. Santcroos. Bandwidth Estimations for Test Traffic Measurement Project, Dec. 2003. http://www.caida.org/outreach/isma/0312/slides/msantcroos.pdf.

[39] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the Characteristics and Origins of Internet Flow Rates. In *ACM SIGCOMM 2002*, 2002.