End-to-End Internet Packet Dynamics

Vern Paxson
Network Research Group
Lawrence Berkeley National Laboratory*
University of California, Berkeley
vern@ee.lbl.gov

Abstract

We discuss findings from a large-scale study of Internet packet dynamics conducted by tracing 20,000 TCP bulk transfers between 35 Internet sites. Because we traced each 100 Kbyte transfer at both the sender and the receiver, the measurements allow us to distinguish between the end-to-end behaviors due to the different directions of the Internet paths, which often exhibit asymmetries. We characterize the prevalence of unusual network events such as out-of-order delivery and packet corruption; discuss a robust receiver-based algorithm for estimating "bottleneck bandwidth" that addresses deficiencies discovered in techniques based on "packet pair"; investigate patterns of packet loss, finding that loss events are not well-modeled as independent and, furthermore, that the distribution of the duration of loss events exhibits infinite variance; and analyze variations in packet transit delays as indicators of congestion periods, finding that congestion periods also span a wide range of time scales.

1 Introduction

As the Internet grows larger, measuring and characterizing its dynamics grows harder. Part of the problem is how quickly the network changes. Another part, though, is its increasing heterogeneity. It is more and more difficult to measure a plausibly representative cross-section of its behavior. The few studies to date of end-to-end packet dynamics have all been confined to measuring a handful of Internet paths, because of the great logistical difficulties presented by larger-scale measurement [Mo92, Bo93, CPB93, Mu94]. Consequently, it is hard to gauge how representative their findings are for today's Internet. Recently, we devised a measurement framework in which a number of sites run special measurement daemons ("NPDs") to facilitate measurement. With this framework, the number of Internet paths available for measurement grows as N^2 for N sites, yielding an attractive scaling. We previously used the framework with $N = 37$ sites to study end-to-end routing dynamics of about 1,000 Internet paths [Pa96].

In this study we report on a large-scale experiment to study end-to-end Internet packet dynamics.[1] Our analysis is based on measurements of TCP bulk transfers conducted between 35 NPD sites (§ 2). Using TCP—rather than fixed-rate UDP or ICMP "echo" packets as done in [Bo93, CPB93, Mu94]—reaps significant benefits. First, TCP traffic is "real world," since TCP is widely used in today's Internet. Consequently, any network path properties we can derive from measurements of a TCP transfer can potentially be directly applied to tuning TCP performance. Second, TCP packet streams allow fine-scale probing without unduly loading the network, since TCP adapts its transmission rate to current congestion levels.

Using TCP, however, also incurs two serious analysis headaches. First, we need to distinguish between the apparently intertwined effects of the transport protocol and the network. To do so, we developed `tcpanaly`, a program that understands the specifics of the different TCP implementations in our study and thus can separate TCP behavior from network behavior [Pa97a]. `tcpanaly` also forms the basis for the analysis in this paper: after removing TCP effects, it then computes a wide range of statistics concerning network dynamics.

Second, TCP packets are sent over a wide range of time scales, from milliseconds to many seconds between consecutive packets. Such irregular spacing greatly complicates correlational and frequency-domain analysis, because a stream of TCP packets does not give us a traditional time series of constant-rate observations to work with. Consequently, in this paper we do not attempt these sorts of analyses, though we hope to pursue them in future work. See also [Mu94] for previous work in applying frequency-domain analysis to Internet paths.

In § 3 we characterize unusual network behavior: out-of-order delivery, replication, and packet corruption. Then in § 4 we discuss a robust algorithm for estimating the "bottleneck" bandwidth that limits a connection's maximum rate. This estimation is crucial for subsequent analysis because knowing the bottleneck rate lets us determine when the closely-spaced TCP data packets used for our network probes are *correlated* with each other. (We note that the stream of ack packets returned by the TCP data receiver in general is *not* correlated, due to the small size and larger spacing of the acks.) Once we can determine which probes were correlated and which not, we then can turn to analysis of end-to-end Internet packet loss (§ 5) and delay (§ 6). In § 7 we briefly summarize our findings, a number of which challenge commonly-held assumptions about network behavior.

*The work was supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

[1] This paper is necessarily terse due to space limitations. A longer version is available [Pa97b].

2 The Measurements

We gathered our measurements using the "NPD" measurement framework we developed and discussed in [Pa96]. 35 sites participated in two experimental runs. The sites include educational institutes, research labs, network service providers, and commercial companies, in 9 countries. We conducted the first run, $\mathcal{N}_1$, during Dec. 1994, and the second, $\mathcal{N}_2$, during Nov–Dec. 1995. Thus, differences between $\mathcal{N}_1$ and $\mathcal{N}_2$ give an indication how Internet packet dynamics changed during the course of 1995. Throughout this paper, when discussing such differences, we always limit discussion to the 21 sites that participated in both $\mathcal{N}_1$ and $\mathcal{N}_2$.

Each measurement was made by instructing daemons running at two of the sites to send or receive a 100 Kbyte TCP bulk transfer, and to trace the results using `tcpdump` [JLM89]. Measurements occurred at Poisson intervals, which, in principle, results in unbiased measurement, even if the sampling rate varies [Pa96]. In $\mathcal{N}_1$, the mean per-site sampling interval was about 2 hours, with each site randomly paired with another. Sites typically participated in about 200 measurements, and we gathered a total of 2,800 pairs of traces. In $\mathcal{N}_2$, we sampled pairs of sites in a series of *grouped* measurements, varying the sampling rate from minutes to days, with most rates on the order of 4–30 minutes. These groups then give us observations of the path between the site pair over a wide range of time scales. Sites typically participated in about 1,200 measurements, for a total of 18,000 trace pairs. In addition to the different sampling rates, the other difference between $\mathcal{N}_1$ and $\mathcal{N}_2$ is that in $\mathcal{N}_2$ we used Unix socket options to assure that the sending and receiving TCPs had big "windows," to prevent window limitations from throttling the transfer's throughput.

We limited measurements to a total of 10 minutes. This limit leads to *under-representation* of those times during which network conditions were poor enough to make it difficult to complete a 100 Kbyte transfer in that much time. Thus, our measurements are *biased* towards more favorable network conditions. In [Pa97b] we show that the bias is negligible for North American sites, but noticeable for European sites.

3 Network Pathologies

We begin with an analysis of network behavior we might consider "pathological," meaning unusual or unexpected: out-of-order delivery, packet replication, and packet corruption. It is important to recognize pathological behaviors so subsequent analysis of packet loss and delay is not skewed by their presence. For example, it is very difficult to perform any sort of sound queueing delay analysis in the presence of out-of-order delivery, since the latter indicates that a first-in-first-out (FIFO) queueing model of the network does not apply.

3.1 Out-of-order delivery

Even though Internet routers employ FIFO queueing, any time a route changes, if the new route offers a lower delay than the old one, then reordering can occur [Mo92]. Since we recorded packets at both ends of each TCP connection, we can detect network reordering, as follows. First, we remove from our analysis any trace pairs suffering packet filter errors [Pa97a]. Then, for each arriving packet p_i, we check whether it was sent after the last non-reordered packet. If so, then it becomes the new such packet. Otherwise, we count its arrival as an instance of a network reordering. So, for example, if a flight of ten packets all arrive in the order sent except the last one arrives before all of the others, we consider this to reflect 9 reordered packets rather than 1. Using this definition emphasizes "late" arrivals rather than "premature" arrivals. It turns out that counting late arrivals gives somewhat higher ($\approx +25\%$) numbers than counting premature arrivals—not a big difference, though.

Observations of reordering. Out-of-order delivery is fairly prevalent in the Internet. In $\mathcal{N}_1$, 36% of the traces included at least one packet (data or ack) delivered out of order, while in $\mathcal{N}_2$, 12% did. Overall, 2.0% of *all* of the $\mathcal{N}_1$ data packets and 0.6% of the acks arrived out of order (0.3% and 0.1% in $\mathcal{N}_2$). Data packets are no doubt more often reordered than acks because they are frequently sent closer together (due to ack-every-other policies), so their reordering requires less of a difference in transit times.

We should *not* infer from the differences between reordering in $\mathcal{N}_1$ and $\mathcal{N}_2$ that reordering became less likely over the course of 1995, because out-of-order delivery varies greatly from site-to-site. For example, fully 15% of the data packets sent by the "`ucol`" site[2] during $\mathcal{N}_1$ arrived out of order, much higher than the 2.0% overall average. As discussed in [Pa96], we do not claim that the individual sites participating in the measurement framework are plausibly representative of Internet sites in general, so site-specific behavior cannot be argued to reflect general Internet behavior.

Reordering is also highly asymmetric. For example, only 1.5% of the data packets sent *to* `ucol` during $\mathcal{N}_1$ arrived out of order. This means a sender cannot soundly infer whether the packets it sends are likely to be reordered, based on observations of the acks it receives, which is too bad, as otherwise the reordering information would aid in determining the optimal duplicate ack threshold to use for fast retransmission (see below).

The site-to-site variation in reordering coincides with our earlier findings concerning route flutter among the same sites [Pa96]. We identified two sites as particularly exhibiting flutter, `ucol` and the "`wustl`" site. For the part of $\mathcal{N}_1$ during which `wustl` exhibited route flutter, 24% of all of the data packets it sent arrived out of order, a rather stunning degree of reordering. If we eliminate `ucol` and `wustl` from the analysis, then the proportion of all of the $\mathcal{N}_1$ data packets delivered out-of-order falls by a factor of two. We also note that in $\mathcal{N}_2$, packets sent by `ucol` were reordered only 25 times out of nearly 100,000 sent, though 3.3% of the data packets sent *to* `ucol` arrived out of order, dramatizing how over long time scales, site-specific effects can completely change.

Thus, we should not interpret the prevalence of out-of-order delivery summarized above as giving representative numbers for the Internet, but instead form the rule of thumb: Internet paths are *sometimes* subject to a high incidence of reordering, but the effect is strongly site-dependent, and apparently correlated with route fluttering, which makes sense since route fluttering provides a mechanism for frequently reordering packets.

We observed reordering rates as high as 36% of all packets arriving in a single connection. Interestingly, some of the most highly reordered connections did not suffer *any* packet loss, and no needless retransmissions due to false signals from duplicate acks. We also occasionally observed humongous reordering "gaps." However, the evidence suggests that these gaps are not due to route changes, but a different effect. Figure 1 shows a sequence plot exhibiting a massive reordering event. This plot reflects packet arrivals at the TCP receiver, where each square marks the upper sequence number of an

[2] See [Pa96] for specifics concerning the sites mentioned in this paper.

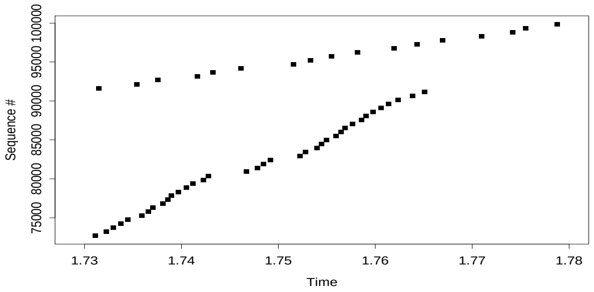

Figure 1: Out-of-order delivery with two distinct slopes

arriving data packet. All packets were sent in increasing sequence order.

Fitting a line to the upper points yields a data rate of a little over 170 Kbyte/sec, which was indeed the true (T1) bottleneck rate (§ 4). The slope of the packets delivered *late*, though, is just under 1 Mbyte/sec, consistent with an Ethernet bottleneck. What has apparently happened is that a router with Ethernet-limited connectivity to the receiver stopped forwarding packets for 110 msec just as sequence 72,705 arrived, most likely because at that point it processed a routing update [FJ94]. It finished between the arrival of 91,137 and 91,649, and began forwarding packets normally again at their arrival rate, namely T1 speed. Meanwhile, it had queued 35 packets while processing the update, and these it now finally forwarded whenever it had a chance, so they went out as quickly as possible, namely at Ethernet speed, but interspersed with new arrivals.

We observed this pattern a number of times in our data—not frequent enough to conclude that it is anything but a pathology, but often enough to suggest that significant momentary increases in networking delay can be due to effects different from both route changes and queueing; most likely due to router forwarding lulls.

Impact of reordering. While out-of-order delivery can violate one's assumptions about the network—in particular, the abstraction that it is well-modeled as a series of FIFO queueing servers—we find it often has little impact on TCP performance. One way it can make a difference, however, is in determining the TCP "duplicate ack" threshold a sender uses to infer that a packet requires retransmission. If the network never exhibited reordering, then as soon as the receiver observed a packet arriving that created a sequence "hole," it would know that the expected in-sequence packet was dropped, and could signal to the sender calling for prompt retransmission. Because of reordering, however, the receiver does *not* know whether the packet in fact was dropped; it may instead just be late. Presently, TCP senders retransmit if $N_d = 3$ "dups" arrive, a value chosen so that "false" dups caused by out-of-order delivery are unlikely to lead to spurious retransmissions.

The value of $N_d = 3$ was chosen primarily to assure that the threshold was conservative. Large-scale measurement studies were not available to further guide the selection of the threshold. We now examine two possible ways to improve the fast retransmit mechanism: by delaying the generation of dups to better disambiguate packet loss from reordering, and by altering the threshold to improve the balance between seizing retransmission opportunities, versus avoiding unneeded retransmissions.

We first look at packet reordering time scales to determine how long a receiver needs to wait to disambiguate reordering from loss. We only look at the time scales of data packet reorderings, since ack reorderings do not affect the fast retransmission process. We find a wide range of times between an out-of-order arrival and the later arrival of the last packet sent before it. One noteworthy artifact in the distribution is the presence of "spikes" at particular values, the strongest at 81 msec. This turns out to be due to a 56 Kbit/sec link, which has a bottleneck bandwidth of about 6,320 user data bytes/sec. Consequently, transmitting a 512 byte packet across the link requires 81.0 msec, so data packets of this size can arrive no closer, even if reordered. Thus we see that reordering can have associated with it a *minimum* time, which can be quite large.

Inspecting the $\mathcal{N}_1$ distributions further, we find that a strategy of waiting 20 msec would identify 70% of the out-of-order deliveries. For $\mathcal{N}_2$, the same proportion can be achieved waiting 8 msec, due to its overall shorter reordering times (presumably due to overall higher bandwidths). Thus, even though the upper end of the distribution is very large (12 seconds!), a generally modest wait serves to disambiguate most sequence holes.

We now look at the degree to which false fast retransmit signals due to reordering are actually a problem. We classify each sequence of dups as either *good* or *bad*, depending on whether a retransmission in response to it was necessary or unnecessary. When considering a refinement to the fast retransmission mechanism, our interest lies in the resulting ratio of *good* to *bad*, $R_{g:b}$, controlled by both the dup ack threshold value N_d we consider, and the *waiting time*, W, observed by the receiver before generating a dup upon the advent of a sequence hole.

For current TCP, $N_d = 3$ dups and $W = 0$. For these values, we find in $\mathcal{N}_1$, $R_{g:b} = 22$, and in $\mathcal{N}_2$, $R_{g:b} = 300!$ The order of magnitude improvement between $\mathcal{N}_1$ and $\mathcal{N}_2$ is due to the use in $\mathcal{N}_2$ of bigger windows (§ 2), and hence more opportunity for generating *good* dups. Clearly, the current scheme works well. While $N_d = 4$ improves $R_{g:b}$ by about a factor of 2.5, it also diminishes fast retransmit opportunities by about 30%, a significant loss.

For $N_d = 2$, we gain about 65–70% more fast retransmit opportunities, a hefty improvement, each generally saving a connection from an expensive timeout retransmission. The cost, however, is that $R_{g:b}$ falls by about a factor of three. If the receiving TCP waited $W = 20$ msec before generating a second dup, then $R_{g:b}$ falls only slightly (30% for $\mathcal{N}_1$, not at all for $\mathcal{N}_2$). Unfortunately, adding to TCPs $N_d = 2$ coupled with the $W = 20$ msec delay requires both sender and receiver modifications, greatly increasing the problem of *deploying* the change. Since partial deployment of only the sender change ($N_d = 2$) significantly increases spurious retransmissions, we conclude that, due to the size of the Internet's installed base, safely lowering N_d is impractical.

We note that the TCP *selective acknowledgement* ("SACK") option, now pending standardization, also holds promise for honing TCP retransmission [MMSR96]. SACK provides sufficiently fine-grained acknowledgement information that the sending TCP can generally tell which packets require retransmission and which have safely arrived (§ 5.4). To gain any benefits from SACK, however, requires that both the sender and the receiver support the option, so the deployment problems are similar to those discussed above. Furthermore, use of SACK aids a TCP in determining *what* to retransmit, but not *when* to retransmit. Because these considerations are orthogonal, investigating the effects of lowering N_d to 2 merits investigation, even in face of impending deployment of SACK.

We observed one other form of dup ack series potentially leading to unnecessary retransmission. Sometimes a series occurs for which the original ack (of which the others are dups) had acknowledged *all* of the outstanding data. When this occurs, the subsequent dups are *always* due to an unnecessary retransmission arriving at the receiving TCP, until at least a round-trip time (RTT) after the sending TCP sends new data. For $N_d = 3$, these sorts of series are 2-15 times more frequent than *bad* series, which is why they merit discussion. They are about 10 times rarer than *good* series. They occur during retransmission periods when the sender has already filled all of the sequence holes and is now retransmitting unnecessarily. Use of SACK eliminates these series. So would the following heuristic: whenever a TCP receives an ack, it notes whether the ack covers all of the data sent so far. If so, it then ignores any duplicates it receives for the ack, otherwise it acts on them in accordance with the usual fast retransmission mechanism.

3.2 Packet replication

In this section we look at *packet replication*: the network delivering multiple copies of the same packet. Unlike reordering, it is difficult to see how replication can occur. Our imaginations notwithstanding, it does happen, albeit rarely. We suspect one mechanism may involve links whose link-level technology includes a notion of retransmission, and for which the sender of a packet on the link incorrectly believes the packet was not successfully received, so it sends the packet again.[3]

In $\mathcal{N}_1$, we observed only once instance of packet replication, in which a pair of acks, sent once, arrived 9 times, each copy coming 32 msec after the last. The fact that two packets were together replicated does not fit with the explanation offered above for how a single packet could be replicated, since link-layer effects should only replicate one packet at a time. In $\mathcal{N}_2$, we observed 65 instances of the network infrastructure replicating a packet, all of a single packet, the most striking being 23 copies of a data packet arriving in a short blur at the receiver. Several sites dominated the $\mathcal{N}_2$ replication events: in particular, the two Trondheim sites, "`sintef1`" and "`sintef2`", accounted for half of the events (almost all of these involving `sintef1`), and the two British sites, "`ucl`" and "`ukc`", for half the remainder. After eliminating these, we still observed replication events among connections between 7 different sites, so the effect is somewhat widespread.

Surprisingly, packets can also be replicated at the sender, before the network has had much of a chance to perturb them. We know these are true replications and not packet filter duplications, as discussed in [Pa97a], because the copies have had their TTL fields decremented. There were no sender-replicated packets in $\mathcal{N}_1$, but 17 instances in $\mathcal{N}_2$, involving two sites (so the phenomenon is clearly site-specific).

3.3 Packet corruption

The final pathology we look at is *packet corruption*, in which the network delivers to the receiver an imperfect copy of the original packet. For data packets, `tcpanaly` cannot directly verify the checksum because the packet filter used in our study only recorded the packet headers, and not the payload. (For "pure acks," i.e., acknowledgement packets with no data payload, it directly verifies the checksum.) Consequently, `tcpanaly` includes algorithms to infer whether data packets arrive with invalid checksums, discussed in [Pa97a]. Using that analysis, we first found that one site, "`lbli`," was much more prone to checksum errors than any other. Since `lbli`'s Internet link is via an ISDN link, it appears quite likely that these are due to noise on the ISDN channels.

After eliminating `lbli`, the proportion of corrupted packets is about 0.02% in both datasets. No other single site strongly dominated in suffering from corrupted packets, and in $\mathcal{N}_2$, most of the sites receiving corrupted packets had fast (T1 or greater) Internet connectivity, so the corruptions are not primarily due to noisy, slow links. Thus, this evidence suggests that, as a rule of thumb, the proportion of Internet data packets corrupted in transit is around 1 in 5,000 (but see below).

A corruption rate of 1 packet in 5,000 is certainly not negligible, because TCP protects its data with a 16-bit checksum. Consequently, on average one bad packet out of 65,536 will be erroneously accepted by the receiving TCP, resulting in *undetected data corruption*. If the 1 in 5,000 rate is indeed correct, then about one in every 300 million Internet packets is accepted with corruption—certainly, many each day. In this case, we argue that TCP's 16-bit checksum is no longer adequate, if the goal is that globally in the Internet there are very few corrupted packets accepted by TCP implementations. If the checksum were instead 32 bits, then only about one in $2 \cdot 10^{13}$ packets would be accepted with corruption.

Finally, we note that the data checksum error rate of 0.02% of the packets is much higher than that measured directly (by verifying the checksum) for pure acks. For pure acks, we found only 1 corruption out of 300,000 acks in $\mathcal{N}_1$, and, after eliminating `lbli`, 1 out of 1.6 million acks in $\mathcal{N}_2$. This discrepancy can be partially addressed by accounting for the different lengths of data packets versus pure acks. It can be further reconciled if "header compression" such as CSLIP is used along the Internet paths in our study [Ja90], as that would greatly increase the relative size of data packets to that of pure acks. But it seems unlikely that header compression is widely used for high-speed links, and most of the inferred $\mathcal{N}_2$ data packet corruptions occurred for T1 and faster network paths.

One possibility is that the packets inferred by `tcpanaly` infer as arriving corrupted—because the receiving TCP did not respond to them in any fashion—actually were never received by the TCP for a different reason, such as inadequate buffer space. We partially tested for this possibility by computing corruption rates for only those traces monitored by a packet filter running on machine separate from the receiver (but on the same local network), versus those running on the receiver's machine. The former resulted in slightly higher inferred corruption rates, but not significantly so, so if the TCP is failing to receive the packets in question, it must be due to a mechanism that still enables the packet filter on the receiving machine to receive a copy. One can imagine such mechanisms, but it seems unlikely they would lead to drop rates of 1 in 5,000.

Another possibility is that data packets are indeed much more likely to be corrupted than the small pure ack packets, because of some artifact in how the corruption occurs. For example, it may be that corruption primarily occurs *inside* routers, where it goes undetected by any link-layer checksum, and that the mechanism (e.g., botched DMA, cache inconsistencies) only manifests itself for packets larger than a particular size.

Finally, we note that bit errors in packets transmitted using CSLIP can result in surprising artifacts when the CSLIP receiver reconstructs the packet header—such as introducing the appearance of in-sequence data, when none was actually sent!

In summary, we cannot offer a definitive answer as to overall Internet packet corruption rates: but the conflicting evidence that corruption may occur fairly frequently argues for further study in

[3] We have observed traces (not part of this study) in which more than 10% of the packets were replicated. The problem was traced to an improperly configured bridging device.

order to resolve the question.

4 Bottleneck Bandwidth

In this section we discuss how to estimate a fundamental property of a network connection, the *bottleneck bandwidth* that sets the upper limit on how quickly the network can deliver the sender's data to the receiver. The bottleneck comes from the slowest forwarding element in the end-to-end chain that comprises the network path. We make a crucial distinction between *bottleneck* bandwidth and *available* bandwidth. The former gives an upper bound on how fast a connection can *possibly* transmit data, while the less-well-defined latter term denotes how fast the connection *should* transmit to preserve network stability. Thus, available bandwidth never exceeds bottleneck bandwidth, and can in fact be much smaller (§ 6.3).

We will denote a path's bottleneck bandwidth as ρ_B. For measurement analysis, ρ_B is a fundamental quantity because it determines what we term the *self-interference time-constant*, Q_b. Q_b measures the amount of time required to forward a given packet through the bottleneck element. If a packet carries a total of b bytes and the bottleneck bandwidth is ρ_B byte/sec, then:

$$Q_b = \frac{b}{\rho_B}, \tag{1}$$

in units of seconds. From a queueing theory perspective, Q_b is simply the service time of a b-byte packet at the bottleneck link. We use the term "self-interference" because if the sender transmits two b-byte packets with an interval $\Delta T_s < Q_b$ between them, then the second one is guaranteed to have to wait behind the first one at the bottleneck element (hence the use of "Q" to denote "queueing"). We will always discuss Q_b in terms of *user data bytes*, i.e., TCP packet payload, and for ease of discussion will assume b is constant. We will not use the term for acks.

For our measurement analysis, accurate assessment of Q_b is critical. Suppose we observe a sender transmitting packets p_1 and p_2 an interval ΔT_s apart. Then if $\Delta T_s < Q_b$, the delays experienced by p_1 and p_2 are *perforce correlated*, and if $\Delta T_s \geq Q_b$ their delays, if correlated, are not due to self-interference but some other source (such as additional traffic from other connections, or processing delays). Thus, we need to know Q_b so we can distinguish those measurements that are necessarily correlated from those that are not. If we do not do so, then we will skew our analysis by mixing together measurements with built-in delays (due to queueing at the bottleneck) with measurements that do not reflect built-in delays.

4.1 Packet pair

The bottleneck estimation technique used in previous work is based on "packet pair" [Ke91, Bo93, CC96]. The fundamental idea is that if two packets are transmitted by the sender with an interval $\Delta T_s < Q_b$ between them, then when they arrive at the bottleneck they will be spread out in time by the transmission delay of the first packet across the bottleneck: after completing transmission through the bottleneck, their spacing will be exactly Q_b. Barring subsequent delay variations, they will then arrive at the receiver spaced not ΔT_s apart, but $\Delta T_r = Q_b$. We then compute ρ_B via Eqn 1.

The principle of the bottleneck spacing effect was noted in Jacobson's classic congestion paper [Ja88], where it in turn leads to the "self-clocking" mechanism. Keshav formally analyzed the behavior of packet pair for a network of routers that all obey the "fair queueing" scheduling discipline (not presently used in the Internet), and developed a provably stable flow control scheme based on packet pair measurements [Ke91]. Both Jacobson and Keshav were interested in estimating *available* rather than *bottleneck* bandwidth, and for this *variations* from Q_b due to queueing are of primary concern (§ 6.3). But if, as for us, the goal is to estimate ρ_B, then these variations instead become noise we must deal with.

Bolot used a stream of packets sent at fixed intervals to probe several Internet paths in order to characterize delay and loss [Bo93]. He measured round-trip delay of UDP echo packets and, among other analysis, applied the packet pair technique to form estimates of bottleneck bandwidths. He found good agreement with known link capacities, though a limitation of his study is that the measurements were confined to a small number of Internet paths.

Recent work by Carter and Crovella also investigates the utility of using packet pair in the Internet for estimating ρ_B [CC96]. Their work focusses on `bprobe`, a tool they devised for estimating ρ_B by transmitting 10 consecutive ICMP echo packets and recording the interarrival times of the consecutive replies. Much of the effort in developing `bprobe` concerns how to filter the resulting raw measurements in order to form a solid estimate. `bprobe` currently filters by first widening each estimate into an interval by adding an error term, and then finding the point at which the most intervals overlap. The authors also undertook to calibrate `bprobe` by testing its performance for a number of Internet paths with known bottlenecks. They found in general it works well, though some paths exhibited sufficient noise to sometimes produce erroneous estimates.

One limitation of both studies is that they were based on measurements made only at the data sender. (This is not an intrinsic limitation of the techniques used in either study). Since in both studies, the packets echoed back from the remote end were the same size as those sent to it, neither analysis was able to distinguish whether the bottleneck along the forward and reverse paths was the same. The bottleneck could differ in the two directions due to asymmetric routing, for example [Pa96], or because some media, such as satellite links, can have significant bandwidth asymmetries depending on the direction traversed [DMT96].

For estimating bottleneck bandwidth by measuring TCP traffic, a second problem arises: if the only measurements available are those at the sender, then "ack compression" (§ 6.1) can significantly alter the spacing of the small ack packets as they return through the network, distorting the bandwidth estimate. We investigate the degree of this problem below.

For our analysis, we consider what we term *receiver*-based packet pair (RBPP), in which we look at the pattern of data packet arrivals at the receiver. We also assume that the receiver has full timing information available to it. In particular, we assume that the receiver knows when the packets sent were *not* stretched out by the network, and can reject these as candidates for RBPP analysis. RBPP is considerably more accurate than sender-based packet pair (SBPP), since it eliminates the additional noise and possible asymmetry of the return path, as well as noise due to delays in generating the acks themselves. We find in practice this additional noise can be quite large.

4.2 Difficulties with packet pair

As shown in [Bo93] and [CC96], packet pair techniques often provide good estimates of bottleneck bandwidth. We find, however,

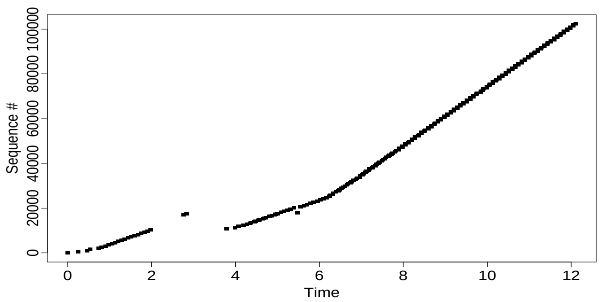

Figure 2: Bottleneck bandwidth change

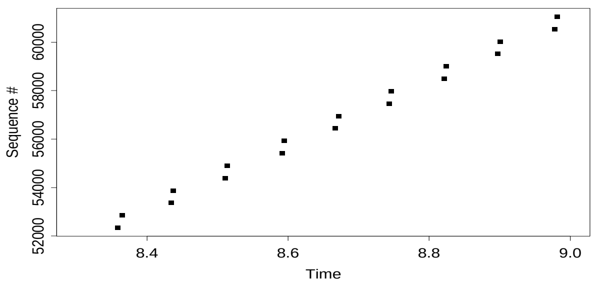

Figure 3: Enlargement of part of previous figure's right half

four potential problems (in addition to noise on the return path for SBPP). Three of these problem can often be addressed, but the fourth is more fundamental.

Out-of-order delivery. The first problem stems from the fact that for some Internet paths, out-of-order packet delivery occurs quite frequently (§ 3.1). Clearly, packet pairs delivered out of order completely destroy the packet pair technique, since they result in $\Delta T_r < 0$, which then leads to a negative estimate for ρ_B. Out-of-order delivery is symptomatic of a more general problem, namely that the two packets in a pair may not take the same route through the network, which then violates the assumption that the second queues behind the first at the bottleneck.

Limitations due to clock resolution. Another problem relates to the receiver's clock resolution, C_r, meaning the minimum difference in time the clock can report. C_r can introduce large margins of error around estimates of ρ_B. For example, if $C_r = 10$ msec, then for $b = 512$ bytes, packet pair cannot distinguish between $\rho_B =$ 51,200 byte/sec, and $\rho_B = \infty$.

We had several sites in our study with $C_r = 10$ msec. A technique for coping with large C_r is to use packet *bunch*, in which $k \geq 2$ back-to-back packets are used, rather than just two. Thus, the overall arrival interval ΔT_r^k spanned by the k packets will be about $k - 1$ times larger than that spanned by a single packet pair, diminishing the uncertainty due to C_r.

Changes in bottleneck bandwidth. Another problem that *any* bottleneck bandwidth estimation must deal with is the possibility that the bottleneck *changes* over the course of the connection. Figure 2 shows a sequence plot of data packets arriving at the receiver for a trace in which this happened. The eye immediately picks out a transition between one overall slope to another, just after $T = 6$. The first slope corresponds to 6,600 byte/sec, while the second is 13,300 byte/sec, and increase of a factor of two. Here, the change is due to `lbli`'s ISDN link activating a second *channel* to double the link bandwidth, but in general bottleneck shifts can occur due to other mechanisms, such as routing changes.

Multi-channel bottleneck links. A more fundamental problem with packet-pair techniques arises from the effects of *multi-channel* links, for which packet pair can yield *incorrect overestimates* even in the absence of any delay noise. Figure 3 expands a portion of Figure 2. The slope of the large linear trend in the plot corresponds to 13,300 byte/sec, as earlier noted. However, we see that the line is actually made up of pairs of packets. The slope between the pairs corresponds to a data rate of 160 Kbyte/sec. However, this trace involved `lbli`, a site with an ISDN link that has a hard limit of 128 Kbit/sec = 16 Kbyte/sec, a factor of ten smaller! Clearly, an estimate of $\rho_B \approx 160$ Kbyte/sec must be wrong, yet that is what a packet-pair calculation will yield.

What has happened is that the bottleneck ISDN link uses two *channels* that operate in *parallel*. When the link is idle and a packet arrives, it goes out over the first channel, and when another packet arrives shortly after, it goes out over the *other* channel. *They don't queue behind each other!* Multi-channel links violate the assumption that there is a *single* end-to-end forwarding path, with disastrous results for packet-pair, since in their presence it can form completely misleading overestimates for ρ_B.

We stress that the problem is more general than the circumstances shown in this example. First, while in this example the parallelism leading to the estimation error came from a single link with two separate physical channels, the exact same effect could come from a router that balances its outgoing load across two different links. Second, it may be tempting to dismiss this problem as correctable by using packet bunch with $k = 3$ instead of packet pair. This argument is not compelling without further investigation, however, because packet bunch could be more prone to error for regular bottlenecks; and, more fundamentally, $k = 3$ only works if the parallelism comes from *two* channels. If it came from *three* channels (or load-balancing links), then $k = 3$ will still yield misleading estimates.

4.3 Robust bottleneck estimation

Motivated by the shortcomings of packet pair, we developed a significantly more robust procedure, "packet bunch modes" (PBM). The main observation behind PBM is that we can deal with packet-pair's shortcomings by forming estimates for a *range* of packet bunch sizes, and by allowing for *multiple* bottleneck values or apparent bottleneck values. By considering different bunch sizes, we can accommodate limited receiver clock resolutions and the possibility of multiple channels or load-balancing across multiple links, while still avoiding the risk of underestimation due to noise diluting larger bunches, since we also consider small bunch sizes. By allowing for finding multiple bottleneck values, we again accommodate multi-channel (and multi-link) effects, and also the possibility of a bottleneck *change*.

Allowing for multiple bottleneck values rules out use of the most common robust estimator, the median, since it presupposes unimodality. We instead focus on identifying *modes*, i.e., local maxima in the density function of the distribution of the estimates. We then observe that:

(i) If we find two strong modes, for which one is found only at the beginning of the connection and one at the end, then we have evidence of a bottleneck *change*.

(ii) If we find two strong modes which span the same portion of the connection, and if one is found only for a packet bunch

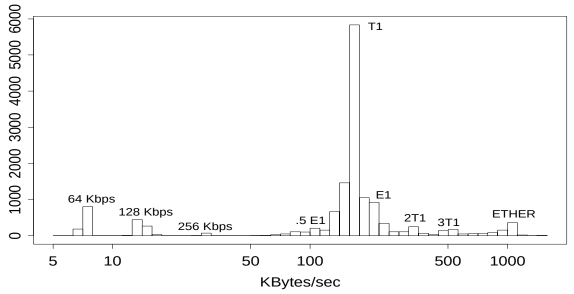

Figure 4: Histogram of single-bottleneck estimates for $\mathcal{N}_2$

size of m and the other only for bunch sizes $> m$, then we have evidence for an m-channel bottleneck link.

(iii) We can find both situations, for a link that exhibits both a change and a multi-channel link, such as shown in Figure 2.

Turning these observations into a working algorithm entails a great degree of niggling detail, as well as the use of a number of heuristics. Due to space limitations, we defer the particulars to [Pa97b]. We note, though, that one salient aspect of PBM is that it forms its final estimates in terms of *error bars* that nominally encompass $\pm 20\%$ around the bottleneck estimate, but might be narrower if estimates cluster sharply around a particular value, or wider if limited clock resolution prevents finer bounds. PBM always tries bunch sizes ranging from two packets to five packets. If required by limited clock resolution or the failure to find a compelling bandwidth estimate (about one quarter of all of the traces, usually due to limited clock resolution), it tries progressively larger bunch sizes, up to a maximum of 21 packets. We also note that nothing in PBM is specific to analyzing TCP traffic. All it requires is knowing when packets were sent relative to one another, how they arrived relative to one another, and their size.

We applied PBM to $\mathcal{N}_1$ and $\mathcal{N}_2$ for those traces for which `tcpanaly`'s packet filter and clock analysis did not uncover any uncorrectable problems [Pa97a, Pa97b]. After removing `lbli`, which frequently exhibited both bottleneck changes and multi-channel effects, PBM detected a single bottleneck 95–98% of the time; failed to produce an estimate 0-2% of the time (due to excessive noise or reordering); detected a bottleneck change in about 1 connection out of 250; and inferred a multi-channel bottleneck in 1-2% of the connections (though some of these appear spurious). Since all but single bottlenecks are rare, we defer discussion of the others to [Pa97b], and focus here on the usual case of finding a single bottleneck.

Unlike [CC96], we do not know *a priori* the bottleneck bandwidths for many of the paths in our study. We thus must fall back on self-consistency checks in order to gauge the accuracy of PBM. Figure 4 shows a histogram of the estimates formed for $\mathcal{N}_2$. (The $\mathcal{N}_1$ estimates are similar, though lower bandwidth estimates are more common.) The 170 Kbyte/sec peak clearly dominates, and corresponds to the speed of a T1 circuit after removing overhead.[4]. The 7.5 Kbyte/sec corresponds to 64 Kbit/sec links and the 13–14 Kbyte/sec peak reflects 128 Kbit/sec links. The 30 Kbyte/sec peak corresponds to a 256 Kbit/sec link, seen almost exclusively for connections involving a U.K. site. The 1 Mbyte/sec peaks are due to Ethernet bottlenecks, and likely reflect T3-connectivity beyond the limiting Ethernet.

[4] Recall that we compute ρ_B in terms of TCP *payload* bytes.

We speculate that the 330 Kbyte/sec peak reflects use of two T1 circuits in parallel, 500 Kbyte/sec reflects three T1 circuits (not half an Ethernet, since there is no easy way to subdivide an Ethernet's bandwidth), and 80 Kbyte/sec comes from use of half of a T1. Similarly, the 100 Kbyte/sec peak most likely is due to splitting a single E1 circuit in half. Indeed, we find non-North American sites predominating these connections, as well exhibiting peaks at 200–220 Kbyte/sec, above the T1 rate and just a bit below E1. This peak is absent from North American connections.

In summary, we believe we can offer plausible explanations for all of the peaks. Passing this self-consistency test in turn argues that PBM is indeed detecting true bottleneck bandwidths.

We next investigate the stability of bottleneck bandwidth over time. If we consider successive estimates for the same sender/receiver pair, then we find that 50% differ by less than 1.75%; 80%, by less than 10%; and 98% differ by less than a factor of two. Clearly, bottlenecks change infrequently.

The last property of bottleneck bandwidth we investigate is symmetry: how often is the bottleneck from host A to host B the same as that from B to A? Bottleneck asymmetries are an important consideration for sender-based "echo" measurement techniques, since these will observe the *minimum* bottleneck of the two directions [Bo93, CC96]. We find that for a given pair of hosts, the median estimates in the two directions differ by more than $\pm$ 20% about 20% of the time. This finding agrees with the observation that Internet paths often exhibit major routing asymmetries [Pa96]. The bottleneck differences can be quite large, with for example some paths T1-limited in one direction but Ethernet-limited in the other. In light of these variations, we see that sender-based bottleneck measurement will sometimes yield quite inaccurate results.

4.4 Efficacy of packet-pair

We finish with a look at how packet pair performs compared to PBM. We confine our analysis to those traces for which PBM found a single bottleneck. If packet pair produces an estimate lying within $\pm$ 20% of PBM's, then we consider it agreeing with PBM, otherwise not.

We evaluate "receiver-based packet pair" (RBPP, per § 4.1) by considering it as PBM limited to packet bunch sizes of 2 packets (or larger, if needed to resolve limited clock resolutions). We find RBPP estimates almost always (97–98%) agree with PBM. Thus, if (1) PBM's general clustering and filtering algorithms are applied to packet pair, (2) we do packet pair estimation at the *receiver*, (3) the receiver benefits from sender timing information, so it can reliably detect out-of-order delivery and lack of bottleneck "expansion," and (4) we are not concerned with multi-channel effects, then packet pair is a viable and relatively simple means to estimate the bottleneck bandwidth.

We also evaluate "sender-based packet pair" (SBPP), in which the sender makes measurements by itself. SBPP is of considerable interest because a sender can use it without any cooperation from the receiver, making it easy to deploy in the Internet. To fairly evaluate SBPP, we assume use by the sender of a number of considerations for forming sound bandwidth estimates, detailed in [Pa97b]. Even so, we find, unfortunately, that SBPP does not work especially well. In both datasets, the SBPP bottleneck estimate agrees with PBM only about 60% of the time. About one third of the estimates are too low, reflecting inaccuracies induced by excessive delays incurred by the acks on their return. The remaining 5–6% are overestimates (typically 50% too high), reflecting ack compression (§ 6.1).

5 Packet Loss

In this section we look at what our measurements tell us about packet loss in the Internet: how frequently it occurs and with what general patterns (§ 5.1); differences between loss rates of data packets and acks (§ 5.2); the degree to which loss occurs in bursts (§ 5.3); and how well TCP retransmission matches genuine loss (§ 5.4).

5.1 Loss rates

A fundamental issue in measuring packet loss is to avoid confusing measurement drops with genuine losses. Here is where the effort to ensure that `tcpanaly` understands the details of the TCP implementations in our study pays off [Pa97a]. Because we can determine whether traces suffer from measurement drops, we can exclude those that do from our packet loss analysis and avoid what could otherwise be significant inaccuracies.

For the sites in common, in $\mathcal{N}_1$, 2.7% of the packets were lost, while in $\mathcal{N}_2$, 5.2%, nearly twice as many. However, we need to address the question of whether the increase was due to the use of bigger windows in $\mathcal{N}_2$ (§ 2). With bigger windows, transfers will often have more data in flight and, consequently, load router queues much more.

We can assess the impact of bigger windows by looking at loss rates of *data* packets versus those for *ack* packets. Data packets stress the forward path much more than the smaller ack packets stress the reverse path, especially since acks are usually sent at half the rate of data packets due to ack-every-other-packet policies. On the other hand, the rate at which a TCP transmits data packets *adapts* to current conditions, while the ack transmission rate does not unless an entire flight of acks is lost, causing a sender timeout. Thus, we argue that ack losses give a clearer picture of overall Internet loss patterns, while data losses tell us specifically about the conditions as perceived by TCP connections.

In $\mathcal{N}_1$, 2.88% of the acks were lost and 2.65% of the data packets, while in $\mathcal{N}_2$ the figures are 5.14% and 5.28%. Clearly, the bulk of the difference between the $\mathcal{N}_1$ and $\mathcal{N}_2$ loss rates is not due to the use of bigger windows in $\mathcal{N}_2$. Thus we conclude that, overall, packet loss rates nearly doubled during 1995. We can refine these figures in a significant way, by conditioning on observing at least one loss during a connection. Here we make a tacit assumption that the network has two states, "quiescent" and "busy," and that we can distinguish between the two because when it is quiescent, we do not observe *any* (ack) loss.

In both $\mathcal{N}_1$ and $\mathcal{N}_2$, about half the connections had no ack loss. For "busy" connections, the loss rates jump to 5.7% in $\mathcal{N}_1$ and 9.2% in $\mathcal{N}_2$. Thus, even in $\mathcal{N}_1$, if the network was busy (using our simplistic definition above), loss rates were quite high, and for $\mathcal{N}_2$ they shot upward to a level that in general will seriously impede TCP performance.

So far, we have treated the Internet as a single aggregated network in our loss analysis. Geography, however, plays a crucial role. To study geographic effects, we partition the connections into four groups: "Europe," "U.S.," "Into Europe," and "Into U.S." European connections have both a European sender and receiver, U.S. have both in the United States. "Into Europe" connections have European data *senders* and U.S. data *receivers*. The terminology is backwards here because what we assess are *ack* loss rates, and these are generated by the receiver. Hence, "Into Europe" loss rates reflect those experienced by packet streams traveling from the U.S. into Europe. Similarly, "Into U.S." are connections with U.S. data senders and European receivers.

Region	Quies_1	Quies_2	Busy_1	Busy_2	Δ
Europe	48%	58%	5.3%	5.9%	+11%
U.S.	66%	69%	3.6%	4.4%	+21%
Into Europe	40%	31%	9.8%	16.9%	+73%
Into U.S.	35%	52%	4.9%	6.0%	+22%
All regions	53%	52%	5.6%	8.7%	+54%

Table 1: Conditional ack loss rates for different regions

Table 1 summarizes loss rates for the different regions, conditioning on whether any acks were lost ("quiescent" or "busy"). The second and third columns give the proportion of all connections that were quiescent in $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively. We see that except for the trans-Atlantic links going into the U.S., the proportion of quiescent connections is fairly stable. Hence, loss rate increases are primarily due to higher loss rates during the already-loaded "busy" periods. The fourth and fifth columns give the proportion of acks lost for "busy" periods, and the final column summarizes the relative change of these figures. None of the busy loss rates is especially heartening, and the trends are *all* increasing. The 17% $\mathcal{N}_2$ loss rate going into Europe is particularly glum.

Within regions, we find considerable site-to-site variation in loss rates, as well as variation between loss rates for packets inbound to the site and those outbound (§ 5.2). We did not, however, find any sites that seriously skewed the above figures.

In [Pa97b] we also analyze loss rates over the course of the day, here omitted due to limited space. We find an unsurprising diurnal pattern of "busy" periods corresponding to working hours and "quiescent" periods to late night and especially early morning hours. However, we also find that our *successful* measurements involving European sites exhibit a definite skew towards oversampling the quiescent periods, due to effects discussed in § 2. Consequently, the European loss rates given above are *underestimates*.

We finish with a brief look at how loss rates evolve over time. We find that observing a zero-loss connection at a given point in time is quite a good predictor of observing zero-loss connections up to several hours in the future, and remains a useful predictor, though not as strong, even for time scales of days and weeks [Pa97b]. Similarly, observing a connection that suffered loss is also a good predictor that future connections will suffer loss. The fact that prediction loses some power after a few hours supports the notion developed above that network paths have two general states, "quiescent" and "busy," and provides evidence that both states are long-lived, on time scales of hours. This again is not surprising, since we discussed earlier how these states exhibit clear diurnal patterns. That they are long-lived, though, means that caching loss information should prove beneficial.

Finally, we note that the predictive power of observing a specific loss *rate* is much lower than that of observing the presence of zero or non-zero loss. That is, even if we know it is a "busy" or a "quiescent" period, the loss rate measured at a given time only somewhat helps us predict loss rates at times not very far (minutes) in the future, and is of little help in predicting loss rates a number of hours in the future.

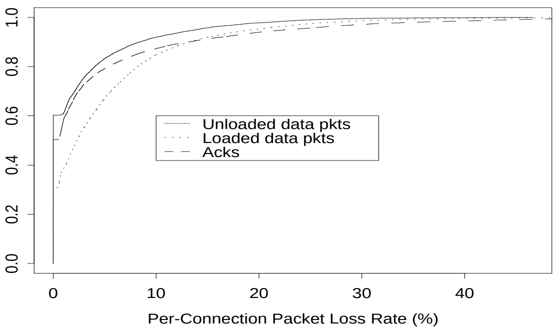

Figure 5: $\mathcal{N}_2$ loss rates for data packets and acks

5.2 Data packet loss vs. ack loss

We now turn to evaluating how patterns of packet loss differ among data packets (those carrying any user data) and ack packets. We make a key distinction between "loaded" and "unloaded" data packets. A "loaded" data packet is one that presumably had to queue at the bottleneck link behind one of the connection's previous packets, while an unloaded data packet is one that we know did not have to queue at the bottleneck behind a predecessor. We distinguish between the two by computing each packet's *load*, as follows.

Suppose the methodology in § 4 estimates the bottleneck bandwidth as ρ_B. It also provides *bounds* on the estimate, i.e., a minimum value ρ_B^- and a maximum ρ_B^+. We can then determine the maximum amount of time required for a b-byte packet to transit the bottleneck, namely: $\phi_b^+ = b/\rho_B^-$ sec.

Let T_i^s be the time at which the sender transmits the ith data packet. We then sequentially associate a *maximum load* λ_i^+ with each packet (assume for simplicity that b is constant). The first packet's load is:

$$\lambda_1^+ = \phi_b^+.$$

Subsequent packets have a load:

$$\lambda_i^+ = \phi_b^+ + \max\left[(T_{i-1}^s + \lambda_{i-1}^+) - T_i^s, 0\right].$$

λ_i^+ thus reflects the maximum amount of extra delay the ith packet incurs due to its own transmission time across the bottleneck link, plus the time required to first transmit any preceding packets across the bottleneck link, if i will arrive at the bottleneck before they completed transmission. In queueing theory terms, λ_i^+ reflects the ith packet's (maximum) waiting time at the bottleneck queue, in the absence of competing traffic from exogenous sources.

If $T_i^s < T_{i-1}^s + \lambda_{i-1}^+$, then we will term packet i "loaded," meaning that it had to wait for pending transmission of earlier packets. Otherwise, we term it "unloaded." (We can also develop "central" estimates rather than maximum estimates using ρ_B instead of ρ_B^- in this chain of reasoning. These are the values used in § 6.3.)

Using this terminology, in both $\mathcal{N}_1$ and $\mathcal{N}_2$, about 2/3's of the data packets were loaded. Figure 5 shows the distributions of loss rates during $\mathcal{N}_2$ for unloaded data packets, loaded data packets, and acks. All three distributions show considerable probability of zero loss. We immediately see that loaded packets are much more likely to be lost than unloaded packets, as we would expect. In addition, acks are consistently more likely than unloaded packets to be lost, but generally less likely to be lost than loaded packets, except during times of severe loss. We interpret the difference between ack and data loss rates as reflecting the fact that, while an

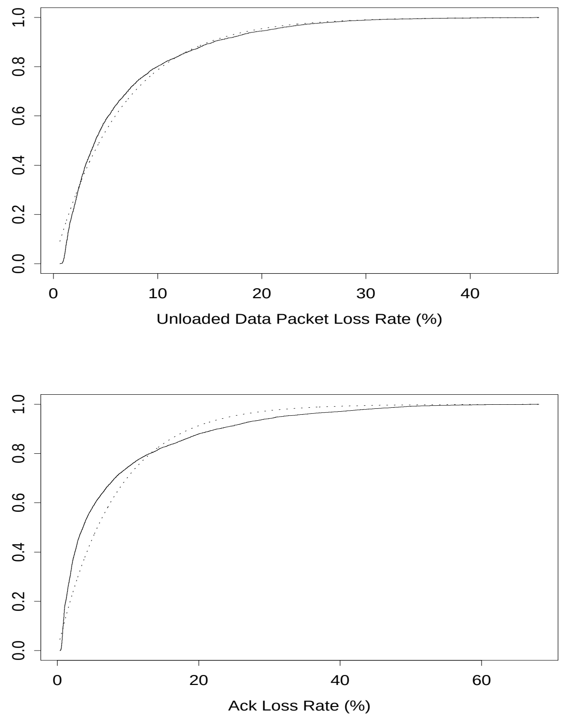

Figure 6: Distribution of $\mathcal{N}_2$ unloaded data packet and ack non-zero loss rates (solid), with fitted exponential distributions (dotted)

ack stream presents a much lighter load to the network than a data packet stream, the ack stream does *not* adapt to the current network conditions, while the data packet stream does, lowering its transmission rate in an attempt to diminish its loss rate.

It is interesting to note the extremes to which packet loss can reach. In $\mathcal{N}_2$, the largest unloaded data packet loss rate we observed was 47%. For loaded packets it climbed to 65%, and for acks, 68%. As we would expect, these connections all suffered egregiously. However, they *did* manage to successfully complete their transfers within their alloted ten minutes, a testimony to TCP's tenacity. For all of these extremes, *no* packets were lost in the reverse direction! Clearly packet loss on the forward and reverse paths is sometimes completely independent. Indeed, the coefficient of correlation between combined (loaded and unloaded) data packet loss rates and ack loss rates in $\mathcal{N}_1$ is 0.21, and in $\mathcal{N}_2$, the loss rates appear uncorrelated (coefficient of -0.02), perhaps due to the greater prevalence of significant routing asymmetry [Pa96].

Further investigating the loss rate distributions, one interesting feature we find is that the non-zero portions of both the unloaded and loaded data packet loss rates agree closely with exponential distributions, while that for acks is not so persuasive a match. Figure 6 shows the distributions of the per-connection loss rates for unloaded data packets (top) and acks (bottom) in $\mathcal{N}_2$, for those connections that suffered at least one loss. In both plots we have added an exponential distribution fitted to the mean of the loss rate (dotted). We see that for unloaded data packets (and also for loaded packets, not shown), the loss rate distribution is quite close to exponential, with the only significant disagreement in the lower tail. (This tail is subject to granularity effects, since for a trace with p packets, the mini-

Type of loss	P_l^u		P_l^c	
	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_1$	$\mathcal{N}_2$
Loaded data pkt	2.8%	4.5%	49%	50%
Unloaded data pkt	3.3%	5.3%	20%	25%
Ack	3.2%	4.3%	25%	31%

Table 2: Unconditional and conditional loss rates

mum non-zero loss rate will be $\frac{1}{p}$.) The close fit is widespread—not dominated by a few sites. For ack loss rates, however, we see that the fit is considerably less compelling.

While striking, interpreting the fit to the exponential distribution is difficult. If, for example, packet loss occurs independently and with a constant probability, then we would expect the loss rate to reflect a binomial distribution, but that is *not* what we observe. (We also know from the results in § 5.1 that there is *not* a single Internet packet loss rate, or anything approaching such a situation.)

It seems likely that the better exponential fit for data loss rates than ack loss rates holds a clue. The most salient difference between the transmission of data packets and that of acks is that the rate at which the sender transmits data packets *adapts* to the current network conditions, and furthermore it adapts *based on observing data packet loss*. Thus, if we passively measure the loss rate by observing the fate of a connection's TCP data packets, then we in fact are making measurements using a mechanism whose goal is to lower the value of what we are measuring (by spacing out the measurements). Consequently, we need to take care to distinguish between measuring overall Internet packet loss rates, which is best done using *non-adaptive* sampling, versus measuring loss rates *experienced* by a transport connection's packets—the two can be quite different.

Finally: the link between the adaptive sampling and the striking exponential distribution eludes us. We suspect it will prove an interesting area for further study.

5.3 Loss bursts

In this section we look at the degree to which packet loss occurs in *bursts* of more than one consecutive loss.

The first question we address is the degree to which packet losses are well-modeled as independent. In [Bo93], Bolot investigated this question by comparing the unconditional loss probability, P_l^u, with the conditional loss probability, P_l^c, where P_l^c is conditioned on the fact that the previous packet was also lost. He investigated the relationship between P_l^u and P_l^c for different packet spacings δ, ranging from 8 msec to 500 msec. He found that P_l^c approaches P_l^u as δ increases, indicating that loss correlations are short-lived, and concluded that "losses of probe packets are essentially random as long as the probe traffic uses less than 10% of the available capacity of the connection over which the probes are sent." The path he analyzed, though, included a heavily loaded trans-Atlantic link, so the patterns he observed might not be typical.

Table 2 summarizes P_l^u and P_l^c for the different types of packets and the two datasets. Clearly, for TCP packets we must discard the assumption that loss events are well-modeled as independent. Even for the low-burden, relatively low-rate ack packets, the loss probability jumps by a factor of seven if the previous ack was lost. We would expect to find the disparity strongest for loaded data packets, as these must contend for buffer with the connection's own previous packets, as well as any additional traffic, and indeed this is the case. We find the effect least strong for unloaded data packets, which accords with these not having to contend with the connection's previous packets, and having their rate diminished in the face of previous loss.[5]

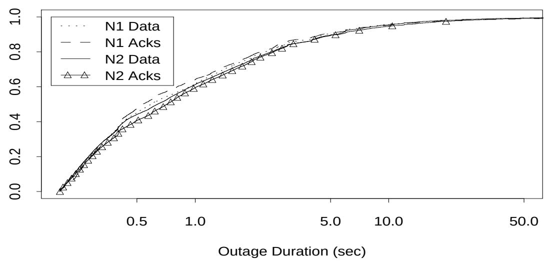

Figure 7: Distribution of packet loss outage durations exceeding 200 msec

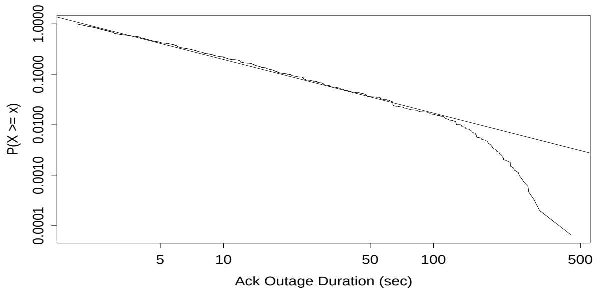

Figure 8: Log-log complementary distribution plot of $\mathcal{N}_2$ ack outage durations

The relative differences between P_l^u and P_l^c in Table 2 all exceed those computed by Bolot by a large factor. His greatest observed ratio of P_l^c to P_l^u was about 2.5:1. However, his P_l^u were all much higher than those in Table 2, even for $\delta = 500$ msec, suggesting that the path he measured differed considerably from a typical path in our study.

Given that packet losses occur in bursts, the next natural question is: how big? To address this question, we group successive packet losses into *outages*. Figure 7 shows the distribution of outage durations for those lasting more than 200 msec (the majority). We see that all four distributions agree fairly closely.

It is clear from Figure 7 that outage durations span several orders of magnitude. For example, 10% of the $\mathcal{N}_2$ ack outages were 33 msec or shorter (not shown in the plot), while another 10% were 3.2 sec or longer, a factor of a hundred larger. Furthermore, the upper tail of the distributions are consistent with Pareto distributions. Figure 8 shows a complementary distribution plot of the duration of $\mathcal{N}_2$ ack outages, for those lasting more than 2 sec (about 16% of all the outages). Both axes are log-scaled. A straight line on such a plot corresponds to a Pareto distribution. We have added a least-squares fit. We see the long outages fit quite well to a Pareto distribution with shape parameter $\alpha = 1.06$, except for the extreme

[5] It is interesting that loaded packets are unconditionally less likely to be lost than unloaded packets. We suspect this reflects the fact that lengthy periods of heavy loss or outages will lead to timeout retransmissions, and these are unloaded. Note that these statistics differ from the distributions shown in Figure 5 because those are for *per-connection* loss rates, while Table 2 summarizes loss probabilities over *all the packets* in each dataset.

Type of RR	Solaris$_1$	Solaris$_2$	Other$_1$	Other$_2$
% all packets	6%	6%	1%	2%
% retrans.	66%	59%	26%	28%
Unavoidable	14%	33%	44%	17%
Coarse feed.	1%	1%	51%	80%
Bad RTO	84%	66%	4%	3%

Table 3: Proportion of redundant retransmissions (RRs) due to different causes

upper tail, which is subject to truncation because of the 600 sec limit on connection durations (§ 2).

A shape parameter $\alpha \leq 2$ means that the distribution has *infinite variance*, indicating immense variability. Pareto distributions for activity and inactivity periods play key roles in some models of self-similar network traffic [WTSW95], suggesting that packet loss outages could contribute to how TCP network traffic might fit to ON/OFF-based self-similarity models.

Finally, we note that the patterns of loss bursts we observe might be greatly shaped by use of "drop tail" queueing. In particular, deployment of Random Early Detection could significantly affect these patterns and the corresponding connection dynamics [FJ93].

5.4 Efficacy of TCP retransmission

The final aspect of packet loss we investigate is how efficiently TCP deals with it. Ideally, TCP retransmits if and only if the retransmitted data was indeed lost. However, the transmitting TCP lacks perfect information, and consequently can retransmit unnecessarily. We analyzed each TCP transmission in our measurements to determine whether it was a *redundant retransmission* (RR), meaning that the data sent had already arrived at the receiver, or was in flight and would successfully arrive. We classify three types of RRs:

unavoidable because all of the acks for the data were lost;

coarse feedback meaning that had earlier acks conveyed finer information about sequence holes (such as provided by SACK), then the retransmission could have been avoided; and

bad RTO meaning that had the TCP simply waited longer, it would have received an ack for the data (bad retransmission timeout).

Table 3 summarizes the prevalence of the different types of RRs in $\mathcal{N}_1$ and $\mathcal{N}_2$. We divide the analysis into Solaris 2.3/2.4 TCP senders and others because in [Pa97a] we identified the Solaris 2.3/2.4 TCP as suffering from significant errors in computing RTO, which the other implementations do not exhibit. We see that in $\mathcal{N}_1$, a fair proportion of the RRs were unavoidable. (Some of these might however have been avoided had the receiving TCP generated more acks.) But for $\mathcal{N}_2$, only about 1/6 of the RRs for non-Solaris TCPs were unavoidable, the difference no doubt due to $\mathcal{N}_2$'s use of bigger windows (§ 2) increasing the mean number of acks in flight.

"Coarse feedback" RRs would presumably all be fixed using SACK, and these are the majority of RRs for non-Solaris TCPs. Solaris TCPs would not immediately benefit from SACK because many of their RRs occur before a SACK ack could arrive, anyway.

"Bad RTO" RRs indicate that the TCP's computation of the retransmission timeout was erroneous. These are the bane of Solaris 2.3/2.4 TCP, as noted above. Fixing the Solaris RTO calculation eliminates about 4-5% of *all* of the data traffic generated by the TCP.[6] For non-Solaris TCPs, bad RTO RRs are rare, providing solid evidence that the standard TCP RTO estimation algorithm developed in [Ja88] performs quite well for avoiding RRs. A separate question is whether the RTO estimation is overly conservative. A thorough investigation of this question is complex because a revised estimator might take advantage of both higher-resolution clocks and the opportunity to time multiple packets per flight. Thus, we leave this interesting question for future work.

In summary: ensuring standard-conformant RTO calculations and deploying the SACK option together eliminate virtually all of the avoidable redundant retransmissions. The remaining RRs are rare enough to not present serious performance problems.

6 Packet Delay

The final aspect of Internet packet dynamics we analyze is that of packet delay. Here we focus on network dynamics rather than transport protocol dynamics. Consequently, we confine our analysis to variations in one-way transit times (OTTs) and omit discussion of RTT variation, since RTT measurements conflate delays along the forward and reverse path.

For reasons noted in § 1, we do not attempt frequency-domain analysis of packet delay. We also do not summarize the marginal distribution of packet delays. Mukherjee found that packet delay along a particular Internet path is well-modeled using a shifted gamma distribution, but the parameters of the distribution vary from path to path and over the course of the day [Mu94]. Since we have about 1,000 distinct paths in our study, measured at all hours of the day, and since the gamma distribution varies considerably as its parameters are varied, it is difficult to see how to summarize the delay distributions in a useful fashion. We hope to revisit this problem in future work.

Any accurate assessment of delay must first deal with the issue of clock accuracy. This problem is particularly pronounced when measuring OTTs since doing so involves comparing measurements from two separate clocks. Accordingly, we developed robust algorithms for detecting clock *adjustments* and *relative skew* by inspecting sets of OTT measurements, described in [Pa97b]. The analysis in this section assumes these algorithms have first been used to reject or adjust traces with clock errors.

OTT variation was previously analyzed by Claffy and colleagues in a study of four Internet paths [CPB93]. They found that mean OTTs are often *not* well approximated by dividing RTTs in half, and that variations in the paths' OTTs are often asymmetric. Our measurements confirm this latter finding. If we compute the interquartile range (75th percentile minus 25th) of OTTs for a connection's unloaded data packets versus the acks coming back, in $\mathcal{N}_1$ the coefficient of correlation between the two is an anemic 0.10, and in $\mathcal{N}_2$ it drops to 0.006.

6.1 Timing compression

Packet timing *compression* occurs when a flight of packets sent over an interval ΔT_s arrives at the receiver over an interval $\Delta T_r < \Delta T_s$. To first order, compression should not occur, since the main mechanism at work in the network for altering the spacing between packets is queueing, which in general *expands* flights of packets (cf.

[6] We note that this problem has been fixed in Solaris 2.5.1.

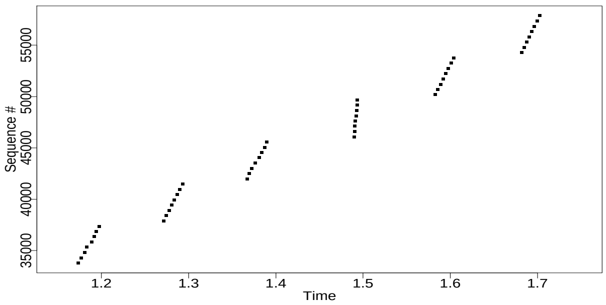

Figure 9: Data packet timing compression

§ 4.1). However, compression can occur if a flight of packets is at some point *held up* by the network, such that transmission of the first packet stalls and the later packets have time to catch up to it.

Zhang et al. predicted from theory and simulation that acks could be compressed ("ack compression") if a flight arrived at a router without any intervening packets from cross traffic (hence, the router's queue is *draining*) [ZSC91]. Mogul subsequently analyzed a trace of Internet traffic and confirmed the presence of ack compression [Mo92]. His definition of ack compression is somewhat complex since he had to infer endpoint behavior from an observation point inside the network. Since we can compute from our data both ΔT_s and ΔT_r, we can instead directly evaluate the presence of compression. He found compression was correlated with packet loss but considerably more rare. His study was limited, however, to a single 5-hour traffic trace.

Ack compression. To detect ack compression, for each group of at least 3 acks we compute:

$$\xi = \frac{\Delta T_r + C_r}{\Delta T_s - C_s}, \tag{2}$$

where C_r and C_s are the receiver and sender's clock resolutions, so ξ is a conservative estimate of the degree of compression. We consider a group compressed if $\xi < 0.75$. We term such a group a *compression event*. In $\mathcal{N}_1$, 50% of the connections experienced at least one compression event, and in $\mathcal{N}_2$, 60% did. In both, the mean number of events was around 2, and 1% of the connections experienced 15 or more. Almost all compression events are small, with only 5% spanning five or more acks. Finally, a significant minority (10–25%) of the compression events occurred for dup acks. These are sent with less spacing between them than regular acks sent by ack-every-other policies, so it takes less timing perturbation to compress them.

Were ack compression frequent, it would present two problems. First, as acks arrive they advance TCP's sliding window and "clock out" new data packets at the rate reflected by their arrival [Ja88]. For compressed acks, this means that the data packets go out *faster* than previously, which can result in network stress. Second, sender-based measurement techniques such as SBPP (§ 4.1) can misinterpret compressed acks as reflecting greater bandwidth than truly available. Since, however, we find ack compression relatively rare and small in magnitude, the first problem is not serious,[7] and the second can be dealt with by judiciously removing upper extremes from sender-based measurements.

[7]Indeed, it has been argued that occasional ack compression is beneficial, since it provides an opportunity for self-clocking to discover newly-available bandwidth.

Data packet timing compression. For data packet timing compression, our concerns are different. Sometimes a flight of data packets is sent at a high rate due to a sudden advance in the receiver's offered window. Normally these flights are spread out by the bottleneck and arrive at the receiver with a distance Q_b between each packet (§ 4). If after the bottleneck their timing is compressed, then use of Eqn 2 will *not* detect this fact unless they are compressed to a greater degree than their sending rate. Figure 9 illustrates this concern: the flights of data packets arrived at the receiver at 170 Kbyte/sec (T1 rate), except for the central flight, which arrived at Ethernet speed. However, it was also sent at Ethernet speed, so for it, $\xi \approx 1$.

Consequently, we consider a group of data packets as "compressed" if they arrive at greater than twice the upper bound on the estimated bottleneck bandwidth, ρ_B^+. We only consider groups of at least four data packets, as these, coupled with ack-every-other policies, have the potential to then elicit a pair of acks reflecting the compressed timing, leading to bogus self-clocking.

These compression events are rarer than ack compression, occurring in only 3% of the $\mathcal{N}_1$ traces and 7% of those in $\mathcal{N}_2$. We were interested in whether some paths might be plagued by repeated compression events due to either peculiar router architectures or network dynamics. Only 25–30% of the traces with an event had more than one, and just 3% had more than five, suggesting that such phenomena are rare. But those connections with multiple events are dominated by a few host pairs, indicating that the phenomenon does occur repeatedly, and is sometimes due to specific routers.

It appears that data packet timing compression is rare enough not to present a problem. That it does occur, though, again highlights the necessity for outlier-filtering when conducting timing measurements. (It also has a measurement benefit: from the arrival rate of the compressed packets, we can estimate the downstream bottleneck rate.)

6.2 Queueing time scales

In this section we briefly develop a rough estimate of the time scales over which queueing occurs. If we take care to eliminate suspect clocks, reordered packets, compressed timing, and traces exhibiting TTL shifts (which indicate routing changes), then we argue that the remaining measured OTT variation reflects queueing delays.

We compute the *queueing variation on the time scale* τ as follows. We partition the packets sent by a TCP into intervals of length τ. For each interval, let n_l and n_r be the number of successfully-arriving packets in the left and right halves of the interval. If either is zero, or if $n_l < \frac{1}{4}n_r$ or vice versa, then we reject the interval as containing too few measurements or too much imbalance between the halves. Otherwise, let m_l and m_r be the median OTTs of the two halves. We then define the interval's queueing variation as $|m_l - m_r|$. Finally, let ΔQ_τ be the median of $|m_l - m_r|$ over all such intervals.

Thus, ΔQ_τ reflects the "average" variation we observe in packet delays over a time scale of τ. By using medians, this estimate is robust in the presence of noise due to non-queueing effects, or queueing spikes. By dividing intervals in two and comparing only variation between the two halves, we confine ΔQ_τ to *only* variations on the time scale of τ. Shorter or longer lived variations are in general not included.

We now analyze ΔQ_τ for different values of τ, confining ourselves to variations in ack OTTs, as these are not clouded by self-interference and adaptive transmission rate effects. The question is:

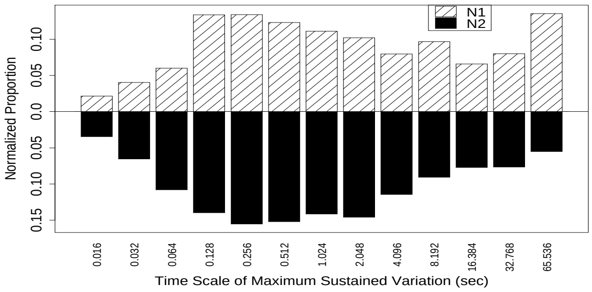

Figure 10: Proportion (normalized) of connections with given timescale of maximum delay variation ($\widehat{\tau}$)

are their particular τ's on which most queueing variation occurs? If so, then we can hope to engineer for those time scales. For example, if the dominant τ is less than a connection's RTT, then it is pointless for the connection to try to adapt to queueing fluctuations, since it cannot acquire feedback quickly enough to do so.

For each connection, we range through $2^4, 2^5, \ldots, 2^{16}$ msec to find $\widehat{\tau}$, the value of τ for which ΔQ_τ is greatest. $\widehat{\tau}$ reflects the time scale for which the connection experienced the greatest OTT variation. Figure 10 shows the normalized proportion of the connections in $\mathcal{N}_1$ and $\mathcal{N}_2$ exhibiting different values of $\widehat{\tau}$. Normalization is done by dividing the number of connections that exhibited $\widehat{\tau}$ with the number that had durations at least as long as $\widehat{\tau}$. For both datasets, time scales of 128–2048 msec primarily dominate. This range, though, spans more than an order of magnitude, and also exceeds typical RTT values. Furthermore, while less prevalent, $\widehat{\tau}$ values all the way up to 65 sec remain common, with $\mathcal{N}_1$ having a strong peak at 65 sec (which appears genuine; perhaps due to periodic outages caused by router synchronization [FJ94], eliminated by the end of 1995).

We summarize the figure as indicating that *Internet delay variations occur primarily on time scales of 0.1-1 sec, but extend out quite frequently to much larger times.*

6.3 Available bandwidth

The last aspect of delay variation we look at is an interpretation of how it reflects the *available bandwidth*. In § 5.2 we developed a notion of data packet i's "load," λ_i, meaning how much delay it incurs due to queueing at the bottleneck behind its predecessors, plus its own bottleneck transmission time ϕ_b. Since every packet requires ϕ_b to transit the bottleneck, *variations* in OTT do not include ϕ_b, but *will* reflect $\lambda_i - \phi_b$. Term this value ψ_i, and let γ_i denote the difference between packet i's measured OTT and the minimum observed OTT.

If the network path is completely unloaded except for the connection's load itself (no competing traffic), then we should have $\psi_i = \gamma_i$, i.e., all of i's delay variation is due to queueing behind its predecessors. More generally, define

$$\beta = \frac{\sum_i (\psi_i + \phi_i)}{\sum_j (\gamma_j + \phi_j)}.$$

β then reflects the proportion of the packet's delay due to the connection's own loading of the network. If $\beta \approx 1$, then all of the delay variation is due to the connection's own queueing load on the network, while, if $\beta \approx 0$, then the connection's load is *insignificant* compared to that of other traffic in the network.

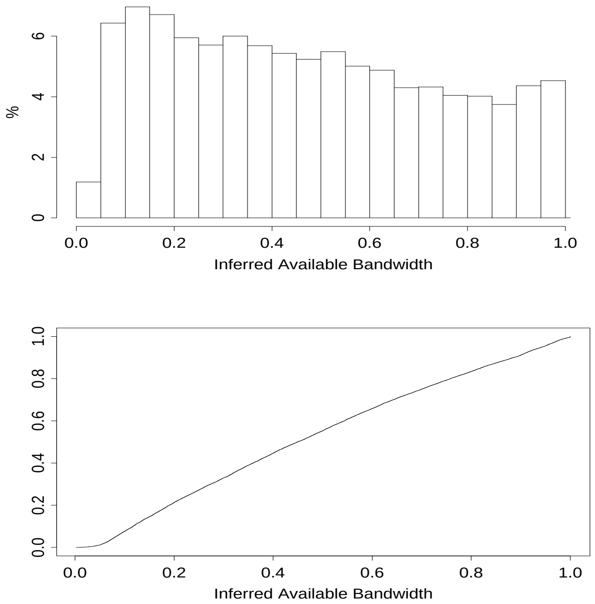

Figure 11: Density and cumulative distribution of $\mathcal{N}_2$ inferred available bandwidth (β)

More generally, $\sum_i (\psi_i + \phi_i)$ reflects the resources consumed by the connection, while $\sum_j (\gamma_j + \phi_j) - \sum_i (\psi_i + \phi_i) = \sum_j \gamma_j - \sum_i \psi_i$ reflects the resources consumed by the competing connections.

Thus, β captures the proportion of the total resources that were consumed by the connection itself, and we interpret β as reflecting the *available bandwidth*. Values of β close to 1 mean that the entire bottleneck bandwidth was available, and values close to 0 mean that almost none of it was actually available.

Note that we can have $\beta \approx 1$ even if the connection does not consume all of the network path's capacity. All that is required is that, to the degree that the connection did attempt to consume network resources, they were readily available. This observation provides the basis for hoping that we might be able to use β to estimate available bandwidth without fully stressing the network path.

We can gauge how well β truly reflects available bandwidth by computing the coefficient of correlation between β and the connection's overall throughput (normalized by dividing by the bottleneck bandwidth). For $\mathcal{N}_1$, this is 0.44, while for $\mathcal{N}_2$, it rises to 0.55.

Figure 11 shows the density and cumulative distribution of β for $\mathcal{N}_2$. Not surprisingly, we find that Internet connections encounter a broad range of available bandwidth.[8] As is generally the case with Internet characteristics, a single figure like this can oversimplify the situation. We note, for example, that confining the evaluation of β to European connections results in a sharp leftward shift in the density, indicating generally less available bandwidth, while for U.S. connections, the density shifts to the right. Furthermore, for paths with higher bottleneck bandwidths, we generally find lower values of β, reflecting that such paths tend to be shared among more competing connections. Finally, we note that the predictive power of β tends to be fairly good. On average, a given observation of β will be within 0.1 of later observations of β for the same path, for time periods up to several hours.

[8] The depressed density at $\beta \approx 0$ reflects a measurement bias [Pa97b].

7 Conclusions

Several conclusions emerge from our study:

- With due diligence to remove packet filter errors and TCP effects, TCP-based measurement provides a viable means for assessing end-to-end packet dynamics.
- We find wide ranges of behavior, such that we must exercise great caution in regarding any aspect of packet dynamics as "typical."
- Some common assumptions such as in-order packet delivery, FIFO bottleneck queueing, independent loss events, single congestion time scales, and path symmetries are all violated, sometimes frequently.
- When implemented correctly, TCP's retransmission strategies work in a sufficiently conservative fashion.
- The combination of path asymmetries and reverse-path noise render sender-only measurement techniques markedly inferior to those that include receiver-cooperation.

This last point argues that when the measurement of interest concerns a unidirectional path—be it for measurement-based adaptive transport techniques such as TCP Vegas [BOP94], or general Internet performance metrics such as those in development by the IPPM effort [A+96]—the extra complications incurred by coordinating sender and receiver are worth the effort.

8 Acknowledgements

This work would not have been possible without the efforts of the many volunteers who installed the Network Probe Daemon at their sites. I am indebted to:

G. Almes, J. Alsters, J-C. Bolot, K. Bostic, H-W. Braun, D. Brown, R. Bush, B. Camm, B. Chinoy, K. Claffy, P. Collinson, J. Crowcroft, P. Danzig, H. Eidnes, M. Eliot, R. Elz, M. Flory, M. Gerla, A. Ghosh, D. Grunwald, T. Hagen, A. Hannan, S. Haug, J. Hawkinson, TR Hein, T. Helbig, P. Hyder, A. Ibbetson, A. Jackson, B. Karp, K. Lance, C. Leres, K. Lidl, P. Linington, S. McCanne, L. McGinley, J. Milburn, W. Mueller, E. Nemeth, K. Obraczka, I. Penny, F. Pinard, J. Polk, T. Satogata, D. Schmidt, M. Schwartz, W. Sinze, S. Slaymaker, S. Walton, D. Wells, G. Wright, J. Wroclawski, C. Young, and L. Zhang.

I am likewise indebted to Keith Bostic, Evi Nemeth, Rich Stevens, George Varghese, Andres Albanese, Wieland Holfelder, and Bernd Lamparter for their invaluable help in recruiting NPD sites. Thanks, too, to Peter Danzig, Jeff Mogul, and Mike Schwartz for feedback on the design of NPD.

This work greatly benefited from discussions with Domenico Ferrari, Sally Floyd, Van Jacobson, Mike Luby, Greg Minshall, John Rice, and the comments of the anonymous reviewers. My heartfelt thanks.

References

[A+96] G. Almes et al, "Framework for IP Provider Metrics," Internet draft, *ftp://ftp.isi.edu/internet-drafts/draft-ietf-bmwg-ippm-framework-00.txt*, Nov. 1996.

[Bo93] J-C. Bolot, "End-to-End Packet Delay and Loss Behavior in the Internet," *Proc. SIGCOMM '93*, pp. 289-298, Sept. 1993.

[BOP94] L. Brakmo, S. O'Malley and L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *Proc. SIGCOMM '94*, pp. 24-35, Sept. 1994.

[CC96] R. Carter and M. Crovella, "Measuring Bottleneck Link Speed in Packet-Switched Networks," Tech. Report BU-CS-96-006, Computer Science Department, Boston University, Mar. 1996.

[CPB93] K. Claffy, G. Polyzos and H-W. Braun, "Measurement Considerations for Assessing Unidirectional Latencies," *Internetworking: Research and Experience*, 4 (3), pp. 121-132, Sept. 1993.

[DMT96] R. Durst, G. Miller and E. Travis, "TCP Extensions for Space Communications," *Proc. MOBICOM '96*, pp. 15-26, Nov. 1996.

[FJ93] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, 1(4), pp. 397-413, Aug. 1993.

[FJ94] S. Floyd and V. Jacobson, "The Synchronization of Periodic Routing Messages," *IEEE/ACM Transactions on Networking*, 2(2), pp. 122-136, Apr. 1994.

[Ja88] V. Jacobson, "Congestion Avoidance and Control," *Proc. SIGCOMM '88*, pp. 314-329, Aug. 1988.

[JLM89] V. Jacobson, C. Leres, and S. McCanne, `tcpdump`, available via anonymous ftp to ftp.ee.lbl.gov, June 1989.

[Ja90] V. Jacobson, "Compressing TCP/IP headers for low-speed serial links," RFC 1144, Network Information Center, SRI International, Menlo Park, CA, February, 1990.

[Ke91] S. Keshav, "A Control-Theoretic Approach to Flow Control", *Proc. SIGCOMM '91*, pp. 3-15, Sept. 1991.

[MMSR96] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, DDN Network Information Center, Oct. 1995.

[Mo92] J. Mogul, "Observing TCP Dynamics in Real Networks," *Proc. SIGCOMM '92*, pp. 305-317, Aug. 1992.

[Mu94] A. Mukherjee, "On the Dynamics and Significance of Low Frequency Components of Internet Load," *Internetworking: Research and Experience*, Vol. 5, pp. 163-205, December 1994.

[Pa96] V. Paxson, "End-to-End Routing Behavior in the Internet," *Proc. SIGCOMM '96*, pp. 25-38, Aug. 1996.

[Pa97a] V. Paxson, "Automated Packet Trace Analysis of TCP Implementations," *Proc. SIGCOMM '97*, Sep. 1997.

[Pa97b] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," Ph.D. dissertation, University of California, Berkeley, April 1997.

[WTSW95] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson, "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," *Proc. SIGCOMM '95*, pp. 100-113, Sept. 1995.

[ZSC91] L. Zhang, S. Shenker, and D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," *Proc. SIGCOMM '91*, pp. 133-147, Sept. 1991.