End-to-End Internet Packet Dynamics

Vern Paxson

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: 1) characterize the prevalence of unusual network events such as out-of-order delivery and packet replication; 2) discuss a robust receiver-based algorithm for estimating "bottleneck bandwidth" that addresses deficiencies discovered in techniques based on "packet pair;" 3) 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 4) analyze variations in packet transit delays as indicators of congestion periods, finding that congestion periods also span a wide range of time scales.**

Index Terms—**Computer networks, computer network performance, computer network reliability, failure analysis, internetworking, stability.**

I. 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 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 [1], [5], [17], [18]. Consequently, it is hard to gauge the degree to which their findings are representative. To address this problem, we devised a measurement framework in which a number of sites run special measurement daemons (NPD's) 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 the end-to-end routing dynamics of about 1000 Internet paths [20].

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 (Section II). Using TCP—rather than fixed-rate UDP or ICMP "echo" packets, as done in [1], [5], [18]—reaps significant benefits. First, TCP traffic is "real world," since TCP is widely used in today's Internet [25]. 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 presents two analysis problems. First, to analyze packet dynamics using TCP requires a way to measure the sending and receiving times of individual packets, which TCP does not provide. We instead must record the traffic with a packet filter. Packet-filter measurement can be imperfect, in particular suffering from measurement "drops," in which the filter fails to record all of the traffic. Unless we identify traces with such errors, we can derive inaccurate conclusions about packet dynamics such as loss rates. To address this problem, we developed `tcpanaly`, a program that understands the specifics of the different TCP implementations in our study, and thus can infer when the packet filter has made an error [21]. We then exclude erroneous traces from any analysis that would be skewed by the error. `tcpanaly` also forms the basis for the analysis in this paper: after verifying the trace's integrity, it then computes 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 [18] for previous work in applying frequency-domain analysis to Internet paths.)

In Section III, we characterize unusual network behavior: out-of-order delivery, replication, and packet corruption. Then in Section IV, 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 necessarily queue behind each other and, hence, their timing dynamics will be correlated. Once we can determine which probes were correlated and which not, we then can turn to analysis of end-to-end Internet packet loss (Section V) and delay (Section VI). In Section VII, we summarize our findings, several of which challenge commonly-held assumptions about network behavior.

Manuscript received April 14, 1998; revised February 26, 1999 and March 8, 1999; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. Partridge. This work was supported by the Director, Office of Energy Research, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division of the U.S. Department of Energy under Contract DE-AC03-76SF00098. An early version of this paper appeared in Proceedings of ACM SIGCOMM '97.

The author is with the Network Research Group, Lawrence Berkeley National Laboratory, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: vern@aciri.org).

Publisher Item Identifier S 1063-6692(99)05591-0.

[1] This paper is necessarily terse, due to space limitations. A longer version is available [22].

II. The Measurements

We gathered our measurements using the NPD measurement framework we developed and discussed in [20]. Thirty-five sites participated in two experimental runs. The sites include educational institutes, research labs, network service providers, and commercial companies, in nine countries. We conducted the first run, $\mathcal{N}_1$, during December 1994, and the second ($\mathcal{N}_2$) during November–December 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 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` [12]. Measurements occurred at Poisson intervals which, in principle, results in unbiased measurement, even if the sampling rate varies [20]. In $\mathcal{N}_1$, the mean per-site sampling interval was about 2 h, with each site randomly paired with another. Sites typically participated in about 200 measurements, and we gathered a total of 2800 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 min. 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 1200 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 TCP's had big "windows," to prevent window limitations from throttling the transfer's throughput.

We limited measurements to a total of 10 min. 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 toward more favorable network conditions. In [22] we show that the bias is negligible for North American sites, but noticeable for European sites.

III. 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.

A. 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 [17]. 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 [21]. Then, for each arriving packet p_i, we check whether it was sent after the last nonreordered 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 for the last one, which arrives before all of the others, we consider this to reflect nine reordered packets rather than one. 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, but as our goal is only to convey the rough magnitude of reordering, this difference is not particularly significant.

Observations of Reordering: Out-of-order delivery is fairly prevalent in the Internet. In $\mathcal{N}_1$, 36% of the connections included at least one packet (data or ack) delivered out of order, while in $\mathcal{N}_2$, 12% did. Overall, 2% 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.

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 unfortunate, as otherwise the reordering information would aid in determining the optimal duplicate ack threshold to use for TCP fast retransmission (see below).

The site-to-site variation in reordering coincides with earlier findings concerning route flutter among the same sites [20]. That study 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 correlated with route fluttering.

We observed reordering rates as high as 36% of all packets arriving in a single connection. Interestingly, some of the

[2]See [20] for specifics concerning the sites mentioned in this paper.

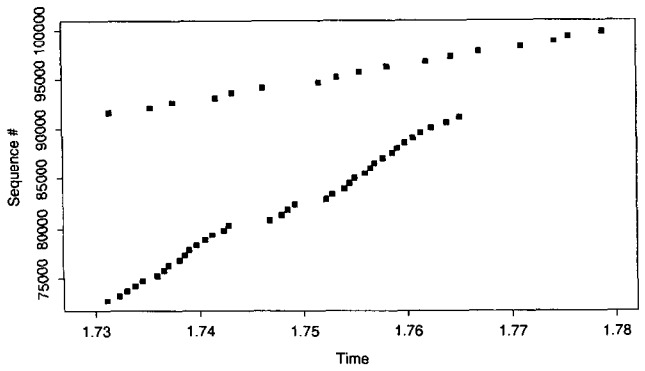

Fig. 1. Out-of-order delivery with two distinct slopes.

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 very large reordering "gaps." However, the evidence suggests that these gaps are not due to route changes, but a different effect. Fig. 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 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/s, which was indeed the true (T1) bottleneck rate (Section IV). The slope of the packets delivered *late*, though, is just under 1 Mbyte/s, 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 ms just as sequence 72 705 arrived, perhaps because at that point it processed a routing update [9]. 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 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 frequently 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—for the connections in our study, it only rarely had significant impact on TCP performance, because generally, the scale of the reordering was just a few packets.

In general, one way reordering can make a difference 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 had been 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 has been 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 ms. This turns out to be due to a 56 kbit/s link, which has a bottleneck bandwidth of about 6320 user data bytes/s. Consequently, transmitting a 512-byte packet across the link requires 81.0 ms, 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 ms would identify 70% of the out-of-order deliveries. For $\mathcal{N}_2$, the same proportion can be achieved waiting 8 ms, 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 s), 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 (Section II), and hence more opportunity for generating *good* dups. Clearly, the current scheme works well. While $N_d = 4$ improves $R_{g:b}$ by a factor of 2.5, it also diminishes fast retransmit opportunities by 30%, a significant loss.

For $N_d = 2$, we gain 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 $W = 20$ ms 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, changing to TCP's $N_d = 2$ coupled with the $W = 20$ ms delay requires both sender and receiver modifications, increasing the problem of *deploying* the change. However, we could instead consider a similar change requiring only changes to the sender: lower N_d to 2, but on the second dup, wait 20 ms before entering fast retransmission. This change turns out to have virtually identical effects to having the receiver perform the wait. $R_{g:b}$ falls only slightly for $\mathcal{N}_1$, and not at all for $\mathcal{N}_2$, and numerous additional fast retransmission opportunities are gained.

We might then be tempted to apply the same sender-side approach to lowering N_d to 1. This works well for $\mathcal{N}_2$, only diminishing $R_{g:b}$ to 180, still a comfortably high value;[3] but for $\mathcal{N}_1$, we obtain $R_{g:b} = 6.45$, which is unacceptably low.

We note that the TCP *selective acknowledgment* (SACK) option also provides a mechanism for improving TCP retransmission [16], [7], [14]. Use of SACK can be complementary to lowering N_d, since SACK focuses on determining *what* to retransmit rather than *when*.

We observed one other form of dup ack series potentially leading to unnecessary retransmission, which we mention briefly because it is surprisingly common. 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 unnecessary retransmissions 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, and 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, as would a simple heuristic of noting when all of the outstanding data has been acknowledged.

B. Packet Replication

Another form of "pathological" network behavior is *packet replication*, in which the network delivers multiple copies of the same packet. Unlike reordering, it is difficult to see how replication can occur, though perhaps one mechanism is unnecessary link-level retransmissions.[4] In $\mathcal{N}_1$, we observed only one instance of packet replication, in which a pair of acks, sent once, arrived nine times, each copy coming 32 ms after the last. In $\mathcal{N}_2$, we observed 65 instances of replication, all of a single packet, the largest being 23 copies of a data packet arriving in a short blur at the receiver. Since the problem was exceedingly rare in our traces, we omit further analysis here.

C. 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, not payloads. (For "pure acks," i.e., acknowledgment 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 [21]. 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 data sets. 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. This evidence suggests that, as a rule of thumb, the proportion of Internet data packets corrupted in transit is around 1 in 5000.

A corruption rate of 1 packet in 5000 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 5000 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.

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 one corruption out of 300 000 acks in $\mathcal{N}_1$, and, after eliminating `lbli`, one out of 1.6-million acks in $\mathcal{N}_2$. This discrepancy suggests that data packets are much more likely to be corrupted than the small pure ack packets because of some artifact of 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.

We gathered a bit of further evidence concerning corruption rates by sampling traffic on a busy FDDI ring connecting a large university (U.C. Berkeley) to the Internet. In a January 1998 sample of 400 000 packets, one in 7500 had an inconsistent checksum. In a November 1998 sample of 294 million packets, about one in 9500 had an inconsistent checksum. Thus, the problem appears quite real, and is under further investigation [19].

[3] However, as noted above, some network paths have substantial *minimum* reordering times. For today's slower rate paths, these times can well exceed the 20-ms figure we have explored. For such paths prone to reordering, we would expect any approach based on delaying $W = 20$ ms to lead to significant, unnecessary retransmissions, and poor performance. This problem is considerably diminished for $N_d = 2$ because then we must have quite substantial (in terms of time) reordering in order to generate enough dups to falsely trigger fast retransmission.

[4] 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.

In summary, we cannot offer a definitive answer as to overall Internet packet corruption rates, but the evidence that corruption occurs fairly frequently argues for further study in order to resolve the question.

IV. 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 *can* transmit while still preserving network stability. Available bandwidth never exceeds bottleneck bandwidth and can, in fact, be much smaller (Section VI-C).

We will denote a path's bottleneck bandwidth as $\mathcal{B}$. For measurement analysis, it is important to estimate $\mathcal{B}$ because from it we can then estimate a bound on interpacket spacing such that if two packets are sent with less spacing between them, then the second packet will have to queue behind the first at the bottleneck, and thus the transmission delays of the two packets will be correlated, rather than providing independent measurements of delay conditions along the path. If a packet carries a total of b bytes and the bottleneck bandwidth is $\mathcal{B}$ byte/s, then define

$$Q_b = b/\mathcal{B}. \tag{1}$$

From a queueing theory perspective, Q_b is simply the service time of a b-byte packet at the bottleneck link. 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 due to another source (such as additional traffic from other connections, or processing delays). 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.

A. Packet Pair

The bottleneck estimation technique used in previous work is based on "packet pair" [13], [1], [3]. 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 $\mathcal{B}$ via (1).

The principle of the bottleneck spacing effect was noted in Jacobson's classic congestion paper [11], 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 [13]. 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 (Section VI-C). But if, as for us, the goal is to estimate $\mathcal{B}$, 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 [1]. 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 $\mathcal{B}$ [3]. Their work focuses on `bprobe`, a tool they devised for estimating $\mathcal{B}$ by transmitting ten 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 [20], or because some media, such as satellite links, can have significant bandwidth asymmetries depending on the direction traversed [6].

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" (Section VI-A) 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.

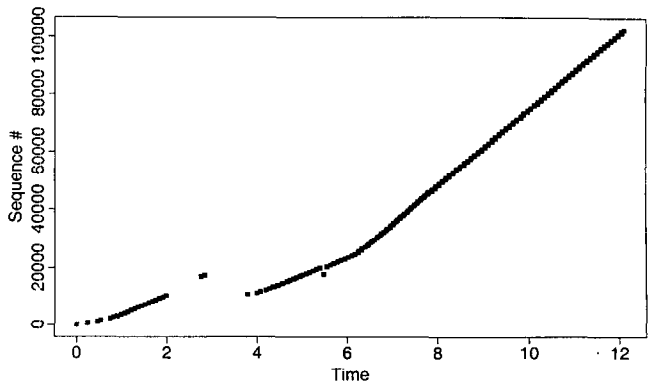

Fig. 2. Bottleneck bandwidth change.

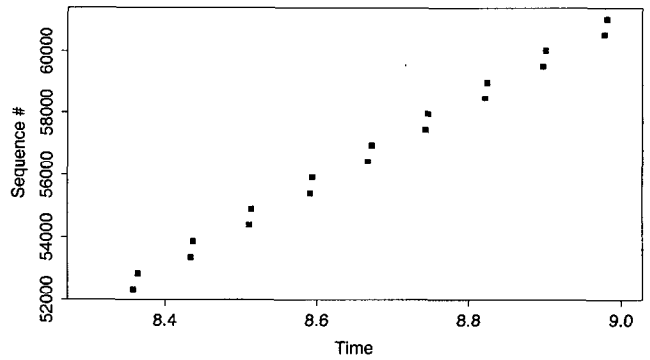

Fig. 3. Enlargement of part of the right half of Fig. 2.

B. Difficulties with Packet Pair

As shown in [1] and [3], packet pair techniques often provide good estimates of bottleneck bandwidth. We find, however, four potential problems (in addition to asymmetries and 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 (Section III-A). 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 $\mathcal{B}$. 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 $\mathcal{B}$. For example, if $C_r = 10$ ms, then for $b = 512$ bytes, a packet pair cannot distinguish between $\mathcal{B} = 51\,200$ byte/s, and $\mathcal{B} = \infty$.

We had several sites in our study with $C_r = 10$ ms. 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. Fig. 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 about 53 kbit/s, while the second is 106 kbit/s, 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 we emphasize that bottleneck shifts can occur due to other mechanisms, such as routing changes or partial layer-2 failures. What is of interest is not that ISDN in particular exhibits this quirk, but the awareness of the general problem that bottleneck bandwidth can indeed change during the course of a connection.

Multichannel Bottleneck Links: A more fundamental problem with packet-pair techniques arises from the effects of *multichannel* links, for which packet pair can yield incorrect overestimates even in the absence of any delay noise. Fig. 3 expands a portion of Fig. 2. The slope of the large linear trend in the plot corresponds to 13 300 byte/s (106 kbit/s) as noted earlier . 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/s. However, this trace involved `lbli`, a site with an ISDN link that has a hard limit of 128 kbit/s = 16 kbyte/s, a factor of ten smaller. Clearly, an estimate of $\mathcal{B} \approx 160$ kbyte/s 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 do not queue behind each other! Multichannel 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 $\mathcal{B}$.

Again, 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

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 $k = 3$ 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.

C. Robust Bottleneck Estimation

We now turn to the question of how we might extend the packet-pair concept to address the difficulties mentioned in the previous section. We term the more robust procedure we developed "packet bunch modes" (PBM). The main observations behind how PBM works is that we can deal with packet pair's shortcomings by forming *receiver-side* estimates for a *range* of packet-bunch sizes, allowing for *multiple* bottleneck values or apparent bottleneck values. Forming estimates at the receiver yields the benefits discussed in Section IV-A. Considering a range of bunch sizes lets us accommodate limited receiver-clock resolutions and the possibility of multiple channels or load-balancing across multiple links by using large bunch sizes. But since we also consider small bunch sizes, we can still minimize the risk of underestimation due to noise diluting the spacing of the bunches. Finally, allowing for finding multiple bottleneck values lets us accommodate multichannel (and multilink) 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:

1) 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*;
2) if we find two strong modes which span the same portion of the connection, and if one is found only for a packet bunch size of m and the other only for bunch sizes $> m$, then we have evidence for an m-channel bottleneck link;
3) we can find both situations, for a link that exhibits both a change and a multichannel link, such as shown in Fig. 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. Examples are: determining how many packet arrivals to consider given limited-clock resolution; propagating clock uncertainties into the bandwidth estimates; dealing with relatively flat modal peaks; merging nearby peaks; avoiding timing artifacts introduced by TCP "self-clocking" and delayed acknowledgment; deciding when we have too few bandwidth measurements for a given bunch size to trust the corresponding estimate; and numerous others.

We defer discussion of these particulars to [22]. 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

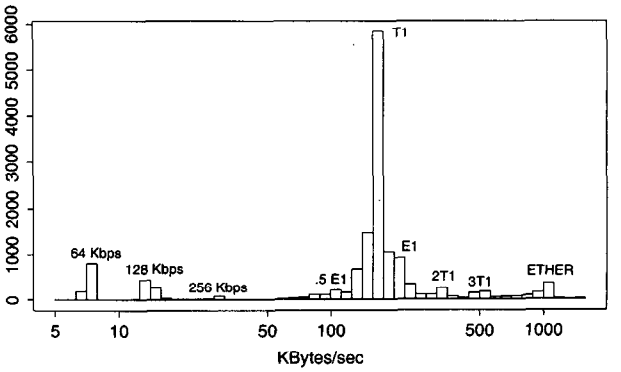

Fig. 4. Histogram of single-bottleneck estimates for $\mathcal{N}_2$.

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 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. After removing `lbli`, which frequently exhibited both bottleneck changes and multichannel 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 multichannel 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 [22], and focus here on the usual case of finding a single bottleneck.

Unlike [3], 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. Fig. 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.) We can arguably identify all of the peaks in the figure as corresponding to known bandwidths such as 170 kbyte/s for a T1 circuit after removing overhead, or possible divisions or pairings of known bandwidths. We find that the E1 peak disappears if we confine the analysis to North American sites, as expected since E1 is used in Europe but not in North America. Since we can offer plausible explanations for all of the peaks, PBM passes the self-consistency test, which 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 [1], [3]. A receiver-based algorithm like PBM, however, can soundly assess the bottleneck in just one direction along the path. Since our data sets include connections in both directions along each path, we can use PBM to estimate the bottleneck bandwidth in each direction, and then compare the two to assess symmetry. This assessment is imperfect, since we do not have simultaneous measurements in each direction, but since the bottleneck bandwidth along a path changes infrequently, we can still compare connections somewhat separated in time.

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 [20]. 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.

D. 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. This restriction might introduce bias by removing traces for which we know a packet pair will perform poorly (such as multichannel links). However, the bias is bounded by the fact that such traces comprise only 2%–5% of all of the traces.

If packet pair produces an estimate lying within $\pm 20\%$ of PBM's, then we consider it to agree with PBM, otherwise not. We evaluate RBPP (per Section IV-A) by considering it as PBM limited to packet bunch sizes of two 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 multichannel effects, then packet pair is a viable and relatively simple means to estimate the bottleneck bandwidth.

We also evaluate the 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 [22]. Even so, we find that SBPP does not work especially well. In both data setss, 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 (Section IV-A).

V. 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 (Section V-A), differences between loss rates of data packets and acks (Section V-B), the degree to which loss occurs in bursts (Section V-C), and how well TCP retransmission matches genuine loss (Section V-D).

A. Loss Rates

A fundamental issue in measuring packet loss is to avoid confusing measurement drops with genuine losses. As mentioned in Section I, we addressed this concern by coding into `tcpanaly` the details of the different TCP implementations in our study, so it could infer measurement drops by detecting apparently inconsistent TCP behavior (such as sending an acknowledgment for data that, according to the trace, never arrived) [21]. 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 $\mathcal{N}_2$, nearly twice as many (5.2%) were lost. However, we need to address the question of whether the increase was due to the use of bigger windows in $\mathcal{N}_2$ (Section II). 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 along the forward path, while the ack transmission rate does not adapt to conditions along the reverse path unless either an entire flight of acks is lost, causing a sender timeout, or there is significant correlation between the loss rates in the forward and reverse directions, which we show in Section V-B is not the case. 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 $\mathcal{N}_2$. We conclude that, overall, packet loss rates nearly doubled during 1995. We can refine these figures 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.

TABLE I
CONDITIONAL ACK LOSS RATES FOR DIFFERENT REGIONS

Region	No-loss$_1$	No-loss$_2$	If-loss$_1$	If-loss$_2$	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%

We will term a connection "loss-free" if it did not experience any lost acks, and "lossy" if at least one of the acks was lost. In both $\mathcal{N}_1$ and $\mathcal{N}_2$, about half the connections were loss-free. For lossy 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.

Geography also plays a crucial role. We partitioned 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. connections have both in the U.S. "Into Europe" connections have U.S. data *receivers* (since they are what transmit the packets of interest, namely the acks, into Europe). Similarly, "Into U.S." are connections with European data receivers.

Table I summarizes loss rates for the different regions, conditioning on whether any acks were lost ("loss-free" or "lossy"). The second and third columns give the proportion of all connections that were loss-free in $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively. We see that except for the trans-Atlantic links going into the United States, the proportion of loss-free 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 all of the lossy connections aggregated together, and the final column summarizes the relative change of these figures. None of the "lossy" 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 (Section V-B). We did not, however, find any sites that seriously skewed the above figures.

In [22], we also analyze loss rates over the course of the day, here omitted due to limited space. We find the expected diurnal pattern of "busy" periods corresponding to working hours and "quiescent" periods to late night and early morning hours. However, we also find that our *successful* measurements involving European sites exhibit a definite bias toward oversampling the quiescent periods, due to effects discussed in Section II. Consequently, the European loss rates given above are underestimates.

B. Data-Packet Loss versus 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 "queued" and "unqueued" data packets. A "queued" data packet is one that presumably had to queue at the bottleneck link behind one of the connection's previous packets, while an unqueued 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 *waiting time*, as follows.

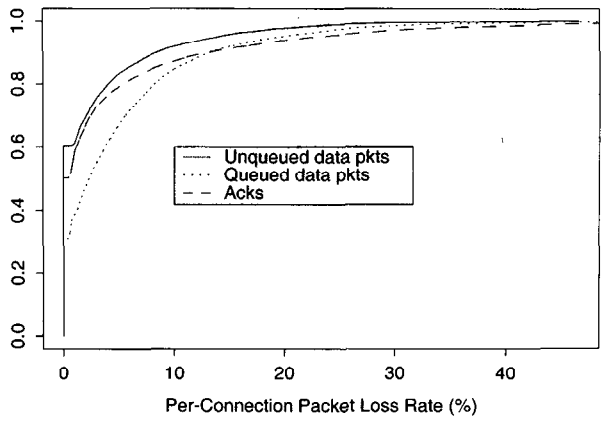

Fig. 5. $\mathcal{N}_2$ loss rates for data packets and acks.

Suppose the methodology in Section IV estimates the bottleneck bandwidth as $\mathcal{B}$. It also provides *bounds* on the estimate, i.e., a minimum value $\mathcal{B}^-$ and a maximum $\mathcal{B}^+$. We can then determine the maximum amount of time required for a b-byte packet to transit the bottleneck, namely: $Q_b^+ = b/\mathcal{B}^-$ s, which is simply (1) using the lower bound on $\mathcal{B}$.

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

$$\lambda_1^+ \coloneqq Q_b^+.$$

Subsequent packets have a waiting time

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

λ_i^+ 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 have 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 $T_i < T_{i-1} + \lambda_{i-1}^+$, then we will term packet i "queued," meaning that it had to wait for pending transmission of earlier packets. Otherwise, we term it "unqueued." (We can also develop "central" estimates rather than maximum estimates using $\mathcal{B}$ instead of $\mathcal{B}^-$ in this chain of reasoning. These are the values used in Section IV-C).

Using this terminology, in both $\mathcal{N}_1$ and $\mathcal{N}_2$, about 2/3 of the data packets were queued. Fig. 5 shows the distributions of loss rates during $\mathcal{N}_2$ for unqueued data packets, queued data packets, and acks. All three distributions show considerable probability of zero loss. We see that queued packets are much more likely to be lost than unqueued packets, as we would expect. In addition, acks are consistently more likely than

unqueued packets to be lost, but generally less likely to be lost than queued packets, except during times of severe loss. We interpret the difference between ack and data loss rates as reflecting the fact that, while an ack stream presents a much lighter load to the network than a data-packet stream, the ack stream does *not* adapt very much to the current network conditions along its forwarding path, 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 unqueued data-packet loss rate we observed was 47%. For queued 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 10 min, 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 (queued and unqueued) 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 routing asymmetry.

A final puzzle is that the nonzero portions of both the unqueued and queued data-packet loss rates agree closely with exponential distributions, while the distribution for acks is not so persuasive a match. Perhaps the better fit for data loss rates reflects the fact that the sender transmits data packets at a rate that adapts to the current network conditions based on observing data-packet loss. The difference highlights that 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 *nonadaptive* sampling, versus measuring loss rates *experienced* by a transport connection's packets—the two can be quite different.

C. 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 [1], 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 to 500 ms. 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 II
UNCONDITIONAL AND CONDITIONAL LOSS RATES

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

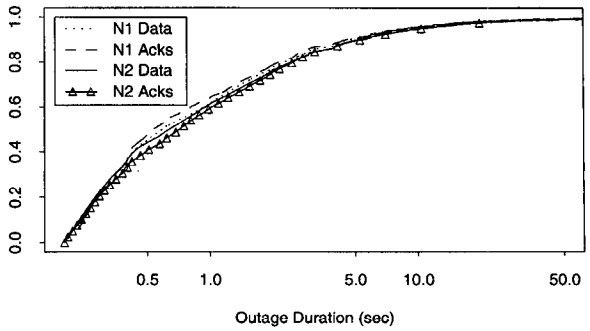

Fig. 6. Distribution of packet loss outage durations exceeding 200 ms.

Table II summarizes P_l^u and P_l^c for the different types of packets and the two data sets. Clearly, TCP packet loss events are not 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 queued 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 unqueued 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]

The relative differences between P_l^u and P_l^c in Table II 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 II, even for $\delta = 500$ ms, 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*. Fig. 6 shows the distribution of outage durations for those lasting more than 200 ms (the majority). We see that all four distributions agree fairly closely.

It is clear from Fig. 6 that outage durations span several orders of magnitude. For example, 10% of the $\mathcal{N}_2$ ack outages were 33 ms or shorter (not shown in the plot), while another 10% were 3.2 s or longer, a factor of 100 larger. Furthermore, the upper tails of the distributions are consistent with Pareto distributions. Fig. 7 shows a complementary distribution plot of the duration of $\mathcal{N}_2$ ack outages, for those lasting more than 2 s (about 16% of all the outages). Both axes are log-scaled. A

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

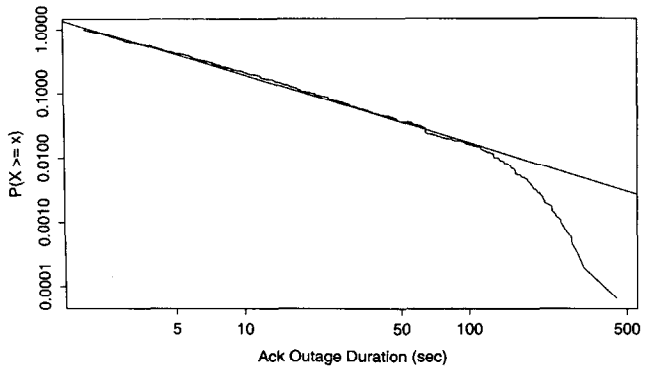

Fig. 7. Log-log complementary distribution plot of $\mathcal{N}_2$ ack outage durations.

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 upper tail, which is subject to truncation because of the 600-s limit on connection durations (Section II).

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 [26], 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 (RED) could significantly affect these patterns and the corresponding connection dynamics [8].

D. 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.

In [21], we identified one TCP implementation as suffering from significant errors in computing RTO, which the other implementations do not exhibit. We removed the corresponding traces from the analysis in this section, as the TCP generated an abnormally large number of RR's.

We classify three types of RR's:

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

TABLE III
PROPORTION OF RR'S DUE TO DIFFERENT CAUSES

Type of RR	$\mathcal{N}_1$	$\mathcal{N}_2$
% all packets	1%	2%
% retrans.	26%	28%
Unavoidable	44%	17%
Coarse feed.	51%	80%
Bad RTO	4%	3%

Because we have traces of connections at both sender and receiver, we can unambiguously determine for each retransmission which acks had arrived or would subsequently arrive at the sender, and so can readily detect RR's and classify them by type. Table III summarizes the prevalence of the different types of RR's in $\mathcal{N}_1$ and $\mathcal{N}_2$. We see that in $\mathcal{N}_1$, a fair proportion of the RR's 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 RR's were unavoidable, the difference no doubt due to $\mathcal{N}_2$'s use of bigger windows (Section II) increasing the mean number of acks in flight.

"Coarse feedback" RR's presumably would all be fixed using SACK, so we see that SACK provides a major benefit in improving TCP retransmission.

"Bad RTO" RR's indicate that the TCP's computation of the retransmission timeout was erroneous. Bad RTO RR's are rare, providing solid evidence that the standard TCP RTO estimation algorithm developed in [11] performs quite well for avoiding RR's. 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 RR's are rare enough to not present serious performance problems.

The last aspects of TCP retransmission we investigate are the patterns of packet loss during fast recovery sequences. Two known problems with TCP loss recovery are that if multiple packets are lost in a single flight, then the recovery is likely to stall until a retransmission timeout occurs, seriously diminishing throughput; and if a retransmitted packet is itself lost, the connection will also incur a timeout [7], [10]. While these problems have been recognized for quite a while, extensive data has not been available in order to gauge the degree to which they actually present difficulties for Internet connections. We analyzed the $\mathcal{N}_1$ and $\mathcal{N}_2$ measurements to provide such data.

In $\mathcal{N}_1$, out of 1178 packets retransmitted using fast recovery, only 3.9% were themselves lost, while in $\mathcal{N}_2$, only 4.5% of 15 444 packets were lost. (These proportions are quite close to the unconditional loss rates we examined in Section V-A, and much lower than the conditional loss rates examined in Section V-C, indicating that *congestion often drains on time scales of RTT's.*) We conclude that the concern of suffering a

timeout due to a lost retransmitted packet is, in practice, not an especially serious problem.

However, in both $\mathcal{N}_1$ and $\mathcal{N}_2$, 1/3 of the time we found that more than one packet was lost in the flight prior to a fast recovery, and about 15% of the time, more than two packets were lost. These proportions are high enough to give solid support for refining the fast recovery mechanism to cope with multiple losses, such as by adding SACK; though we note again, as in Section V-C, that deployment of RED may significantly alter these proportions.

VI. 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 (OTT's) and omit discussion of RTT variation, since RTT measurements conflate delays along the forward and reverse paths.

For reasons noted in Section I, 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 [18]. Since we have about 1000 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 OTT's 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 [23]. 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 [5]. They found that mean OTT's are often *not* well approximated by dividing RTT's in half, and that variations in the paths' OTT's are often asymmetric. Our measurements confirm this latter finding. If we compute the interquartile range (75th percentile minus 25th) of OTT's for a connection's unqueued data packets versus the acks coming back, in $\mathcal{N}_1$ the coefficient of correlation between the two is 0.10, and in $\mathcal{N}_2$ it drops to 0.006.

A. 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. Section IV-A). 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*) [27]. Mogul subsequently analyzed a trace of Internet traffic and confirmed the presence of ack compression [17]. 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 h traffic trace.

Ack compression: A simple metric for detecting ack compression would be to compute $\Delta T_r / \Delta T_s$ which, if less than one, indicates that the packets arrived with their timing compressed. However, we need to take into account the uncertainties in ΔT_r and ΔT_s due to the limited resolution of the clocks used to measure them. Let C_r and C_s be the receiver and sender's clock resolutions, per the discussion in [23]. Then the actual sending spacing is bounded by $\Delta T_s \pm C_s$, and similarly $\Delta T_r \pm C_r$ for the receiving spacing. Therefore, the actual degree of compression ranges from $(\Delta T_r - C_r)/(\Delta T_s + C_s)$ to $\Delta T_r + C_r/\Delta T_s - C_s$.

To be conservative in concluding that a set of packets had compressed timing, we use the latter end of this range. Then, to detect ack compression, for each group of at least three acks, we compute

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

While generally the acks in our traces were generated for every-other data packet, we also included acks sent more frequently, such as duplicate acks.

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 two, 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 [11]. 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 (Section IV-A) can misinterpret compressed acks as reflecting greater bandwidth than truly available. Since, however, we find ack compression relatively rare and small in

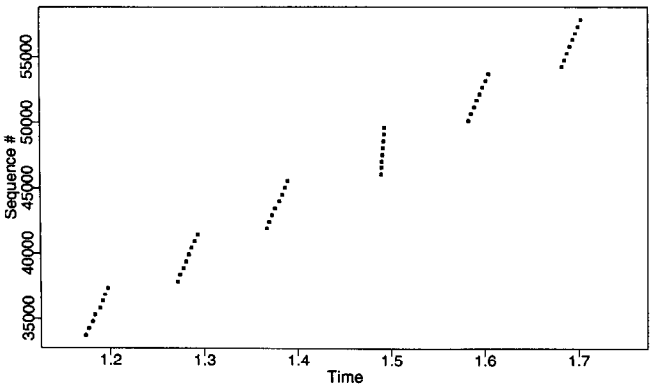

Fig. 8. Data-packet timing compression.

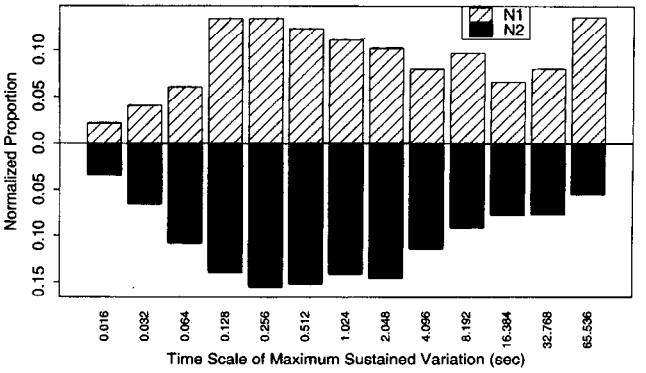

Fig. 9. Proportion (normalized) of connections with given timescale of maximum delay variation ($\hat{\tau}$).

magnitude, the first problem is not serious,[6] and the second can be dealt with by judiciously removing upper extremes from sender-based measurements.

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 (Section IV). If, after the bottleneck, their timing is compressed, then use of (2) will *not* detect this fact unless they are compressed to a greater degree than their sending rate. Fig. 8 illustrates this concern: the flights of data packets arrived at the receiver at 170 kbyte/s (T1 rate), except for the central flight, which arrived at Ethernet speed. However, it was also sent at Ethernet speed, and so $\xi \approx 1$.

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 $\mathcal{B}^+$. 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 in 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.

B. 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 $n_l < n_r/4$ 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 OTT's 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 nonqueueing 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 OTT's, as these are not clouded by queueing at the bottleneck and adaptive transmission rate effects. The question is: are there 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 $2^4, 2^5, \cdots, 2^{16}$ ms to find $\hat{\tau}$, the value of τ for which ΔQ_τ is greatest. $\hat{\tau}$ reflects the time scale for which the connection experienced the greatest OTT variation. Fig. 9 shows the normalized proportion of the connections in $\mathcal{N}_1$ and $\mathcal{N}_2$ exhibiting different values of $\hat{\tau}$. Normalization is done by dividing the number of connections that exhibited $\hat{\tau}$ with the number that had durations at least as long as $\hat{\tau}$. For both data sets, time scales of 128–2048 ms primarily dominate. This range spans an order of magnitude, and exceeds typical RTT values. Furthermore, while less prevalent, $\hat{\tau}$ values all the way up to 65 s remain common, with $\mathcal{N}_1$ having a strong peak at 65 s (perhaps due to periodic outages caused by router synchronization [9], eliminated by the end of 1995).

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

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

C. Available Bandwidth

The last aspect of delay variation we look at is an interpretation of how it reflects the *available bandwidth*. In Section V-B we developed a notion of data packet i's "waiting time," λ_i, meaning the total delay it incurs due to both queueing at the bottleneck behind its predecessors, and the time required for its own transmission across the bottleneck [Q_b, per (1)]. For simplicity, we assume that b is the same for each data packet, though the following discussion can be extended to the case of variable b.

Since every packet requires time Q_b to transit the bottleneck, *variations* in OTT do not include Q_b, but *will* reflect $\psi_i = \lambda_i - Q_b$. ψ_i is the expected additional delay that packet i will experience because it will have to queue behind its predecessors at the bottleneck.

Let γ_i denote the difference between packet i's measured OTT and the minimum observed OTT. Using the same assumptions as in Section VI-B, we interpret γ_i as reflecting queueing delays.

If the network path is completely unloaded except for the connection's load itself (no competing traffic), then we should have $\psi_i = \gamma_i$. That is, the *measured* extra delay (γ_i) can all be accountted for by the *expected* extra delay due to i queueing behind its predecessors.

More generally, define

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

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 overall, we have a situation approximating $\psi_i \approx \gamma_i$, namely all of the delay variation is due to the connection's own queueing load on the network. On the other hand, if $\beta \approx 0$, then the delays experienced by the packet are much higher than those simply due to their own transmission times across the bottleneck and their own queueing behind their predecessors. In this case, the connection's load is *insignificant* compared to that of other traffic in the network.

Similarly, we can say that $\Sigma_i(\psi_i + Q_b)$ reflects the resources consumed by the connection itself, while $\Sigma_j(\gamma_j + Q_b) - \Sigma_i(\psi_i + Q_b) = \Sigma_j\gamma_j - \Sigma_i\psi_i$ 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 one mean that the entire bottleneck bandwidth was available, and values close to zero 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, unlike other available bandwidth estimation techniques [15], [4]. Fully evaluating this possibility remains for future work.

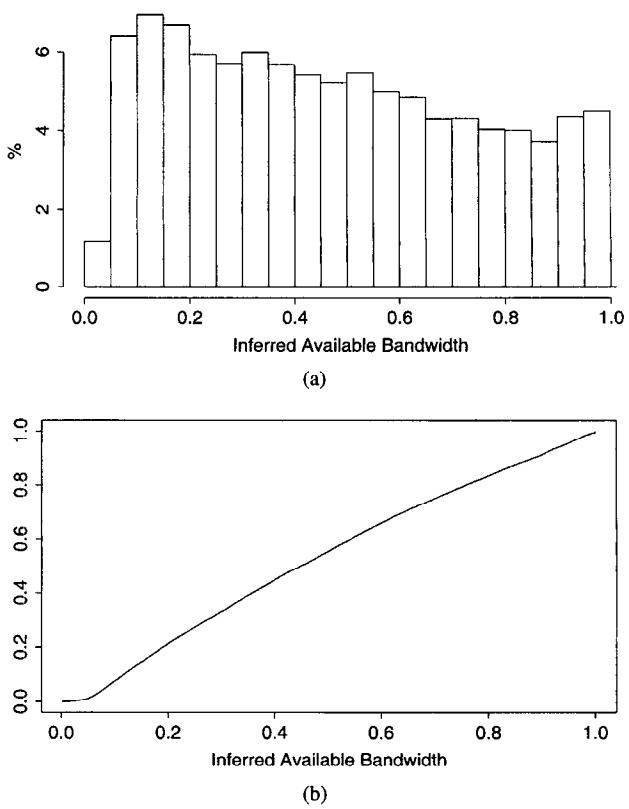

Fig. 10. Density and cumulative distribution of $\mathcal{N}_2$ inferred available bandwidth (β).

We can roughly 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.

Fig. 10 shows the density and cumulative distribution of β for $\mathcal{N}_2$. We find that Internet connections encounter a broad range of available bandwidth.[7] 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.

[7] The depressed density at $\beta \approx 0$ reflects a measurement bias [22].

VII. Conclusion

We analyzed packet traces of 20 800 TCP connections between 35 Internet sites in an attempt to characterize the spectrum of packet dynamics observed along Internet paths. Our analysis of "pathological" network behavior found that packet reordering is surprisingly common, with 36% of the 100 kbyte connections in one data set, and 12% in the other, experiencing at least one reordered packet. Reordering varies considerably from site to site, and while it sometimes occurs in groups as large as dozens of packets, it usually involves only one or two packets and is correlated with routing fluctuations. The timing differences leading to reordering are such that the TCP fast retransmission threshold could be safely lowered from three duplicate acks to two by introducing a 20-ms wait before retransmitting, increasing the fast retransmission opportunities by 2/3.

Our assessment of bottleneck bandwidth uncovered four difficulties with the common "packet pair" approach: out-of-order delivery, limited clock resolution, bottleneck bandwidth changes, and multipathing. We sketched a robust estimator, Packet Bunch Modes (PBM's), to address these difficulties, and then, with it as a reference, found that receiver-based packet-pair estimation works very well, agreeing with PBM 97%–98% of the time, while sender-based packet pair agrees only about 60% of the time.

We found that packet loss rates nearly doubled during the course of 1995, with most of the increase coming from larger loss rates during the same busy periods, rather than longer busy periods. There are considerable geographic differences in loss rates, with European and especially the trans-Atlantic paths having higher rates than the U.S. loss rates along the forward and reverse directions of a network path show little correlation. We found that loss patterns of data packets differ significantly from those of acks, which appears to be due to the fact that the rate at which data packets are sent adapts to current network conditions in an attempt to diminish the experienced loss rate, while the rate of ack packets adapts much less. Loss is not well modeled as independent, with the likelihood that a packet is lost increasing by an order of magnitude if its predecessor was lost. Sustained loss "outages" have heavy-tailed durations with a Pareto-shape parameter of $\alpha = 1.06$, indicating infinite variance which could contribute to finding self-similar traffic behavior. Finally, we find that when correctly implemented the TCP retransmission algorithms perform well in terms of avoiding unnecessary retransmissions, if coupled with the use of SACK.

Our analysis of packet delay found that while 50%–60% of the connections experienced at least one timing compression event, such events tend to be isolated and small in magnitude, so their impact is minor. We made a preliminary assessment of delay variation time scales, finding that variations occur primarily on time scales of 0.1–1 s, but not infrequently extend out to much larger time scales. Our assessment of available bandwidth was conducted in terms of gauging the degree to which a connection's own load along a path compared to the total load along the path, finding that this ratio varied fairly evenly all the way from the connection's load being insignificant to the connection's load being the entire load along the path.

Finally, our study has implications for several measurement considerations.

- With due diligence to remove packet filter errors, 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.
- 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 [2], or general Internet performance metrics such as those in development by the IPPM effort [24]—the extra complications incurred by coordinating sender and receiver yield significant benefits.

Acknowledgment

This work would not have been possible without the efforts of the many volunteers who installed the Network Probe Daemon (NPD) at their sites. The author is 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. The author is also indebted to K. Bostic, E. Nemeth, R. Stevens, G. Varghese, A. Albanese, W. Holfelder, and B. Lamparter for their invaluable help in recruiting NPD sites. The author extends gratitude to P. Danzig, J. Mogul, and M. Schwartz for feedback on the design of NPD, and to L. Rizzo for stimulating discussions concerning lowering the duplicate ack threshold for fast retransmission.

This work greatly benefitted from discussions with D. Ferrari, S. Floyd, V. Jacobson, M. Luby, G. Minshall, J. Rice, and the comments of the anonymous referees. The author expresses his heartfelt thanks.

References

[1] J.-C. Bolot, "End-to-end packet delay and loss behavior in the Internet," in *Proc. SIGCOM'93*, pp. 289–298.

[2] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proc. SIGCOM'94*, pp. 24–35.

[3] R. Carter and M. Crovella, "Measuring bottleneck link speed in packet-switched networks," Comput. Sci. Dept., Boston Univ., Boston, MA, Tech. Rep. BU-CS-96-006, Mar. 1996.
[4] ——, "Dynamic server selection using bandwidth probing in wide-area networks," Tech. Rep. BU-CS-96-007, Comput. Sci. Dept., Boston Univ., Boston, MA, Mar. 1996.
[5] K. Claffy, G. Polyzos, and H.-W. Braun, "Measurement considerations for assessing unidirectional latencies," *Internetworking: Res. and Experience*, vol. 4, no. 3, pp. 121–132, Sept. 1993.
[6] R. Durst, G. Miller, and E. Travis, "TCP extensions for space communications," in *Proc. MOBICOM'96*, pp. 15–26.
[7] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP," *Comput. Commun. Rev.*, vol. 26, no. 3, pp. 5–21, July 1996.
[8] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, pp. 397–413, Aug. 1993.
[9] ——, "The synchronization of periodic routing messages," *IEEE/ACM Trans. Networking*, vol. 2, pp. 122–136, Apr. 1994.
[10] J. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," in *Proc. SIGCOM'96*, pp. 270–280.
[11] V. Jacobson, "Congestion avoidance and control," in *Proc. SIGCOM'88*, pp. 314–329.
[12] V. Jacobson, C. Leres, and S. McCanne. (1989). `tcpdump`. [Online]. Available FTP: ee.lbl.gov
[13] S. Keshav, "A control-theoretic approach to flow control," in *Proc. SIGCOM'91*, pp. 3–15.
[14] M. Mathis and J. Mahdavi, "Forward acknowledgment: Refining TCP congestion control," in *Proc. SIGCOM'96*, 281–291.
[15] ——, "Diagnosing Internet congestion with a transport layer performance tool," in *Proc. INET'96*.
[16] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," RFC 2018, Oct. 1995.
[17] J. Mogul, "Observing TCP dynamics in real networks," in *Proc. SIGCOM'92*, pp. 305–317.
[18] A. Mukherjee, "On the dynamics and significance of low frequency components of Internet load," *Internetworking: Res. and Experience*, vol. 5, pp. 163–205, Dec. 1994.
[19] C. Partridge, private communication, 1998.
[20] V. Paxson, "End-to-end routing behavior in the Internet," *IEEE/ACM Trans. Networking*, vol. 5, pp. 601–615, Oct. 1997.
[21] ——, "Automated packet trace analysis of TCP implementations," in *Proc. SIGCOM'97*, pp. 167–179.
[22] ——, "Measurements and analysis of end-to-end Internet dynamics," Univ. California, Berkeley, CA, Ph.D. dissertation, Apr. 1997.
[23] ——, "On calibrating measurements of packet transit times," in *Proc. SIGMETRICS'98*, pp. 11–21.
[24] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, "Framework for IP performance metrics," RFC 2330, May 1998.
[25] K. Thompson, G. Miller, and R. Wilder, "Wide-area Internet traffic patterns and characteristics," *IEEE Network*, vol. 11, pp. 10–23, Nov./Dec. 1997.
[26] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson, "Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level," *IEEE/ACM Trans. Networking*, vol. 5, pp. 71–86, Feb. 1997.
[27] L. Zhang, S. Shenker, and D. Clark, "Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic," in *Proc. SIGCOM'91*, pp. 133–147.

Vern Paxson received the M.S. and Ph.D. degrees in computer science from the University of California at Berkeley.

He is a Senior Scientist at the AT&T Center for Internet Research at the International Computer Science Institute, Berkeley, CA. He is also a Staff Scientist at the Lawrence Berkeley National Laboratory, Berkeley, CA. His research efforts focus on Internet measurement and network intrusion detection.