Chapter 16

Packet Delay

The final aspect of Internet packet dynamics we analyze is that of packet delay. Delay variation is arguably the most complex element of network behavior to analyze—with loss, for example, the packet either shows up at the receiver or it does not, while with delay there are many shades of possibility and meaning in the time required for a packet to arrive. Likewise, delay variation is potentially the richest source of information about the network, as one of the principle elements contributing to delay is queueing within the network, which is of vital importance in understanding how network capacities evolve over time.

Any accurate assessment of delay must first deal with the issue of clock accuracy, as all delay measurement stems from clock measurements. Unless we tightly calibrate the clocks used for delay measurement, or, equally important, recognize which clocks cannot be well calibrated and discard the corresponding measurements, we cannot know that the subsequent analysis reflects true network behavior and not spurious or misleading clock artifacts. It was these considerations that led us to the lengthy efforts developed in Chapter 12.

We proceed as follows. In § 16.1 we briefly discuss round-trip time (RTT) variation in our measurements, which plays a central role in transport protocol behavior. From the point of view of network path analysis, however, a packet's one-way transit time (OTT) is more fundamental, particularly since RTT measurements conflate delays along the forward and reverse path. Consequently, we devote the remainder of the chapter to OTT analysis. In § 16.2, we discuss OTT variation in large-scale terms. We then in § 16.3 turn to packet timing *compression*—network events in which a group of packets arrive at the receiver more closely spaced together than when they were sent. Compression is a significant event because it introduces potentially misleading discrepancies between the timing of events at the sender and at the receiver, clouding the ability of one endpoint to assess conditions perceived at the other. In § 16.4 we then tackle estimation of the amount of queueing packets encounter during their transit. We attempt to determine the *time scales* associated with queueing, but find wide variation. Finally, in § 16.5 we look at the relationship between queueing delays and *available bandwidth*—the transfer rate the network can sustain for a connection, given the network's current load.

16.1 RTT variation

16.1.1 The role of RTTs

A transport connection's round-trip time (RTT) plays a central role in the connection's behavior. First, a reliable transport protocol such as TCP needs to decide how long to wait for an acknowledgement of data it has sent before retransmitting the data. There is a basic tension between wanting to wait long enough to assure that the protocol does not retransmit unnecessarily, versus not wanting to wait too long so as to unduly delay the connection when in fact retransmission is needed. Our analysis of the Solaris 2.3/2.4 TCP in § 11.5.10 highlights how unfortunate it can be to err on the side of retransmitting too quickly. Network researchers have made considerable efforts in studying how to set a connection's retransmission timeout (RTO), and early problems with TCP's RTO computation identified by Zhang [Zh86] have for the most part been rectified by the work of Karn and Partridge in eliminating ambiguous RTT measurements [KP87], and by that of Jacobson in introducing exponentially-weighted moving averages to estimate both RTT and its variance [Ja88].

The second way in which a connection's RTT influences the connection's behavior concerns the important notion of *bandwidth-delay product* (BDP). A connection's BDP is the product of ρ_A, the available bandwidth, measured in bytes/sec, with τ, the RTT, measured in seconds. The result is a number $B = \rho_A \cdot \tau$ of bytes indicating how much data the connection must have in flight to fully utilize the available bandwidth. A simple way to understand this relationship is to consider that, to fully utilize the available bandwidth, the connection must send ρ_A bytes every second, and thus it must send $\rho_A \cdot \tau$ bytes every round-trip time in order to achieve this goal. A round-trip time, however, exactly corresponds to one cycle of send-and-receive feedback. This relationship, in turn, is directly reflected in the connection's *window* (§ 9.2.2)—the current window controls how much data the connection can have in flight at any given moment, and the window can only change due to feedback for the currently in-flight packets after one RTT has elapsed, since no feedback can arrive sooner than that. Thus, B gives the size of the window the connection must use to fully utilize a bandwidth of ρ_A.

We must, however, make a crucial distinction between these two different roles of RTT in a connection's behavior. For the first role, regulating retransmission, the RTT of interest is how long it might take for a packet to reach the receiver and the corresponding acknowledgement to return to the sender—the *maximum* RTT. For the second role, the RTT of interest is the *minimum* time required for packets to traverse the network path to the receiver and for acks to return. The *larger* values possibly observed for the *actual* amount of time required in general reflect *queueing* along the network path. It does *not* improve a connection's throughput to use such a larger RTT when computing B; it instead only adds to queueing along the path. This observation motivated the development of TCP Vegas [BOP94], in which a significant increase in measured RTT is interpreted as due to using too large a window and adding to queueing along the path, and thus calling for a decrease in the window size to diminish the queueing.

16.1.2 RTT measurement considerations

When discussing RTT times, we must bear in mind that larger packets require larger transmission times, proportional to the bottleneck bandwidth. The effect, naturally, is most apparent on slow links. Accordingly, we need to make sure we do not confuse RTT variation due to packet size with RTT variation due to queueing.

Another consideration is that, if we measure RTT as simply the difference in time between when a packet is sent and when a corresponding reply returns, then we will include in the measurements "response delays" at the receiver (§ 11.6.4). For many purposes, doing so is appropriate, since the roles played by RTT above both concern quantifying the *feedback* time scale, and this includes both the network's delays and those of the receiver. If, however, we wish to discuss only the network's contribution to the feedback time scale, then we need to deduct the response delay from the measured RTT. `tcpanaly` can do this since it knows how to pair packets with their responses. However, we argue that the network's contribution to delay is best studied in terms of one-way transit times (OTT), since doing so allows for the possibility of asymmetries along the two directions of the network path, which we find in § 16.2.3 are in fact common. So, for our RTT analysis, we do not deduct response delays from the measurements, that we might study the entire "closed loop."

Finally, we note that RTT can be measured in two different ways: as the amount of time elapsed between when a TCP sends a packet and when it receives an acknowledgement in response to that packet, or as the time between when a TCP sends an acknowledgement and when it receives the packet liberated by that acknowledgement (§ 11.3.1). As we might expect, overall we find these two values to be very close to one another, except for variations due to "response delays" (§ 11.6.4). (They also can appear different if the clocks at the sender and receiver run at significantly different rates, per § 12.7.7.) In the remainder of this section, we confine our analysis to RTTs measured at the sender.

16.1.3 RTT extremes

Extremes of network behavior are always interesting to consider, since they sometimes challenge the assumptions made by our mental models of how networks "really" work. For example, some might find RTTs larger than a few hundred milliseconds exceedingly unlikely—where could a packet spend all that time?—and thus best treated as pathological events rather than part of the regime we must accommodate as "normal." (We saw how dangerous this can be in Figure 11.9.)

Our data is inappropriate for exploring the full range of RTTs one finds in the Internet, since the set of sites in our study is small, and we would expect RTT extremes to be governed for the most part by geography. This is especially the case for network paths that include satellite links, as these can add hundreds of milliseconds due to the propagation delays up to and back down from the satellite.

However, while geography certainly dominates upper RTT extremes, it is not the only factor. To our surprise, we found that one site in our study, `oce`, experiences extremely high delays for many of its connections. 50% of its connections had a minimum RTT of over 1 sec.

`oce` is sited in the Netherlands. One striking connection came from `wustl` in North America. It never observed an RTT less than 4.4 sec![1] Another came from `unij`, never experiencing an RTT below 2.3 sec—yet `unij` is also in the Netherlands! A `traceroute` from `unij` to `oce` reveals that the route stays wholly within the Netherlands. Furthermore, it shows that all of the delay occurs at the hop between NLNet, the Netherlands Internet backbone, and the `oce` site itself. The

[1] Alas, `wustl` is a Solaris 2.4 site. Its RTO timer had great difficulty accommodating the large RTT, per Figure 11.9. During the first minute of the connection, before the timer finally adapted, it sent 31 new data packets and 51 retransmissions, all but one unnecessary. One packet was retransmitted seven times!

cause of this large delay, which we discussed in § 12.7.8, remains unexplained, despite investigative efforts by staff at the `oce` site. It highlights, however, how commonplace—and often correct—assumptions concerning network behavior can be violated in unexpected ways.

Even after eliminating `oce`, we still find some striking RTT extremes. Connections involving `austr2` experienced minimum RTTs as high as 1.85 sec (to a host in California).[2] If we remove `austr2`, then, curiously, the next highest extremes involved not international traffic but connections with both endpoints in the United States. One, from `wustl` to `adv`, never saw an RTT lower than 1.2 sec, even though a connection ten minutes earlier had a minimum of 156 msec, and one 25 minutes later was back to the typical value of 47 msec. Unfortunately, we do not have a `traceroute` measured right at the time of the anomalous connection. Ones fifteen minutes earlier and 80 minutes later show no anomalies and both report an RTT of about 44 msec.

The most extreme RTT connection in $\mathcal{N}_1$ involved not `korea`, for which we might expect high RTTs (and, indeed, it had plenty), but `nrao` and `bsdi`, in Virginia and Colorado. This connection had a minimum RTT of 1.4 sec and a median value of 2.1 sec. While in § 6.9 we gave an example of a circuitous route involving `bsdi`, `traceroute` reported its RTT as only about 160 msec, much less than observed by this connection; so, we do not have an explanation for what took the packets so long.

So far in this section we have focussed on the *minimum* RTT observed during a connection, which is important for correctly determining B, the bandwidth-delay product. For computing RTO, the connection's retransmission timeout, we instead are interested in the *maximum* RTT, which we now look at briefly. (As discussed in § 15.6, we do not undertake a detailed analysis of how we might modify TCP's RTO algorithms to increase their performance, as this is a complex problem.)

We would expect that RTT maxima can rise very high for connections with slow bottleneck links and many available buffers at the bottleneck. In such cases, the sending TCP will not receive a packet loss signal until it has exhausted the available buffer. For a slow link, a significant amount of buffer can translate into a huge delay as packets finally wend their way through the queue.

The largest apparent RTT we ever observed was 23.8 sec, for a SYN packet and its accompanying SYN-ack. This was not, however, a true RTT: the receiving SunOS 4.1 TCP generating the SYN-ack was retransmitting it in an attempt to establish the connection, and its timer backed off first to 6 sec and then to 24 sec. At the same time, the sender, also a SunOS 4.1 TCP, was backing off its retransmission timer for the original SYN. The two timers were slightly out of phase. Consequently, just before the sender reached the 24 sec retransmission, a SYN-ack arrived from the receiver, leading to the huge apparent RTT. We mention this anomaly because some modifications to TCP such as Hoe's in [Ho96] suggest using the RTT timing for the SYN packet as a quick estimate of the path's true RTT. Such schemes must take care not to get fooled by SYN-ack retransmissions. In this particular case, use of Karn's algorithm would have discarded the RTT measurement as ambiguous [KP87]. However, had the retransmitted SYN-ack arrived just before the *first* retransmission of the SYN (i.e., just before the 6 sec timer expired), then even Karn's algorithm would have accepted the measurement, since the algorithm is predicated upon the assumption that acks are not retransmitted. Finally, we note that Hoe's scheme uses the RTT to estimate B, the bandwidth-delay product. Using a value of 6 sec instead of the correct value of 220 msec would grossly overestimate B, leading to the connection overestimating the window it should use. Hoe's scheme, however, could be easily modified to use a more robust initial RTT estimate, since it does not make any decisions based on

[2] `austr2`, alas, is also a Solaris 2.4 site . . .

B until it has received a flight of 3 closely spaced acks. At that point, there should have been ample opportunity to estimate RTT better.

Putting aside anomalies due to SYN-ack retransmissions, we find that the largest true RTT in our study was 15.1 sec, for a connection involving `oce`. We discussed above `oce`'s peculiarly large RTTs, and in § 12.7.8 the puzzling interplay between the transit times of packets and acks in its connections, so we will not further analyze `oce`-connection RTTs here. If we eliminate `oce`, then we find the next largest RTT comes from the 12-second packet reordering event discussed in § 13.6. Putting aside this pathology, we finally find a "normal" extreme RTT, not due to any unusual network dynamics, of 7.9 sec (involving a connection to `lbli`, which has a low-speed Internet link with a lot of buffer space). A few others range above 6 sec, including one from a high-speed connection between `sintef2` in Norway and `austr` in Australia.

16.1.4 RTT variation during a connection

Another way to characterize RTT extremes is in terms of the variation we observe in RTT over the course of a connection. Our interest lies in whether we can develop a "rule of thumb" such as "it is rare to observe a maximum RTT more than double the minimum RTT." This sort of empirical finding would aid in considering how transport protocols can best adapt to network conditions.

We first note that connections with slow bottlenecks can often experience great swings in RTT as their own packets pile up at the queue for the bottleneck (§ 16.1.3). While such connections are an important consideration for general-purpose transport protocols, for our purposes we eliminate any connection with an estimated ρ_B less than 100 Kbyte/sec, so that we might focus on RTT variations not heavily dominated by the connection's own behavior. We also eliminate connections between `sintef1` and `sintef2`, as they are sited very close together and thus much more easily exhibit large relative swings in RTT, even though in absolute terms the swings are quite small.[3]

After these eliminations, in $\mathcal{N}_2$ we are left with 12,486 connections. Figure 16.1 shows the distribution of the ratio between their maximum RTT and minimum RTT, log-scaled. We compute RTTs from the TCP sender's perspective, using the time required to receive an acknowledgement for a full-sized packet.

The distribution shows great variation, with a median ratio of 2.2:1 (mean of 3:1), but the upper 5% have ratios of 6.7:1 and higher. The entire upper 50% fits closely to a Pareto distribution with $\alpha = 2.1$, shown with a log-log complementary distribution plot in Figure 16.2 (we discussed these plots in § 15.3). A value of $\alpha > 2$ means that the ratio has finite variance, and this is probably due to the fact that the maximum RTT is bounded by the amount of buffer space available along the network path. However, the great degree of variation means that, without additional information, we cannot accurately predict the relationship between the minimum and maximum RTTs.

The ratios exhibit one other striking distribution. If we instead consider the ratio of the *minimum* RTT to the *maximum*, then the corresponding distribution is very nearly normal. Figure 16.3 shows this distribution, with a normal distribution fitted to the mean and variance shown by a dotted line. Figure 16.4 shows a Q-Q plot of the same fitted normal, with the line corresponding to slope 1 and offset 0. Clearly, the agreement is quite good except in the tails. Unfortunately, an

[3]The pair `lbl` and `lbli` do not exhibit this problem because `lbli`'s low-bandwidth ISDN link leads to fairly large RTTs between the two sites.

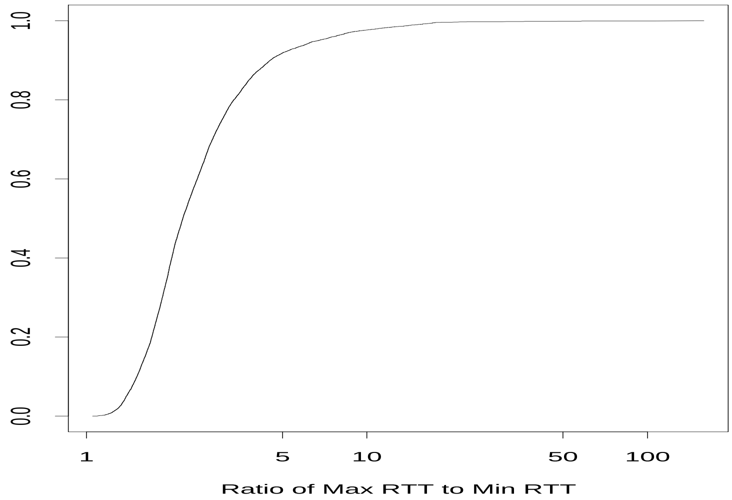

Figure 16.1: Distribution of the ratio between a connection's maximum RTT to minimum RTT

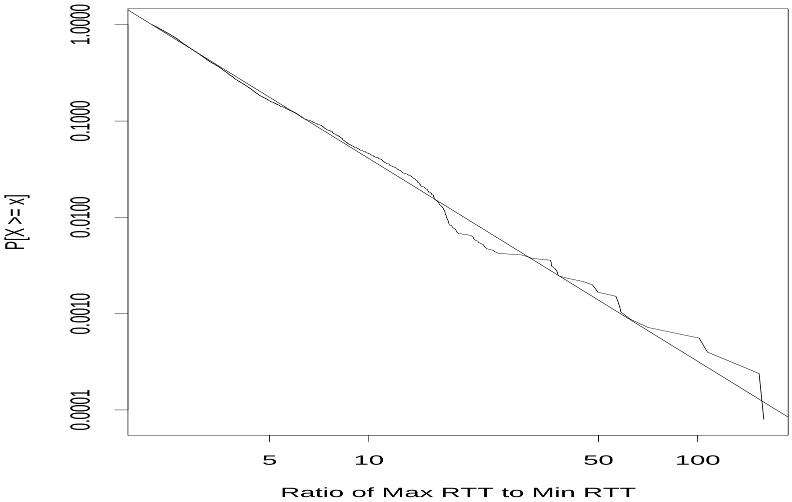

Figure 16.2: Log-log complementary distribution plot of max-min RTT ratio

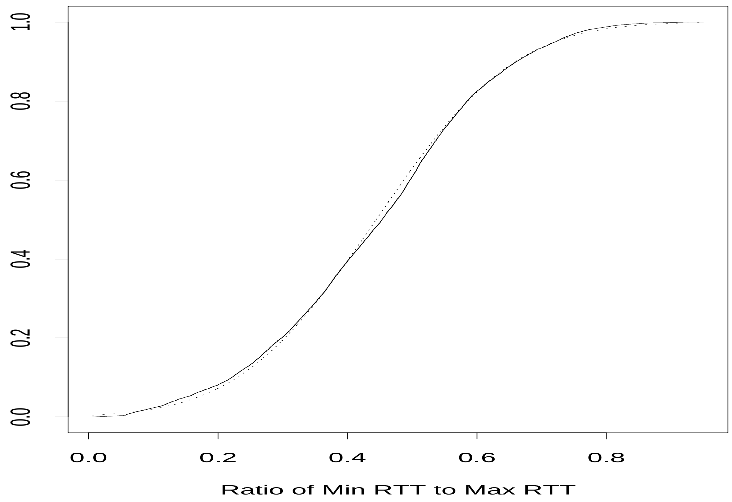

Figure 16.3: Distribution of inverse ratio (minimum RTT to maximum RTT)

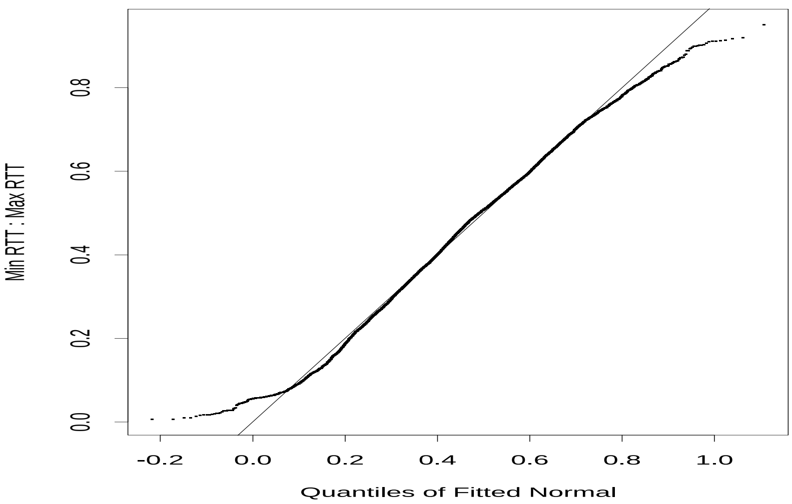

Figure 16.4: Q-Q plot of ratio of minimum RTT to maximum RTT (y-axis) versus fitted normal distribution (x-axis)

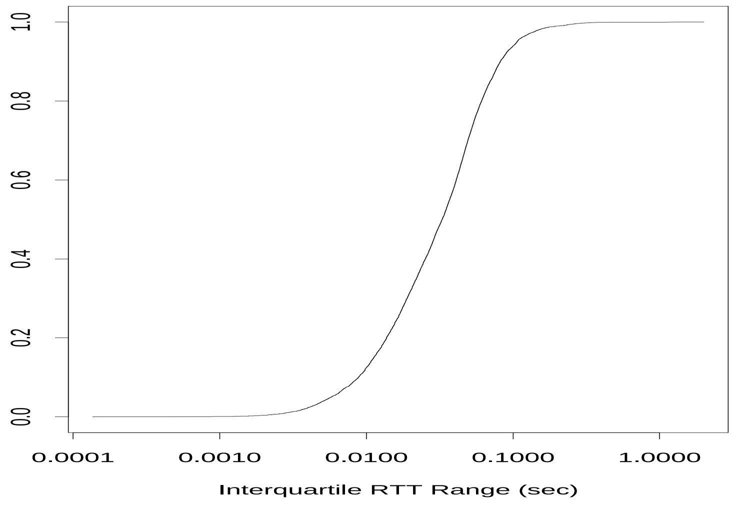

Figure 16.5: Distribution of RTT interquartile range

interpretation for this fit (or for the corresponding Pareto fit) eludes us. As with the elusive exponential fit to data packet loss rates (§ 15.2), we mention the fit here in hopes that it might stimulate further research.

We finish with a look at less extreme RTT variation: the interquartile range (75th percentile minus 25th percentile), IQR. This range gives a much more robust statistic in the sense of being insensitive to extreme values. We are particularly interested in IQR as an aid in estimating the maximum RTT, as this has immediate applications for computing retransmission timeouts (RTOs).

Figure 16.5 shows the distribution of IQR, and Figure 16.6 shows the distribution if we normalize to the minimum RTT. Both plots use a logarithmic scale on the x-axis. We see a wide range of variation, with the lower and upper 5% tails of the absolute range spanning 6 msec up to 106 msec, and, with normalization, the same tails range from a factor of 0.046 up to a factor of 1.23.

The interquartile range is in many ways analogous to a robust version of standard deviation [Ri95]. Consequently, we interpret the wide range of variation as supporting the argument that RTT estimation (for purposes of computing timeouts, for example) must include a notion of variation in addition to estimating the mean or minimum value. Jacobson's estimator does exactly this for TCP [Ja88].

In Figure 16.2 we found that maximum RTTs often are much larger than minimum RTTs. We might wonder, though, whether this discrepancy can be reduced if expressed in terms of RTT variation. For example, it could be the case that the maximum is generally less than n times IQR above the minimum. Unfortunately, this does not appear to be the case. Figure 16.7 shows the distribution of the difference between the maximum and minimum RTT, normalized by dividing by IQR. Again, the x-axis is scaled logarithmically, indicating a wide range of variation. Furthermore, normalization has diminished but not eliminated the Pareto distribution for the upper tail. Instead of occupying a full 50% of the distribution, it now occupies the upper 20%, with $\alpha = 1.84$, within the domain of infinite variance. Finally, these results do not change appreciably if we look at the

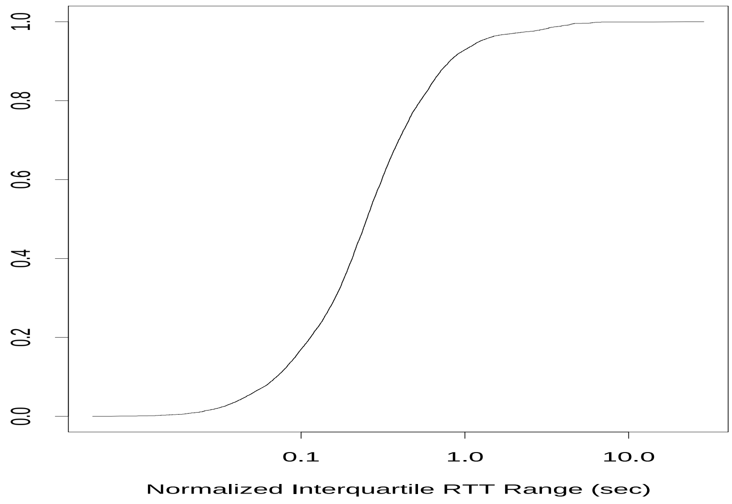

Figure 16.6: Distribution of RTT interquartile range, normalized to minimum RTT

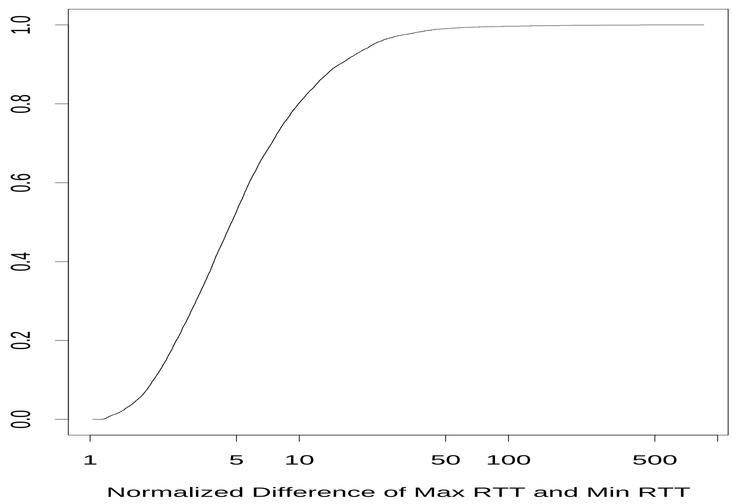

Figure 16.7: Distribution of difference between maximum RTT and minimum RTT, normalized by interquartile range

normalized difference between the maximum RTT and the *median* RTT, rather than the minimum RTT.

From Figure 16.7 it appears that the combination of minimum RTT and interquartile range is inadequate for estimating maximum RTT. TCP RTO estimation is based on similar information, i.e., the estimated RTT mean and standard deviation. Yet, we should not conclude from this that TCP's estimation algorithm cannot work, because the algorithm *updates* its estimates as the connection progresses, using exponentially-weighted moving averages to incorporate new information. Consequently, it has opportunities to *adapt*, while the preceding analysis is *static*. Again, as discussed in § 15.6, we do not undertake here a detailed analysis of how well TCP's RTT estimation algorithm performs, as doing so involves a number of subtle issues.

16.2 OTT variation

For the remainder of this chapter, we focus on one-way transit times (OTTs). Any accurate assessment of delay must first deal with the issue of clock accuracy, from which all delay measurement stems. This problem is particularly pronounced when measuring OTTs since doing so involves comparing measurements from two separate clocks. It was primarily to this end that we undertook the efforts described in Chapter 12 attempting to assure that we can soundly gauge the trustworthiness of the packet timestamps. The subsequent analysis we discuss was always done after first using these algorithms to reject or adjust traces with clock errors.

OTT variation was previously analyzed by Claffy and colleagues in a study of four Internet paths [CPB93a]. They found that mean OTTs are often *not* well approximated by dividing RTTs in half, and that variations in the paths' OTTs are often asymmetric. From our data we cannot confirm their first finding, but we discuss the asymmetry finding shortly.

16.2.1 Why we do not analyze OTT extremes

We do not investigate extreme OTT variation, as we did for RTTs in § 16.1.3, for two reasons. First, most of the RTT extremes are due to network delays, and, in particular, extreme OTTs, so the OTT results are very similar to the RTT results.[4] Second, our absolute OTT values were derived using the approximation that we could rectify clocks in our study by dividing RTTs in half (Eqn 12.5 in § 12.5.1). We know from the Claffy et al. study, and from our earlier results on routing asymmetries (§ 8), that this approximation is often erroneous, and we noted in the derivation that consequently we must refrain from analyzing the absolute OTT values themselves.

16.2.2 Range of OTT variation

Our measurements do, however, let us accurately assess *variations* in OTT. In doing so, we will always distinguish between ack OTTs and data packet OTTs, as we expect the latter to show significantly more variation due to their queueing load. Figure 16.8 shows the distributions of IQR and max-min variations in OTTs for $\mathcal{N}_2$ data packets and acks. Again, we have limited our

[4]This would not have been the case if RTT extremes were due to delays by the TCP endpoint, or combined increases in delay along the two directions of the network path. But neither of these is the dominant effect.

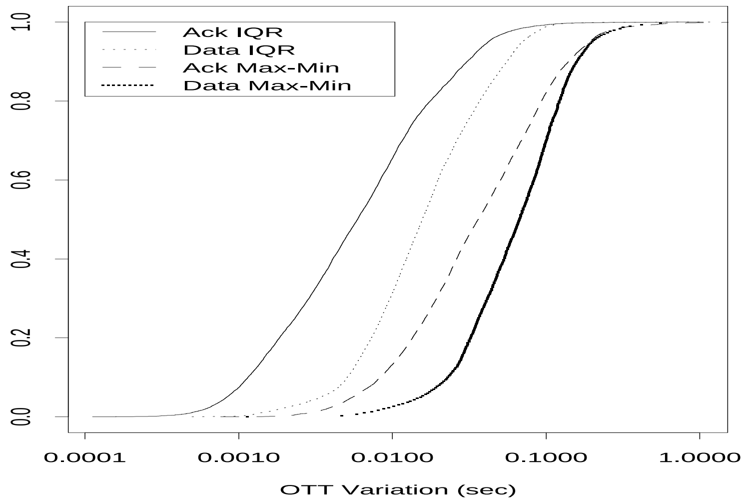

Figure 16.8: Distribution of interquartile and max-min OTT variation

analysis to connections with a bottleneck bandwidth exceeding 100 Kbyte/sec, and have removed those between `sintef1` and `sintef2`.

The x-axis reflects logarithmic scaling; so, as with many aspects of RTT variation, we see a wide range of variation. For example, for data packets the median ratio between the max-min variation and IQR is 3.5:1, and the upper 5% tail exceeds 13:1. For acks, the numbers are higher, the median being 5:1 and the upper 5% tail at 29:1. The difference lies in data packets having a larger IQR to begin with, due to OTT variation caused by the connection's own queueing; for acks, IQR is fairly tame, so the same absolute OTT extreme will be relatively larger when compared to the IQR.

As with normalized RTT variation (Figure 16.7), much of the distribution of the ratio between maximum OTT variation and IQR fits well to a Pareto distribution, for both data packets and acks. Here, the fit is to the entire upper 50% of the distribution, and the α's are well below 2, reflecting sometimes enormous variation.

16.2.3 Path symmetry of OTT variation

We now turn to the relationship between OTT variation on the forward path and that on the reverse path. For $\mathcal{N}_2$, we find that the coefficient of correlation, η, between the max-min OTT variations of the data packets and the corresponding acks is about 0.1—quite weak, though not negligible. For IQR, it drops to 0.06, and for the max-min variation divided by IQR, it drops still further, to 0.02.

However, these statistics do not tell the whole story. As noted above, the forward path is often perturbed by the queueing load of the connection's data packets. We can instead look at OTT variation for only unloaded packets (where a packet is considered unloaded if it does not satisfy Eqn 15.5). Such packets did *not* queue behind their predecessors, unless cross traffic delayed their

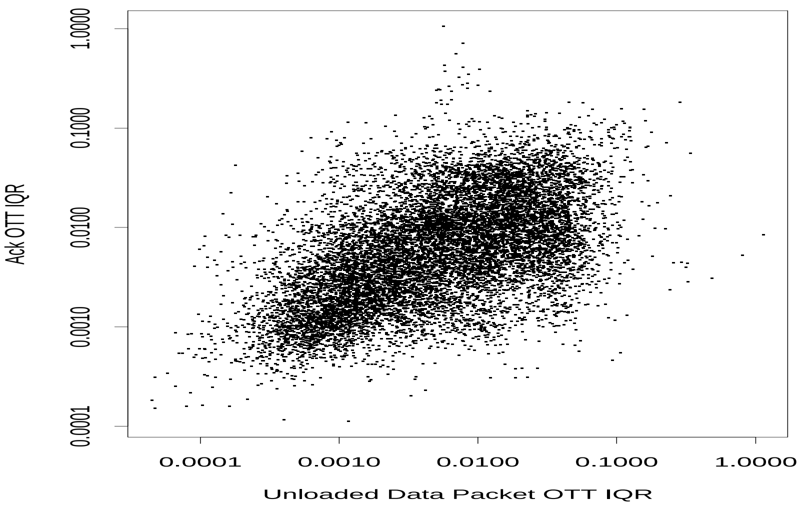

Figure 16.9: Scatter plot of interquartile ranges of unloaded data packet OTT variations (x-axis) versus acks (y-axis)

predecessors. If we analyze only unloaded packets on the forward path, then η between the IQRs of the forward and reverse variations rises to 0.18, considerably more substantial. η for the logarithms of the IQRs is 0.55, indicating that the order of magnitude of the variation along one path is a good predictor of the order of magnitude of the variation encountered along the other.[5] Figure 16.9 shows a scatter plot of the forward path IQR variation for unloaded data packets, versus the ack IQR variation. Note that both axes are log-scaled.

The correlations appear to indicate that delay variations along both directions of an Internet path are indeed coupled, albeit weakly. However, we must investigate a bit further. It could be the case that only *some* Internet paths have coupled variations, while most do not. In particular, we found in § 15.1 that European sites have higher loss rates than those in the United States, and that the paths from Europe to the U.S., and, particularly, from the U.S. to Europe, have the highest loss rates. So it could easily be that traffic between the U.S. and Europe, which traverses in each direction the highly congested trans-Atlantic links, experiences similar delay variations in both directions; while other traffic does not.

To test this effect, we repeated the above analysis with only those $\mathcal{N}_2$ connections between two sites in the U.S. We found that the correlations were only slightly weaker, indicating that the effect has only a mild influence.

In summary: if we know the OTT variation along one direction of a path, then we can fairly well predict the order of magnitude of the variation along the other direction. Predicting the variation to a finer degree is difficult. However, if we are interested not in the intrinsic delays along the path, but the delays actually experienced by a TCP connection, which include variations induced by the connection's load (i.e., its packets queueing behind their predecessors), then prediction is very

[5] If we normalize the IQRs by the round-trip times, the coefficients of correlation do not change much (rising to 0.22 and falling to 0.50, respectively).

difficult: the two directions are nearly uncorrelated.

16.2.4 Relationship between loss rate and OTT variation

It is natural to expect that delay variation might be closely correlated with packet loss, because, whenever packets are delayed in the network, they must be stored somewhere, and that storage will have a finite capacity. Thus, if delay climbs high enough, loss ensues as buffers become exhausted. However, this relationship can be obscured if routers have enough buffers to absorb considerable delay variations. It can also be obscured because delay variation derives from the *end-to-end* concatenation of variations at each hop along a path, while loss is presumed to be governed by one or perhaps a few overloaded elements along the path. Hence, many elements will contribute to delay variation but not to loss.

To investigate the relationship between delay variation and loss, we look at how the IQR of ack OTT variation correlates with the loss rate experienced by the acks. (We confine our analysis to acks to avoid the complications introduced by higher data packet loss rates due to the load they present to the forward path, per § 15.2.)

Overall, we find $\eta = 0.22$, indicating a definite, but not overly strong, linkage. However, much of the linkage comes from low OTT variation being coupled with experiencing *no loss*, a situation we referred to in § 15.1 as "quiescence." If we confine our analysis to those connections experiencing at least one loss ("busy"), then η drops to 0.12. Figure 16.10 shows the corresponding scatter plot. The plot shows some apparent structure: the region corresponding to a very low loss rate (on the y-axis) appears separate from the rest of the plot. However, this difference is a granularity artifact. The log scale highlights the difference between losing a single ack and losing two ack, since the latter corresponds to twice the ack loss rate of the former. Setting aside this artifact, we conclude that there is no strong relationship between OTT variation and loss rate.

If we log-transform both the IQR and the loss rate, then η climbs to 0.35, indicating that the order of magnitude of the IQR is a fairly good predictor of the order of magnitude of the loss rate, but nothing finer. These statistics are virtually unchanged if we confine our analysis to connections between U.S. sites, so the effect is not being skewed by the trans-Atlantic or European sites, which differ in their loss patterns (§ 15.1).

Finally, if we normalize the delay variation IQR by the connection's round-trip time, then correlation *decreases*, and, for "busy" connections, the two become uncorrelated, with $\eta = -0.02$.

We conclude that the linkage between delay variation and loss is weak, though not negligible. Unfortunately, from our data it is difficult to discern which of the two effects mentioned at the beginning of this section weakens the linkage: routers having large amounts of buffer space, or the end-to-end chain accruing a number of small variations into a single, considerably larger variation.

16.2.5 Evolution of OTT variation

We now look briefly at how OTT variation evolves with time. To do so, we follow the methodology used in § 15.5 to assess how loss rates evolve with time. For each connection c between the same source and destination, we compute the pair $\langle \Delta T_c, |\Delta\sigma_c| \rangle$, where ΔT_c is the time between that connection and the next successful connection, c', we observed along that path; and $|\Delta\sigma_c|$ is the absolute value of the difference between the IQRs of the ack OTT variations for c and c', where each IQR is normalized by the connection's round-trip time.

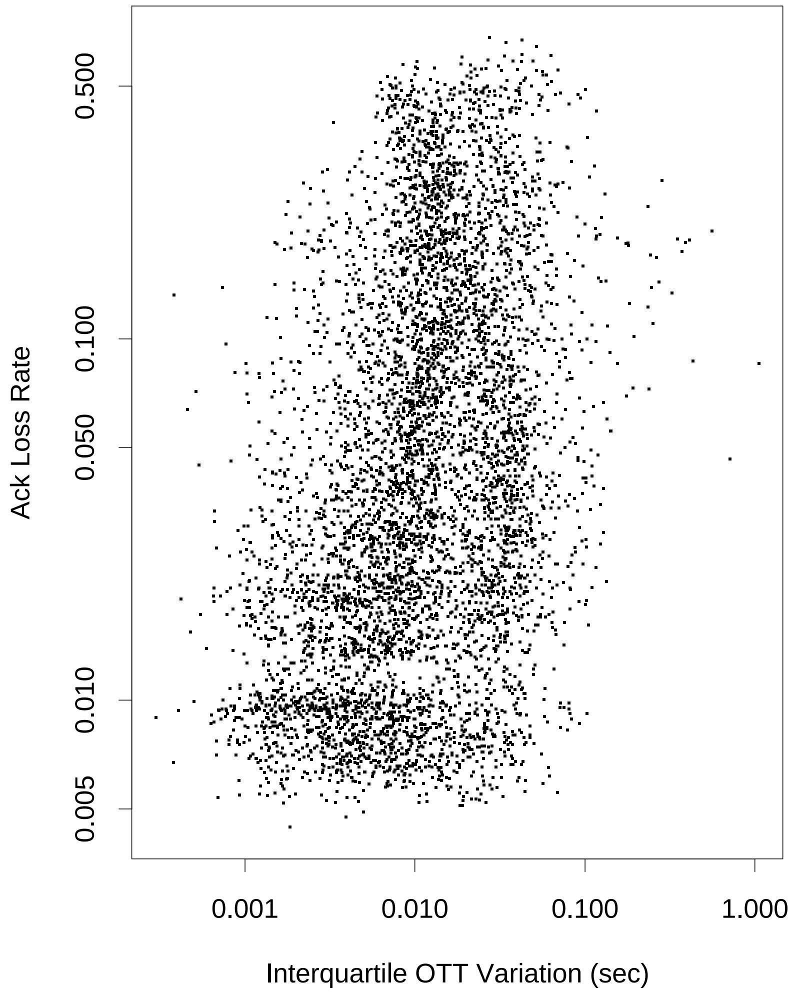

Figure 16.10: Scatter plot of ack loss rate versus interquartile ack OTT variation, for $\mathcal{N}_2$ connections that lost at least one ack

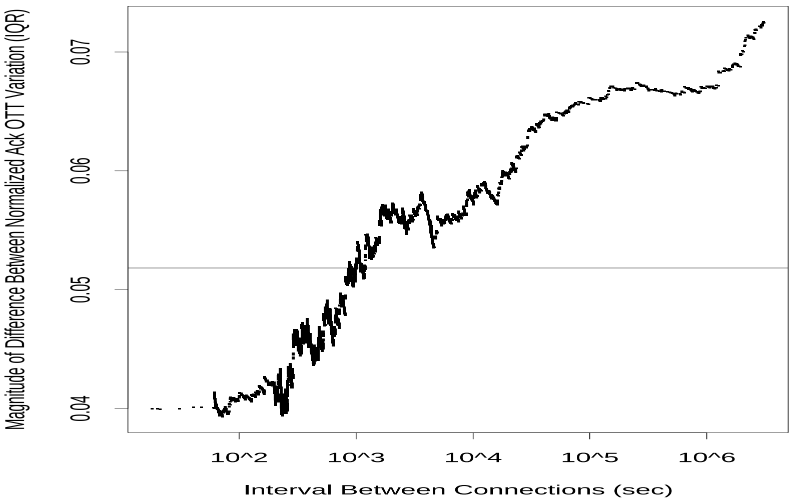

Figure 16.11: Evolution of how the interquartile range of normalized ack OTT variation differs with time

After constructing these pairs, we sort them on ΔT_c and then use an exponentially-weighted moving average (EWMA) with $\alpha = 0.01$ to smooth how $|\Delta\sigma_c|$ evolves as a function of ΔT_c. We first computed the EWMA with an initial value of 0, but inspecting the resulting plot indicated that, even for very small ΔT_c's, $|\Delta\sigma_c|$ was around 0.04, so we used 0.04 for the initial value in our final computation.

Figure 16.11 shows how the smoothed $|\Delta\sigma_c|$ evolves with time. The horizontal line corresponds to the median normalized ack OTT interquartile range for a single connection: a bit over 5% of the RTT. Note that the y-axis ranges from only 0.04 to 0.07. Thus, the change in normalized variation slowly ranges from a bit below the median variation to a bit above, across a wide range of time scales. Figure 16.12 shows the same plot except for "raw" ack OTT variations, that is, the IQR of the variations without normalizing by dividing by the connection's round-trip time. Again, we see a rapid rise followed by a slowly-increasing regime between 6-9 msec (keep in mind that this plot is heavily averaged; some paths have IQR variations far higher than 10 msec). The horizontal line corresponds to the median IQR variation for a single connection—just under 6 msec—which is quickly exceeded.

Since even the minimum $|\Delta\sigma_c|$ is not a great deal below the median normalized OTT variation, and the raw IQR differences rapidly exceed the median raw OTT variation, we conclude that a connection's ack OTT variation is *not* a very good predictor of future variation. This compares with Figure 15.18, which shows that a connection's loss rate is not a very good predictor of its future loss rate, either. Both figures argue that caching detailed network path information will prove beneficial only in the near-term, meaning on the order of a few minutes into the future.

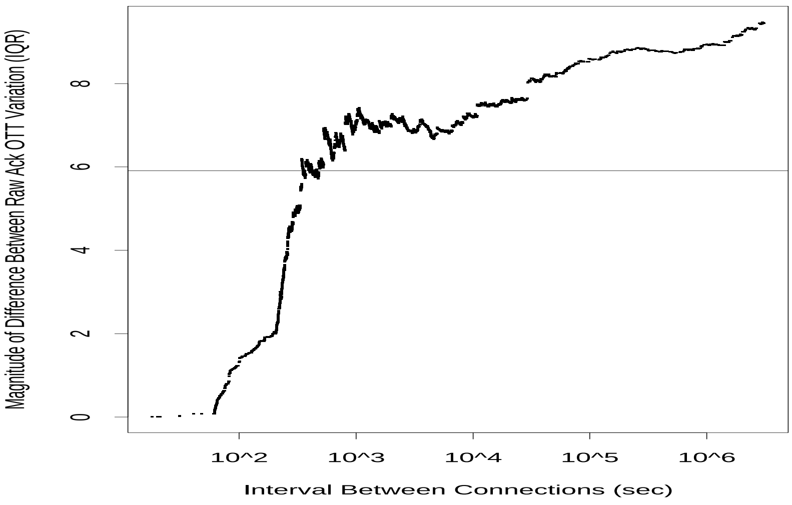

Figure 16.12: Evolution of how the interquartile range of raw ack OTT variation differs with time

16.2.6 Removing load from OTTs

In § 15.2 we developed the notion of "loaded" data packets, namely those which would have to queue behind their predecessors at the bottleneck due to the spacing between the time of their transmission and that of their predecessors. In this section we look at the subtle problem of removing the packet's load, as given by Eqns 15.3 and 15.4. The main problem we face in doing so is that the estimated bottleneck bandwidth given by Eqn 14.12 in Chapter 14 is *inexact*. In particular, our methodology produces an error *range* associated with the estimate.

Depending on which value within this range we use, Eqns 15.3 and 15.4 (or, more accurately, their counterparts for the particular estimate we use) can in some circumstances produce considerably different values for a packet's load. Thus, if we subtract that load from the packet's OTT we can easily under- or over-estimate the packet's "true" OTT, meaning its OTT if it did not have to queue behind its predecessors at the bottleneck.

We can partially address this uncertainty using a self-consistency check for the estimated bottleneck bandwidth. In particular, we can test the soundness of the central estimate of the bottleneck bandwidth, ρ_B, as follows. We first compute for each connection ΔQ, the difference between the minimum and maximum OTTs for the connection's loaded data packets. (The difference is presumably due to queueing, hence the notation ΔQ.) We then subtract out each packet's load (as given by Eqns 15.3 and 15.4 when using ρ_B rather than ρ_B^-, per Eqn 14.12), and compute $\Delta\widetilde{Q}$, the residual difference between the minimum and maximum OTTs. $\Delta\widetilde{Q}$ is thus the counterpart to ΔQ for the loaded packet OTTs, after adjusting for the connection's own contribution to the delays. We would expect to find

$$\Delta\widetilde{Q} \leq \Delta Q,$$

since a connection's extra, self-induced delay should only increase the OTT extremes it experienced.

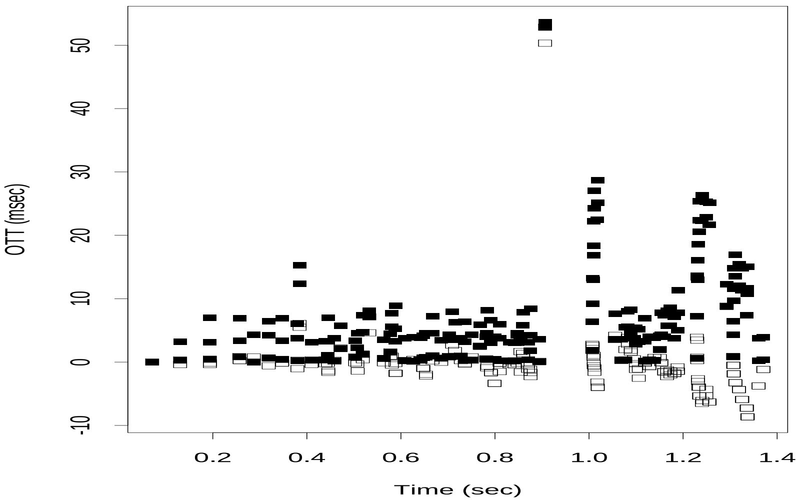

Figure 16.13: OTT plot revealing "broken" bottleneck estimate: one that is too low. Solid squares mark unadjusted OTTs, hollow squares mark OTTs adjusted to remove load based on bottleneck estimate.

If, however, we find that

$$\Delta\widetilde{Q} > 1.1 \cdot \Delta Q, \tag{16.1}$$

and if the difference between the two is also larger than twice the joint clock resolution $R_{s,r}$ (§ 12.3) (to assure that it is not just due to measurement noise), then we consider the bandwidth estimate ρ_B as *broken*: likely wrong, since using it to subtract out queueing effects actually increases the range of OTTs we observe.

This check is not foolproof. It can generate both false positives and false negatives. For example, it may be that the packet with the greatest OTT had little load to subtract out, while that with the least OTT happened to have more load, leading to an erroneous determination that ρ_B is broken. Using a factor of 1.1 in Eqn 16.1 helps avoid the possibility of these sorts of false positives, by only flagging a ρ_B estimate as broken if using it leads to a significant increase in adjusted delay.

The check might also fail, generating a false negative, if ρ_B is indeed quite inaccurate, but subtracting out inaccurate loads from the OTTs still happens to reduce their range. We find that these false negatives are much more likely to occur if ρ_B is too high, since an overestimate leads to relatively little (but still some) load being subtracted. If ρ_B is an underestimate, then excessive load is removed, which tends to lead to some packets having grossly under-adjusted OTTs, widening $\Delta\widetilde{Q}$.

The test is worth making because it detects two situations of interest. First, as noted above, if ρ_B is too low, then the calculated packet loads will be too high, and subtracting them out will often expand the range. Figure 16.13 shows an instance of this occurring. The solid squares show the OTTs of the connection's data packets, and the hollow squares correspond to the OTTs adjusted for the (erroneously too small) bottleneck bandwidth. The trend towards progressively lower adjusted OTTs indicates that the low estimate leads to removing more and more spurious load

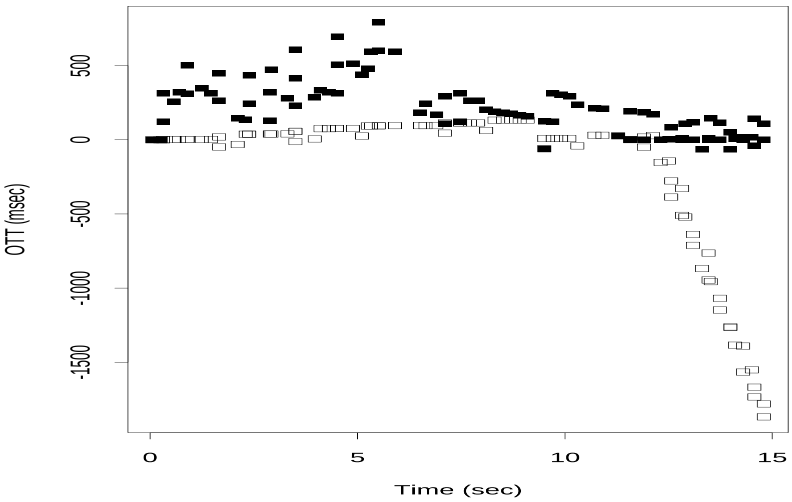

Figure 16.14: Another OTT plot revealing a "broken" bottleneck estimate: one that failed to detect a change in the bottleneck rate. Solid squares mark unadjusted OTTs, hollow squares mark OTTs adjusted to remove load based on bottleneck estimate.

as the connection transmits more packets that are erroneously judged to queue behind one another.

We particularly want to detect the case of ρ_B too low, because later in this chapter we will use load computations as a basis for determining the degree of available bandwidth in the network (§ 16.5), and we want these computations based on solid estimates of packet loads. The other case that the test can detect is the presence of an undiagnosed bottleneck change. If ρ_B corresponds to the slower of the two bottleneck rates, then `tcpanaly` will compute excessive loads for the packets transmitted during the era of the faster bottleneck rate. Figure 16.14 illustrates this happening. The estimated ρ_B is fairly accurate for most of the trace (a bit too high, as indicated by the slowly rising adjusted OTTs—not enough load is being removed). However, at $T = 12$ sec, when the bottleneck rate doubles, the estimate becomes much too low, and leads to removing too much load.[6]

Table XVIII in § 14.7 summarizes how often this check detected a broken bottleneck rate estimate in $\mathcal{N}_1$ and $\mathcal{N}_2$. It was not very often, which contributes to our faith in the PBM algorithm for detecting bottleneck rates (§ 14.6), but it did detect some problems, indicating it is worth the effort to perform the test.

As the lefthand portion of Figure 16.14 indicates, a slight mismatch in ρ_B can lead to definite, spurious trends in the adjusted OTTs. Such trends are apparent even when the estimated ρ_B is quite good. Figure 16.15 shows an OTT plot in which the bottleneck estimate is clearly quite good, as it accounts for virtually all of the variation in the OTTs (the adjusted times, shown with hollow squares, are nearly constant). Yet, if we zoom in on just the adjusted OTTs, shown in Figure 16.16, we see a clear downward trend in the adjusted OTTs. The trend corresponds to 500 μsec over about 300 msec, or about 1 part in 600. Consequently, we see that, even though our

[6]PBM does not detect this bottleneck change because it comes so close to the end of the trace.

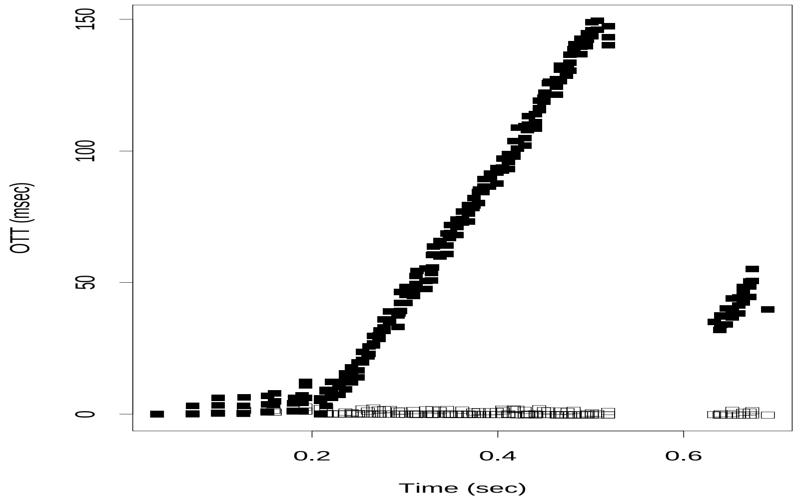

Figure 16.15: OTT plot showing virtually all OTT variation due to connection's own queueing load

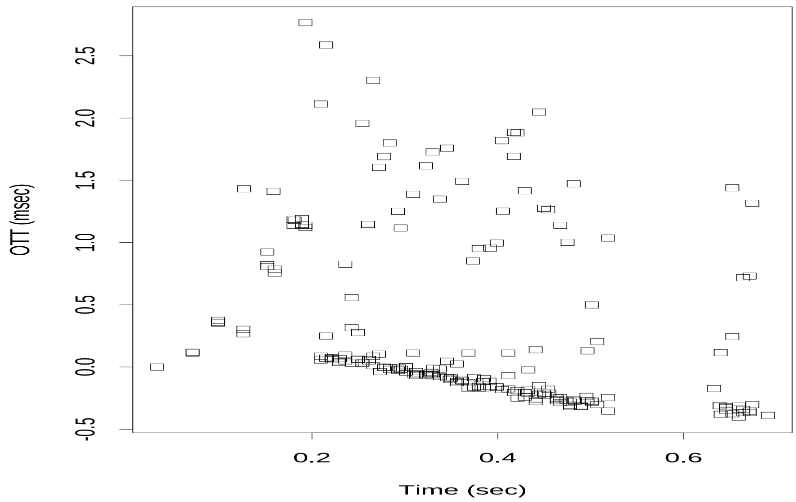

Figure 16.16: Enlargement of adjusted OTTs from previous figure

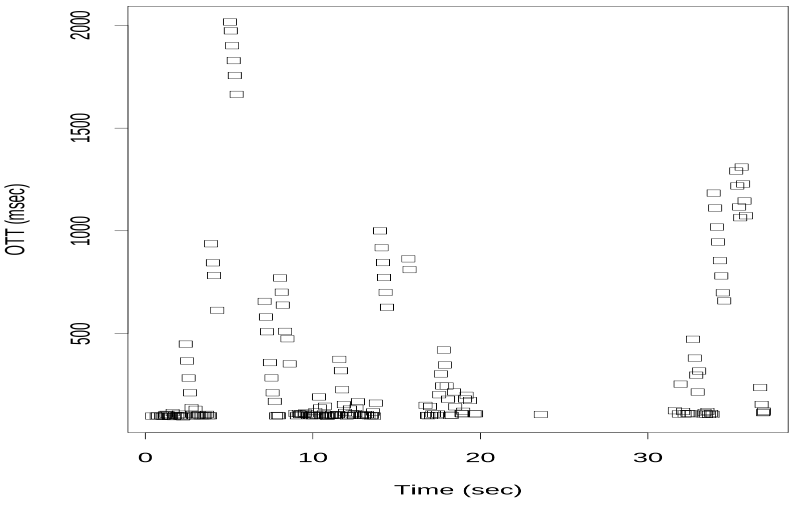

Figure 16.17: Ack OTT plot showing 10-sec periodicities

estimated ρ_B is quite good, it is not sufficiently exact to avoid introducing an artificial trend in the adjusted OTTs.

Because of this problem, we abandoned our original goal of trying to treat loaded packets the same as unloaded packets by adjusting their OTTs, as doing so requires extremely precise estimation of ρ_B. If the estimate is off, we introduce systematic errors that could easily be confused with genuine network effects.

16.2.7 Periodicity in OTTs

In § 15.3 we discussed our efforts at testing whether packet loss patterns exhibit periodicity. We might expect them to do so due to synchronization effects known to sometimes plague Internet routers, resulting in periodic packet forwarding outages. These lead to lengthy delays and perhaps loss, if buffers become exhausted during the outage [FJ94]. In this section we briefly discuss evidence in our data for periodic variations in packet delays.

In attempting to assess delay periodicities, we run into the same problems as when assessing loss periodicities: our data unfortunately are not suited for a proper investigation of the question. § 15.3 outlines the reasons for this and we will not repeat them here. We did, however, attempt the same analysis as in § 15.3: we selected connections between the North American sites exhibiting the highest degree of clock synchronization, singled out the busiest day among them, and analyzed their connections to determine the time at which the connection's largest delay occurred. We then studied plots of the peak delay time versus the same time modulo different possible periodicity intervals. This effort did not find any conclusive evidence of global periodicities.

However, phenomenological inspection of other traces shows that delay periodicities definitely do occur. Figure 16.17 shows a plot of ack OTT times for a connection from `connix` to `ucl`. The distance between the first OTT peak at about 1000 msec and the second such peak (we are ignoring the striking, 2000 msec peak) is 10.07 sec, while that between the second peak and the third such peak (at about 1200 msec) is 19.92 sec. Furthermore, two acks *were* sent about 10 sec after the second peak, but both were lost (hence, they do not appear on the plot). Thus, this trace exhibits strong evidence of a 10-second periodicity. We find a number of other traces to `ucl` with the same spacing between delay peaks, suggesting that this is an ongoing phenomenon.

We observed other traces with apparent 5-second and 30-second periodicities in delay spikes, involving different hosts, indicating that the phenomenon is not confined to only `ucl`. On the other hand, we did not find strong evidence above of global periodic delay variation among the highly-synchronized North American sites. Thus, we conclude that the phenomenon is definitely present, but, if widespread, at least not globally synchronized.

16.3 Timing compression

Packet timing *compression* occurs when a flight of packets are sent over an interval ΔT_s but arrive at the receiver over an interval ΔT_r, with $\Delta T_r < \Delta T_s$. 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, as later ones have to wait behind the transmission of earlier ones (§ 14.2). 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.

Zhang et al. predicted from theory and simulation that acks could be compressed ("ack compression") if a flight arrived at a busy router (one with a significant queue), and if no intervening packets arrived between the different acks [ZSC91]. As the acks queue behind one another, the potentially large spacing between them due to self-clocking (§ 9.2.5) and ack-every-other policies (§ 11.6) would then be lost when the acks were later transmitted back-to-back upon reaching the front of the queue.

This situation corresponds to a *draining* queue: a router that was busy when the first ack arrived (and hence could not service it before the others arrived), and yet new arrivals from other traffic sources are sporadic. If instead new arrivals were steady, then they would occupy slots in the queue between the acks in the flight, and their spacing would be (roughly) preserved, rather than compressed.

Mogul subsequently analyzed a trace of Internet traffic and confirmed (among other phenomena) the presence of ack compression [Mo92]. His definition of ack compression is somewhat complex, involving significant deviations from the median inter-ack spacing, since he had to infer endpoint behavior from an observation point inside the network (a vantage point problem, per § 10.4). But he clearly detected the presence of ack compression. He found that compression was correlated with packet loss but considerably more rare. His study was limited, however, to a single 5-hour traffic trace.

Since we can readily compute from our data both ΔT_s and ΔT_r for any flight of packets, we can use a simpler definition of compression than employed by Mogul. In this section we characterize three different types of compression: ack compression (§ 16.3.1), data packet compression (§ 16.3.2), and receiver compression (§ 16.3.3). We show that all three types of compression occur within the Internet, though each is limited in its effects.

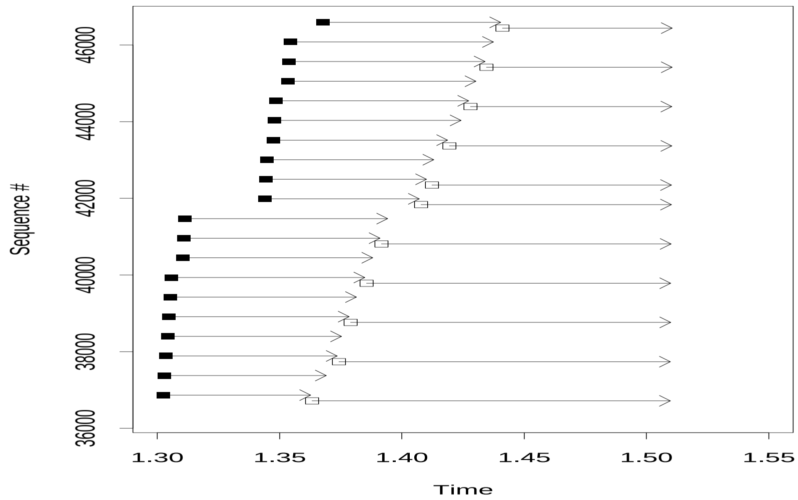

Figure 16.18: Paired sequence plot showing ack compression

16.3.1 Ack compression

If ack compression is frequent, it presents 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 (§ 9.2.5). 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 (§ 14.3) can misinterpret compressed acks as reflecting greater bandwidth than truly available. On the other hand, some researchers argue that occasional ack compression is beneficial since it provides an opportunity for self-clocking to discover newly-available bandwidth.

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

$$\xi = \frac{\Delta T_r + C_r}{\Delta T_s - C_s}, \tag{16.2}$$

where C_r and C_s are the receiver and sender's clock resolutions. Using Eqn 16.2 results in ξ being a conservative estimate, since by adding C_r in the numerator but subtracting C_s in the denominator, we tend to inflate ξ.

We consider a group of acks 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 per connection was around 2, and 1% of the connections experienced 15 or more. Almost all compression events are small, however, with only 5% spanning more than five acks. Figure 16.18 shows a paired sequence plot of one of the larger events, in which eleven acks were compressed. The solid squares indicate when the data packets were sent, and the arrows stemming from them point to their arrival times at the receiver. The corresponding acks (offset downward a bit, for legibility) are shown with hollow squares. The arrows from these squares all stop at virtually the same point in time, $T = 1.51$, indicating that, even though the acks were sent over an interval of 77 msec, they arrived all together, about 760 μsec apart—compressed by a factor of 100.

We also note that 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. Compressed dup acks are only slightly more likely to occur in a large burst than compressed regular acks. In $\mathcal{N}_2$, overall 5.1% of the compression events consisted of six or more acks: 4.8% of the regular-ack compression events, and 6.0

Finally, we classify compression events as "major" if the compression results in the acks arriving at the data sender with a spacing less than that corresponding to the bottleneck bandwidth; otherwise, we term the event "minor." Major events are significant because they reflect a breakdown in self-clocking—namely, the sender will transmit in response to them at a rate exceeding the bottleneck bandwidth—and they also make sender-based bottleneck estimation difficult, since, unless detected, they will lead to overestimates.

Let ρ_B^+ be the upper bound on the estimated bottleneck bandwidth, per Eqn 14.12. If a flight of k packets arrives during an interval ΔT_r, and they together acknowledge a total of b bytes, then we consider the flight to reflect a major compression event if:

$$\frac{b}{\Delta T_r} > \rho_B^+.$$

We apply this test to each ack compression event detected by `tcpanaly`, except we omit the final ack of the event. The reason for this omission is that `tcpanaly` finds compression events by constructing groups of acks for which $\xi < 0.75$, and sometimes the final ack of the group is relatively uncompressed compared to the others (i.e., it raised ξ from a small value to a value near 0.75). Consequently, we omit this final ack to avoid skewing the assessment of "major" events by our methodology for grouping acks into events.

We find that in both $\mathcal{N}_1$ and $\mathcal{N}_2$, about 75% of the compression events are major. This figure only slightly diminishes if we confine our analysis to compressed "regular" acks, eliminating compressed dup acks.

Of the major compression events, 80% reflect acks arriving at a rate corresponding to more than twice ρ_B^+. Thus, when compression occurs, it is usually large enough to result in a significant overestimate of the bottleneck bandwidth.

From these findings, we conclude that ack compression definitely occurs in the Internet, but rarely enough as to not pose a significant problem by corrupting self-clocking or causing excessive burstiness. That it occurs for more than half the connections, however, and that most of these are "major," indicates that a sender-based measurement scheme *must* employ filtering to remove extreme values from its bottleneck estimates, as otherwise it is very likely to overestimate the bottleneck bandwidth, with perhaps disastrous consequences.

16.3.2 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 (§ 14.2). If after the bottleneck their timing is compressed, then use of Eqn 16.2 will *not*

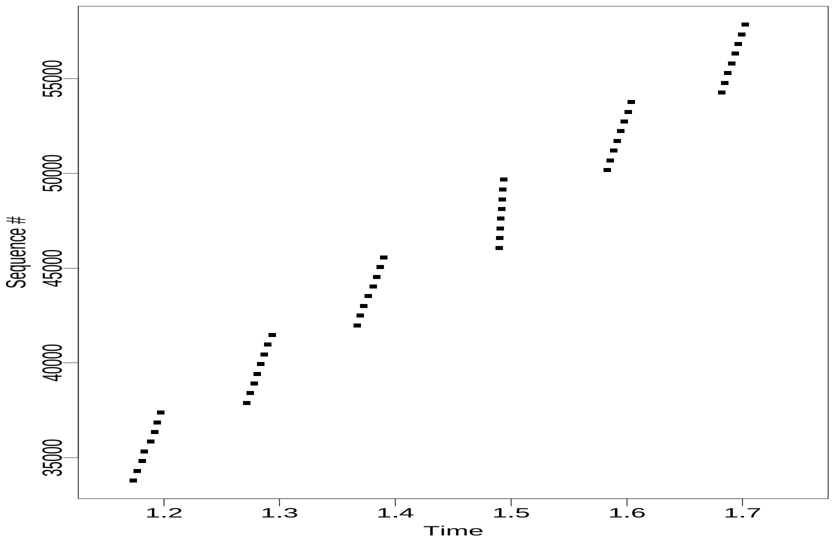

Figure 16.19: Data packet timing compression

detect this fact unless they are compressed to a greater degree than their sending rate. Figure 16.19 illustrates this concern: the flights of data packets arrived at the receiver at 170 Kbyte/sec (T1 rate), except for the central flight, which arrived at Ethernet speed. However, it was also sent at Ethernet speed, so, for it, $\xi \approx 1$.

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

These compression events are more rare than ack compression, occurring in only 3% of the $\mathcal{N}_1$ traces and 7% of those in $\mathcal{N}_2$. We were interested in whether some paths might be plagued by repeated compression events due to either peculiar router architectures or network dynamics. Only 25–30% of the traces with an event had more than one, and 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 some paths are indeed prone to timing compression. Figure 16.20 shows an example. Here, the bottleneck rate is T1, which corresponds closely with the flatter slopes in the plot.

Thus, it appears that data packet timing compression is rare enough not to present a significant problem. That it does occur, though, again highlights the necessity for outlier-filtering when conducting timing measurements.[7]

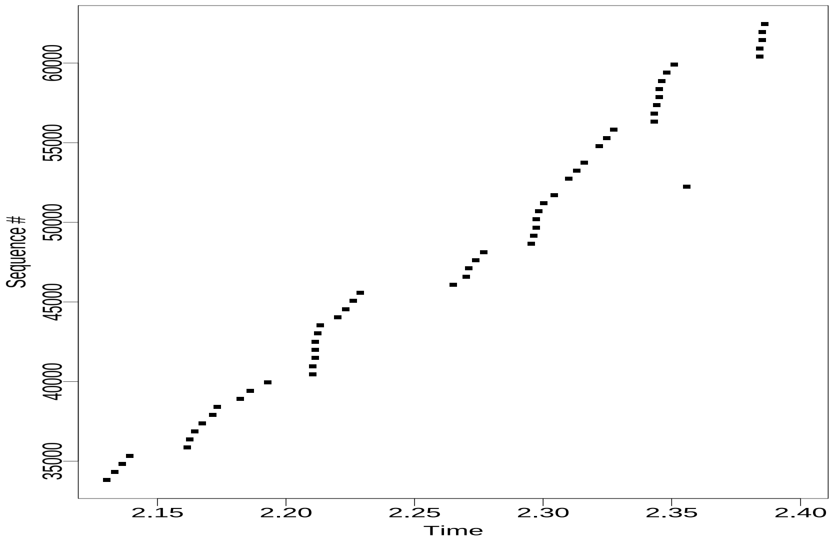

Figure 16.20: Rampant data packet timing compression

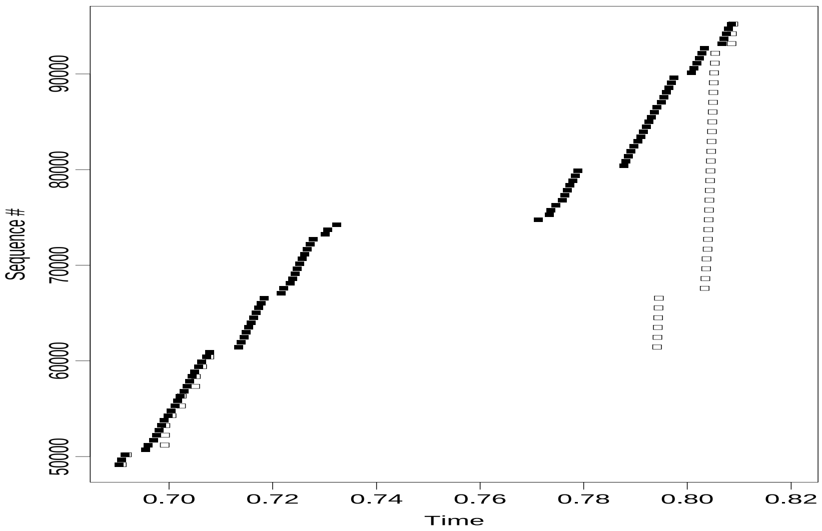

Figure 16.21: Receiver sequence plot showing major receiver compression

16.3.3 Receiver compression

A third type of timing compression occurs when the receiver delays in generating acks in response to incoming data packets, and then generates a whole series of acks at one time. The timing of these acks appears compressed to the sender, though *not* for reasons of network dynamics, but instead due to lulls at its remote peer. Figure 16.21 shows the most striking example in our traces, in which the `lbl` receiver compressed 25 of its acks, sending them over a 2 msec interval instead of over the 83 msec interval corresponding to the data packets they acknowledged. (Slightly earlier, the receiver also compressed 6 other acks, as seen in the figure.)

Since receiver compression is an endpoint effect, its presence tells us nothing about the dynamics of the connection's Internet path. However, receiver compression remains quite interesting because it is an additional noise element that any sender-only measurement scheme must contend with. It also leads to the same consequences as true ack compression, namely a break-down of a connection's self-clocking.

To assess receiver compression, we compute:

$$\xi' = \frac{\Delta T_a + C_r}{\Delta T_d - C_r},$$

where ΔT_a is the spacing between the generated acks, ΔT_d is the spacing between the arriving data packets (the ones that led to the acks), and C_r again is the receiver's clock resolution. As in Eqn 16.2, the addition of C_r in the numerator and subtraction in the denominator makes ξ' conservative. We consider $\xi' < 0.75$ as indicating a receiver compression event. Note that our earlier analysis of ack compression uses ΔT_a as the original spacing of a flight of acks, and then checks whether that was compressed while the packets were in flight. Consequently, that analysis does *not* confuse ack compression with receiver compression: the earlier ack compression analysis only evaluates compression due to network behavior.

We include delayed acks in our analysis, as these affect self-clocking. Sender-based measurement techniques can generally detect delayed acks.[8] In both $\mathcal{N}_1$ and $\mathcal{N}_2$, we find that about 10% of the connections included a receiver compression event of at least three acks. Of these, about three-quarters experienced only one receiver compression event, and, in $\mathcal{N}_1$, none experienced more than four, though, in $\mathcal{N}_2$, the upper limit was 15. Almost all events were only 3 acks in size (95% in $\mathcal{N}_1$, 80% in $\mathcal{N}_2$).

While these statistics indicate that receiver compression is fairly rare, and even less often significant, we must note that, because receiver compression is an *endpoint* effect, these statistics are *not* necessarily representative of its frequency in the Internet as a whole. In particular, we find that just a few sites cause the majority of the receiver compression events in our study, so we have no way of telling whether other sites would tend overall towards more receiver compression or less.

Given this caveat, we note that we find receiver compression, like other forms of timing compression, to be fairly rare. In particular, in our datasets it appears more rare than ack compression, so, if this is a representative finding, then sender-based assessment of ack compression caused by network dynamics will not be terribly skewed by the presence of receiver compression.

---

[7] It also has a measurement benefit: from the arrival rate of the compressed packets, we can estimate the downstream bottleneck rate.

[8] Using the rule that an ack for less than two full segments was presumably delayed. This heuristic could fail in the future, if TCPs begin to ack every packet, which they might do to accelerate the slow-start process.

If the sender-based measurement employs filtering to remove outliers, as it needs to do anyway to deal with ack compression, then receiver compression does not make the measurement significantly harder.

16.4 Queueing analysis

In this section we develop a rough estimate of the time scales over which queueing occurs. If we take care to eliminate suspect clocks (Chapter 12), reordered packets (§ 13.1), compressed packets (§ 16.3), and traces exhibiting TTL shifts (which indicate routing changes, per § 7.7), then we argue that the remaining measured OTT variation is mostly due to queueing. Hence, we can estimate queueing time scales by analyzing time scales of OTT variations.

For a given time scale, τ, we compute the queueing variation on that time scale as follows. First, we partition the packets sent by a TCP into intervals of length τ. For each interval, let n_l and n_r be the number of successfully-arriving packets in the left and right halves of the interval. If either is zero, or if $n_l < \frac{1}{4}n_r$, or vice versa, then we reject the interval as containing too few measurements or too much imbalance between the halves.

Otherwise, let m_l and m_r be the median OTTs of the two halves. We then define the interval's queueing variation as $|m_l - m_r|$. Thus, we quantify the variation as how much the OTTs changed over a time scale of τ, but, by computing the change only as the difference between two intervals of length $\frac{1}{2}\tau$, we include in the variation *only* changes that occurred on the time scale of τ. Changes that occurred on smaller time scales will in general all occur within either the left or right half, and the *median* of the half will not reflect the smaller-time-scale change. Changes occurring on larger time scales will not in general result in variation between the two halves, and so likewise will not enter into the computation.

By using medians, we attempt to reduce the effects of occasionally very large OTTs. We found that means and standard deviations can often be unduly skewed by a small set of large OTTs.

The question remains how to summarize the interval changes. We investigate two different summaries. In the first, we define ΔQ_τ as 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 again is robust in the presence of noise due to non-queueing effects, or transient queueing spikes. In addition, we compute $Q_\tau^{\max}$, the maximum observed difference across any two halves of an interval of length τ. ΔQ_τ thus summarizes *sustained* variation on the time scale τ, while $Q_\tau^{\max}$ summarizes *bursts* of variation on the time scale τ.

We now analyze ΔQ_τ and $Q_\tau^{\max}$ for different values of τ, confining ourselves to variations in ack OTTs, as these are not clouded by self-interference and adaptive transmission rate effects (§ 15.2). The question we wish to address is: are there particular τ's on which most queueing variation occurs? This question is particularly interesting because of its potential implications for engineering transport protocols. 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. Or if, for example, the dominant τ is on the order of 1 sec, then that constant helps us determine the related constants—such as the α's for EWMA estimators—governing how a transport connection should update its RTT estimate in order to compute its retransmission timeout.

For each connection, we range through $2^4, 2^5, \ldots, 2^{16}$ msec to find $\widehat{\tau}$, the value of τ for

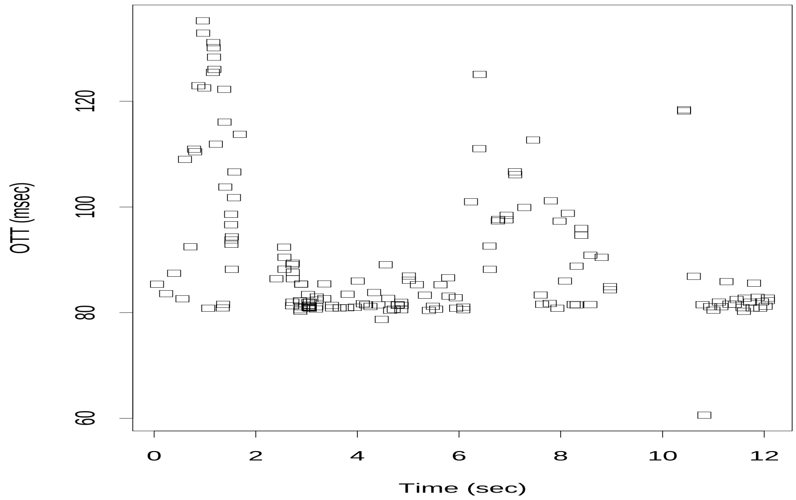

Figure 16.22: Ack OTT plot for a connection with $\widehat{\tau} = 4$ sec for ΔQ_τ

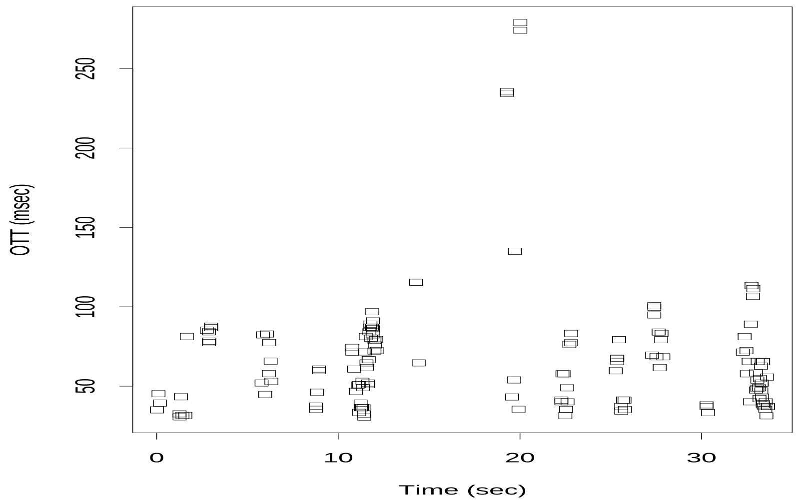

Figure 16.23: Ack OTT plot for a connection with $\widehat{\tau} = 1$ sec for $Q_\tau^{\max}$

which ΔQ_τ or $Q_\tau^{\max}$ is greatest. $\hat{\tau}$ reflects the time scale for which the connection experienced the greatest OTT variation, where the variation is *sustained* if computed for ΔQ_τ, and *momentary* if computed for $Q_\tau^{\max}$. Figure 16.22 shows a plot of the ack OTTs for a connection with $\hat{\tau} = 4$ sec for ΔQ_τ, indicating maximum sustained variation occurs on 4-second time scales. Figure 16.23 shows a connection with $\hat{\tau} = 1$ sec for $Q_\tau^{\max}$, which emphasizes the large increase in delay at $T = 20$ sec. For the first connection, the maximal $Q_\tau^{\max}$ occurs for $\hat{\tau} = 64$ msec, corresponding to the sharp spike just after $T = 1$ sec. For the second, the maximal ΔQ_τ occurs for $\hat{\tau} = 4$ sec, due to the sustained variation on 4-second time scales (for this connection, other time scales have large ΔQ_τ, too, but the largest is for $\tau = 4$ sec). Clearly, the time scales of maximum *sustained* burstiness versus those of maximum *peak* burstiness can differ considerably.

Before looking at the range in $\hat{\tau}$'s for our measurements, a natural calibration question is what sort of $\hat{\tau}$'s we find for synthetic variations. We investigated this question by simulating 10,000 independent and identically distributed (i.i.d.) OTT variations. Each variation was simulated as a random variable drawn from an exponential distribution with $\lambda = 1$,[9] corresponding to an OTT variation computed for one unit of time (the equivalent of 2^4 msec for the preceding discussion). For 100 simulation runs, $\hat{\tau}$ was always ≤ 2 units of time for ΔQ_τ, and ≤ 4 units of time for $Q_\tau^{\max}$. Thus, we see that $\hat{\tau}$ correctly indicates that the variation is confined to small time scales. If we simulate i.i.d. Pareto variations with $\alpha = 1.01$ (so, infinite variance and, just barely, finite mean), we still find $\hat{\tau}$ confined to small time scales, never exceeding 4 units of time. Again, this is what we would expect, because the fundamental time scale of change is one time unit, since the variations are independent.

Figure 16.24 shows the normalized proportion of the connections in $\mathcal{N}_1$ and $\mathcal{N}_2$ exhibiting different values of $\hat{\tau}$ for ΔQ_τ. Normalization is done by dividing the number of connections that exhibited $\hat{\tau}$ by the number that had durations at least as long as $\hat{\tau}$, so that the prevalence of short connections does not skew the distribution. For both datasets, time scales of 128–2048 msec primarily dominate. This range, though, spans more than an order of magnitude, and also exceeds typical RTT values. Furthermore, while less prevalent, $\hat{\tau}$ values all the way up to 65 sec remain common, with $\mathcal{N}_1$ having a strong peak at 65 sec.[10]

Consequently, the figure indicates that *sustained Internet delay variations occur primarily on time scales of 0.1-2 sec, but extend out quite frequently to much larger time scales.*

Figure 16.25 shows the same figure but for $Q_\tau^{\max}$. Here we see that basically the same time scales dominate variation peaks, ranging from 128 to 1024 msec. Smaller time scales clearly contribute, however, and so do larger time scales up to about 4 sec, with $\mathcal{N}_1$ exhibiting a trend towards still larger time scales, while $\mathcal{N}_2$ does not. We interpret the figure as indicating that *peak Internet delay variations also occur primarily on time scales of 0.1-1 sec, but they too extend to larger time scales, and quite often to smaller time scales*. Consequently, it appears clear that there is no single time scale of "burstiness," which accords with the recent "self-similar" models of network traffic [LTWW94], though, as a rule of thumb, most variation occurs on time scales of a quarter-second to a half-second, a bit above usual connection round-trip times. Thus, it appears that transport connections *can* feasibly adapt to queueing changes, but to do so they must act quickly, within a few RTTs, or else it will often be too late.

[9]The results are independent of λ, however, since λ only determines the size of the identically-distributed variations, but not the time scales of the variations among them.

[10]Manual inspection of traces with $\hat{\tau} = 65$ sec indicates that they do indeed exhibit their maximum variation on that time scale, addressing the concern that perhaps the peaks were due to some other effect, and hence spurious.

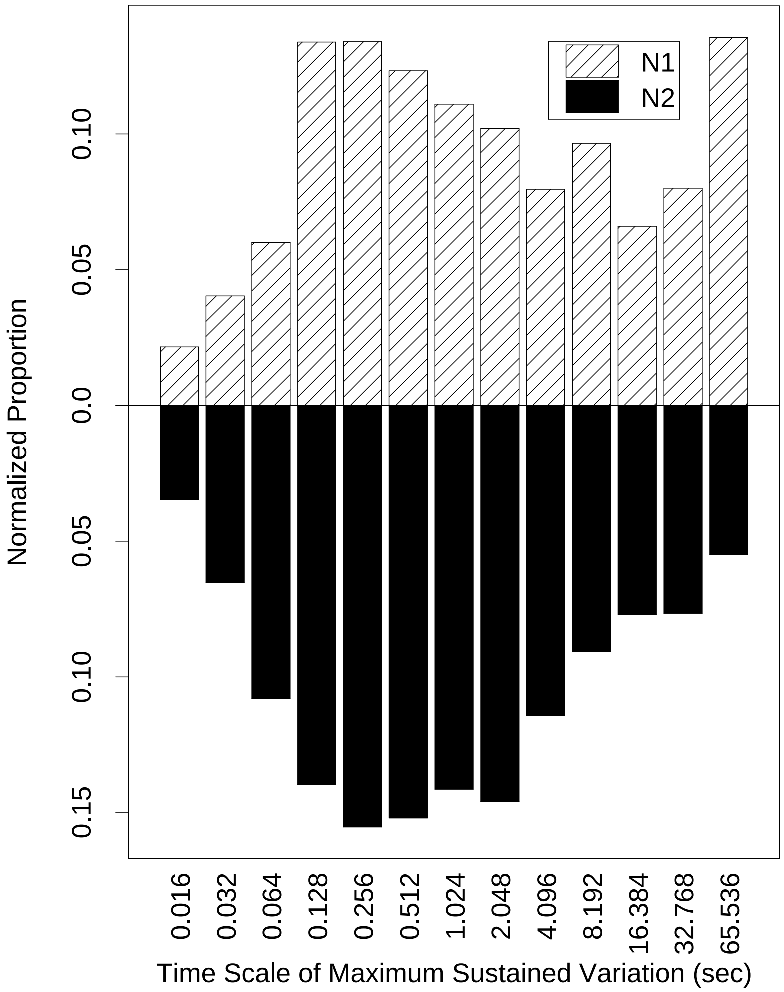

Figure 16.24: Proportion (normalized) of connections with given timescale of maximum sustained delay variation ($\hat{\tau}$)

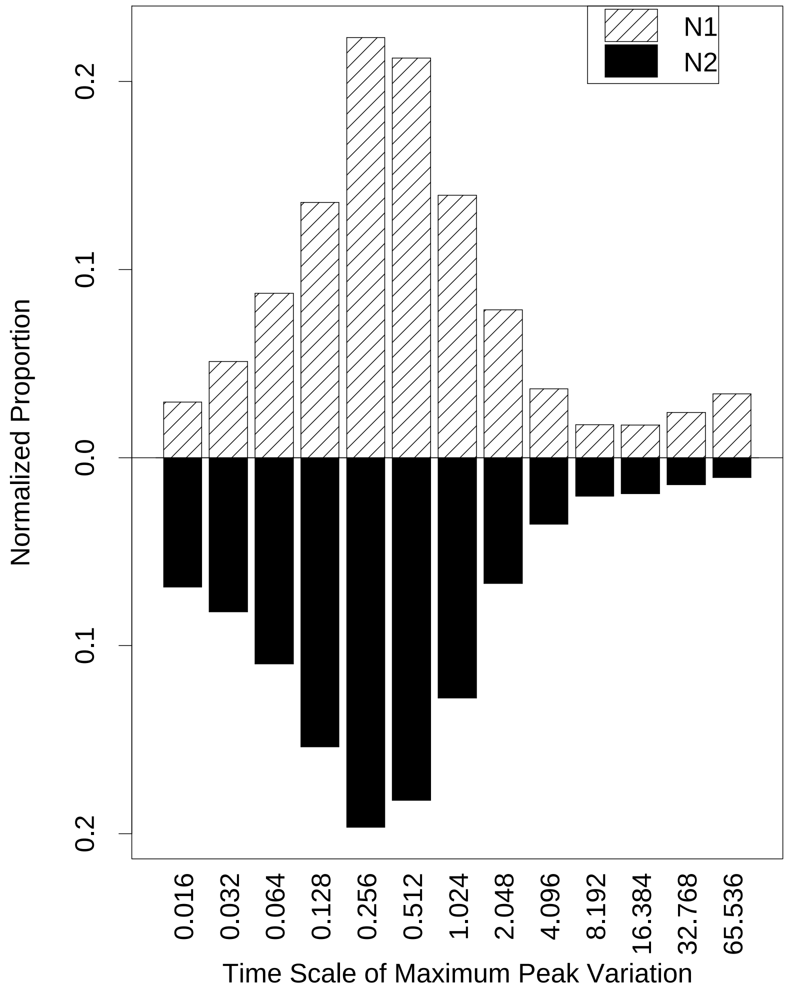

Figure 16.25: Proportion (normalized) of connections with given timescale of maximum peak delay variation ($\widehat{\tau}$)

16.5 Available bandwidth

The last aspect of delay variation we look at is an interpretation of how it reflects the *available bandwidth*. In a packet-switched network, available bandwidth is a somewhat elusive notion. The amount of bandwidth a connection might fruitfully use varies with time, as other cross-traffic connections come and go. From § 16.4 we know that significant OTT fluctuations often occur on time scales of 100-1000 msec, and for the upper end of this range (which actually extends appreciably to much larger time scales), no doubt most of the fluctuations are due to connections beginning or ending, rather than flights of packets within single connections beginning or ending.

Two existing approaches for estimating available bandwidth are `cprobe` [CC96b] and `Treno` [MM96]. `cprobe` works in conjunction with `bprobe` [CC96a], which we discussed in § 14.2. To estimate available bandwidth along a network path, `cprobe` first uses `bprobe` to estimate the bottleneck bandwidth along the path. `cprobe` then transmits four groups of probes, each probe consisting of 10 ICMP echo packets (as with `bprobe`). The echo packets are sent at a rate exceeding that of the estimated bottleneck bandwidth, to make sure they attempt to fully utilize the bottleneck. `cprobe` then computes from the timing of the ICMP echo replies the achieved throughput, and considers the ratio between this and the bottleneck bandwidth to be the *utilization* (similar to the value β which we define in Eqn 16.4 below), which indicates how much of the bottleneck bandwidth was actually available.

`cprobe` has three limitations that we attempt to address. The first is that it requires sending a fairly large flight of packets at a rate known to exceed what the network path can support, so `cprobe` can be viewed as fairly *stressful* to a network path. The second is that, because its probes use ICMP echo packets, which elicit same-sized replies, the achieved throughput the probes achieve will reflect the *minimum* of the available bandwidth along the forward and reverse paths. As we have seen that many path properties are asymmetric, it would not be surprising to find that available bandwidth is, too, and thus, for a unidirectional connection, `cprobe` might produce too pessimistic an estimate. The third limitation is that the pattern in which the probe packets are sent differs from that in which a TCP sender will transmit its data packets. We have seen in § 15.2 that, because TCP adapts its transmission rate to the presence of packet loss along the forward path, network conditions observed by TCP data packets can differ significantly from those observed by TCP ack packets. Thus, we suspect that available bandwidth estimates produces by `cprobe` might not closely reflect the throughput that a TCP would actually achieve.

This second point is addressed by the developers of the `Treno` utility [MM96]. `Treno` also uses ICMP echo packets to probe network paths, but it sends them using an algorithm equivalent to that used by TCP congestion control (§ 9.2). In addition, `Treno` can probe hop-by-hop available bandwidth by using increasing TTL (time-to-live) values in the IP headers of the echo packets it sends, just as does `traceroute` (§ 4.2.1). When doing so, it receives in response from each hop (except the last) not a full-sized echo reply, but a short ICMP Time Exceeded message. Thus, even if the available bandwidth along the return path is less than that along the forward path, `Treno` will still primarily observe the forward-path available bandwidth, just as would a TCP connection that receives only data-less acks in response to its data packets.

The main drawback of `Treno` is that it is a *stressful* technique. It estimates how fast a TCP could transfer data over a given network path by seeing how fast it itself can transfer data over the path, using a standard-conformant, but well-tuned, implementation of the TCP congestion control algorithm.

Ideally, we would like to estimate available TCP bandwidth *without* fully stressing the network path to do so. We do not achieve this goal in our present work. Instead, in this section we analyze our TCP transfer data both to characterize available bandwidths in the Internet, and to explore how we might perhaps in the future develop a non-stressful available-bandwidth estimation technique, based on fine-scale analysis of TCP packet timings. For this technique, the hope is that by carefully scrutinizing the delays of individual TCP packets, we might form a good estimate of the bandwidth available along the path they were sent, without requiring that we send the packets at a rate that saturates the path for any lengthy period of time.

We proceed as follows. First, we need to define what we mean by available bandwidth. We might argue that, if we know that a connection is competing with k other connections, then its fair share of the network resources is $\frac{1}{k+1}$. In particular, the connection's fair share of the bottleneck bandwidth, ρ_B, is $\frac{\rho_B}{k+1}$. These simple notions, however, quickly run into difficulties. First, during a connection's lifetime, competing connections come and go, so there is no single value to assign to k. Second, the competing connections do *not* in general compete along the entire end-to-end path, but only for a portion of it, so there may in fact be a great number of competing connections, but each competing for different resources. Finally, "fairness" itself is an elusive notion: it might well be the case that, for policy reasons (such as who is paying for what), or due to different traffic types, the simple each-gets-an-equal-share division of the resources is deemed inappropriate. (See [Fl91] for further discussion of the difficulty of defining a single notion of fairness.)

With these considerations in mind, we now strive to develop a notion of "equivalent competing connections," in order to talk in general terms about available resources. To do so, we attempt to characterize the network resources available to a connection as a fraction of the total resources in use. The term we will use to capture this notion is "available bandwidth." Here we presume that connections push on the network to extract as much resource from it as they can—TCP's slow start does exactly this.[11] Therefore, if a connection pushes on the network and we observe that it consumed m units of resources, and we can determine that other connections consumed n units of the same resources, then we will consider the available bandwidth as $\frac{m}{m+n}$; or, equivalently, that, over its lifetime, the connection competed with the equivalent of $\frac{n}{m}$ other connections like itself.

We will use as our unit of resource the amounts of buffer space and transmission time the connection consumed at the bottleneck link. In § 15.2 we developed a notion of data packet i's "load," λ_i, meaning how much delay it incurs due to queueing at the bottleneck behind its predecessors, plus its own bottleneck transmission time, ϕ_i, which is directly determined by the packet's size and the bottleneck bandwidth. Let

$$\psi_i = \lambda_i - \phi_i, \tag{16.3}$$

namely, just the amount of a packet's delay that is due to queueing behind its predecessors.

Let γ_i denote the difference between packet i's measured OTT and the minimum observed OTT (for full-sized packets). If the network path is completely unloaded except for the connection's load itself (no competing traffic), then we should have $\psi_i = \gamma_i$, i.e., all of i's delay variation is due to queueing behind its predecessors. More generally, define

$$\beta = \frac{\sum_i (\psi_i + \phi_i)}{\sum_j (\gamma_j + \phi_j)}. \tag{16.4}$$

[11] We do not, however, presume that the *measurement technique* for estimating how much bandwidth is available must also do so.

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

More generally, $\sum_i (\psi_i + \phi_i)$ reflects the resources consumed by the connection, while

$$\sum_j (\gamma_j + \phi_j) - \sum_i (\psi_i + \phi_i) = \sum_j \gamma_j - \sum_i \psi_i$$

reflects the resources consumed by the competing connections.

Note that including the ϕ_i terms in Eqn 16.4 is important: they reflect the basic bottleneck transmission cost. Without them, a connection that does not load the bottleneck link (perhaps because its transmission perfectly matches the bottleneck rate) will exhibit

$$\sum_i \psi_i = 0.$$

In this case, any slight variation in its OTTs, i.e.,

$$\sum_j \gamma_j = \epsilon > 0,$$

will result in $\beta = 0$. But in this limiting case we want our evaluation to indicate that almost all the resource was available (as indicated by $\sum_j \gamma_j$ being small), and this is exactly the limiting behavior of Eqn 16.4.

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

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

We can gauge how well β truly reflects available bandwidth by computing the coefficient of correlation between β and the connection's overall throughput (normalized by dividing by the bottleneck bandwidth). For $\mathcal{N}_1$, this is 0.44, while, for $\mathcal{N}_2$, it rises to 0.55. We conjecture that the difference is due to the use of bigger windows in $\mathcal{N}_2$ (§ 9.3), which lead to more opportunities for fast retransmission. Any time a connection times out, its overall throughput becomes greatly diluted by the lengthy timeout lull.

Thus, the correlations, particularly for $\mathcal{N}_2$, indicate that β is indeed a solid predictor of a connection's likely overall performance. It is not a perfect predictor, however, nor would we expect it to be: a TCP connection's overall throughput is affected by the number of retransmissions it incurs, whether any of these are timeout retransmissions, the receiver's offered window, the sender's internal window (§ 11.3.2), how the TCP manages the congestion window, and the acking policy used by its remote peer, which determines how fast the slow-start sequence increases the window (§ 11.6.1).

Figure 16.26 and Figure 16.27 show the density of β for $\mathcal{N}_1$ and $\mathcal{N}_2$. Values less than zero and greater than one, which can result from erroneous estimates of ρ_B, have been adjusted

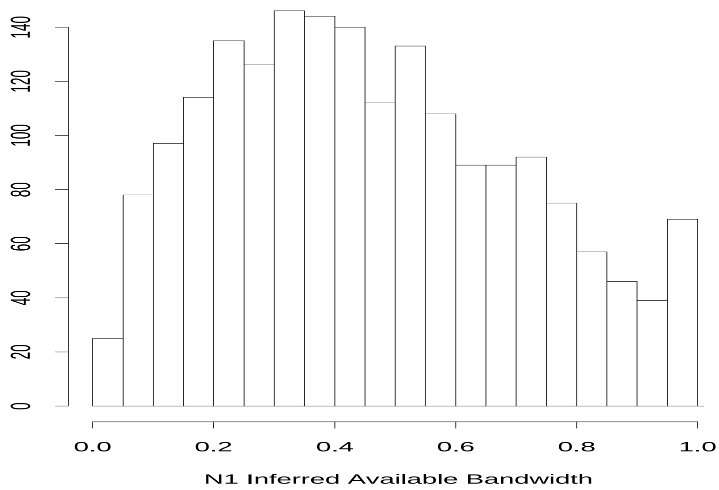

Figure 16.26: Distribution of $\mathcal{N}_1$ inferred available bandwidth (β)

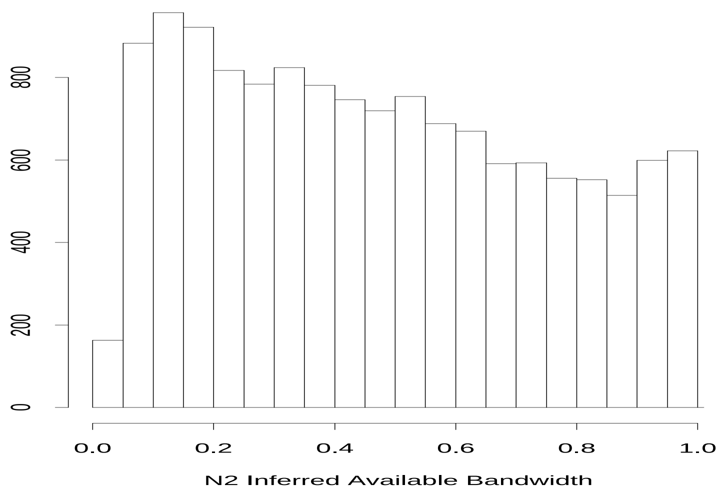

Figure 16.27: Distribution of $\mathcal{N}_2$ inferred available bandwidth (β)

to zero and one, respectively.[12] Clearly, Internet connections encounter a broad range of available bandwidths, ranging from very little to almost all. $\mathcal{N}_1$'s main mode lies at 0.30-0.35, corresponding to about two equivalent competing connections, while for $\mathcal{N}_2$ this shifts considerably downward, to about 0.10-0.15, or eight equivalent competing connections. The overall decrease in β between $\mathcal{N}_1$ and $\mathcal{N}_2$ is clear, though the $\mathcal{N}_2$ density diminishes less quickly than that of $\mathcal{N}_1$, indicating that for it, especially, the range of available bandwidth was indeed very broad. Unfortunately, it is difficult from these statistics to make a definitive statement about how available bandwidth changed over the course of 1995, because the use of bigger windows (§ 9.3) in $\mathcal{N}_2$ means that the notion of "equivalent connection" is different between the two datasets. It is not clear how we could adjust for this difference in order to directly compare the two.

Both densities exhibit two "edge" effects: a greatly diminished density at 0.0-0.05, and a second mode at 0.95–1.0. The first most likely reflects the measurement *bias* our experiment suffers from due to the limited lifetimes of each connection (§ 9.3): those connections for which very little bandwidth was available often did not finish within the allotted ten minutes, and thus do not figure into the measured distribution of β.

The second mode at 0.95–1.0 at first appears to indicate that sometimes a network path is completely quiescent, and packets sail along it without any cross traffic perturbing them. This, however, turns out to only sometimes be the case. Closer inspection of those connections with $\beta \approx 1$ reveals that many are connections with low bottleneck bandwidths. These connections very often are able to completely fill the bottleneck link, because, even if the network can provide only a few non-bottleneck resources to the connection, these still suffice to drive the bottleneck at capacity. That is, the connection requires only modest resources available elsewhere to saturate the bottleneck link and achieve the maximum possible end-to-end performance. We summarize this effect as: *If you only want to go slowly, the network often can provide enough resources for doing so.*

Figure 16.28 and Figure 16.29 show the same densities if we restrict the analysis to connections with $\rho_B \geq 100$ Kbyte/sec. We see that, for $\mathcal{N}_1$, doing so completely eliminates the secondary "all bandwidth available" peak, though, for $\mathcal{N}_2$, it only slightly diminishes it. The difference again appears due to the use of bigger windows in $\mathcal{N}_2$. Figure 16.30 shows the $\mathcal{N}_2$ densities if we restrict ourselves to $\rho_B \geq 250$ Kbyte/sec. Doing so eliminates the T1- and E1-limited connections, which with the bigger windows the $\mathcal{N}_2$ connections could often fill to capacity, much as the $\mathcal{N}_1$ connection could for the slower bottleneck links. Now, the second peak has disappeared, indicating that, at these speeds, the connections could no longer often utilize the entire bottleneck bandwidth.[13] We see that, overall, *as path bandwidths increase, proportionally less bandwidth is available to connections using the path.* This observation is not too surprising: higher bandwidths naturally attract higher traffic loads.

Our observations so far have been based on the load, λ_i, and the bottleneck transmission time, ϕ_i, per Eqn 16.3. Both are computed using the *central* bottleneck bandwidth estimate, ρ_B. The PBM algorithm, however, produces upper and lower *bounds* on the estimate, too, denoted by ρ_B^+ and ρ_B^- (Eqn 14.12). We can accordingly define λ_i^- (ϕ_i^-) and λ_i^+ (ϕ_i^+), based on the upper

[12] We do not discard these connections because sometimes only a slight error in ρ_B will lead to an "out of range" estimate for β, if the connection occurred at a time during which very little or almost all of the bandwidth was available. This point will be developed in more depth shortly.

[13] The depression at 0.0-0.05 has grown, too, a change likely due to the fact that, for high-bandwidth paths, a TCP connection can transfer 100 Kbyte in 10 minutes even in the face of many competing connections, so the measurement bias discussed earlier does not apply to such a large degree.

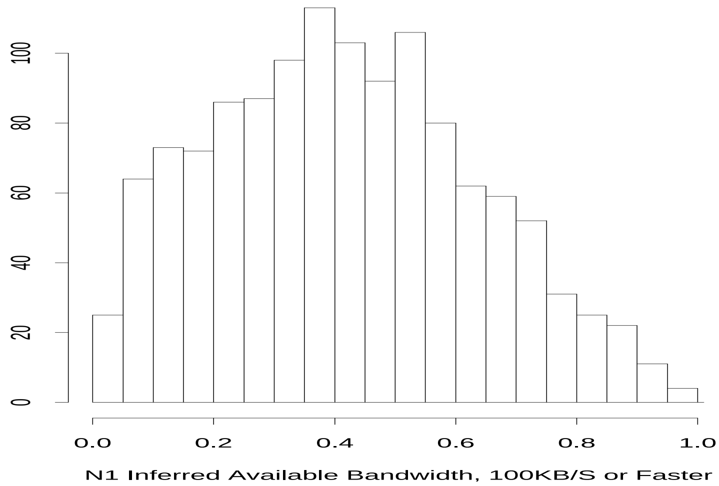

Figure 16.28: Distribution of $\mathcal{N}_1$ inferred available bandwidth (β) for connections with bottleneck rates exceeding 100 Kbyte/sec

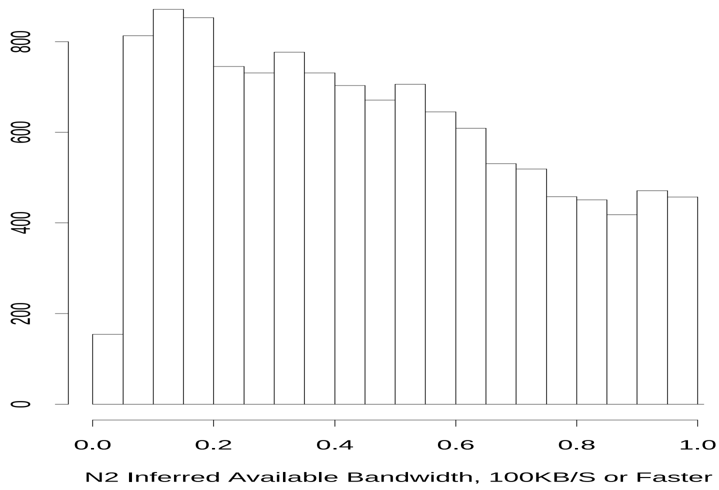

Figure 16.29: Distribution of $\mathcal{N}_2$ inferred available bandwidth (β) for connections with bottleneck rates exceeding 100 Kbyte/sec

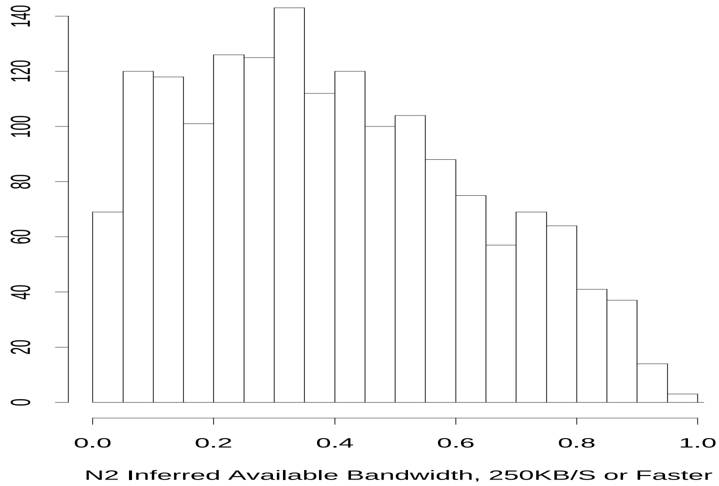

Figure 16.30: Distribution of $\mathcal{N}_2$ inferred available bandwidth (β) for connections with bottleneck rates exceeding 250 Kbyte/sec

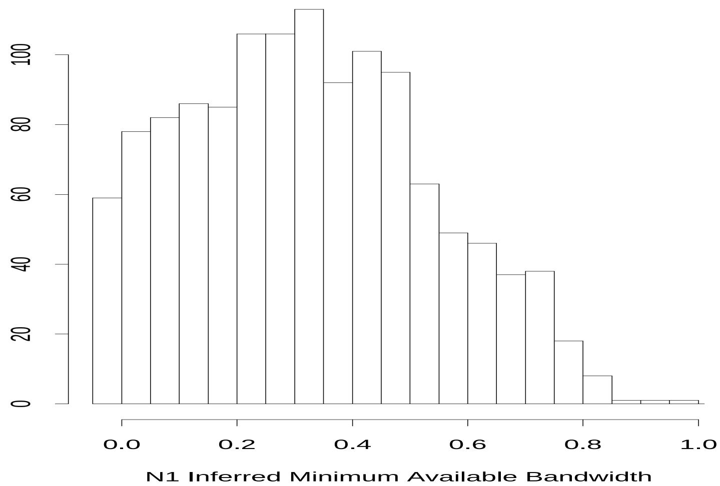

Figure 16.31: Distribution of $\mathcal{N}_1$ minimum inferred available bandwidth (β) for connections with bottleneck rates exceeding 100 Kbyte/sec

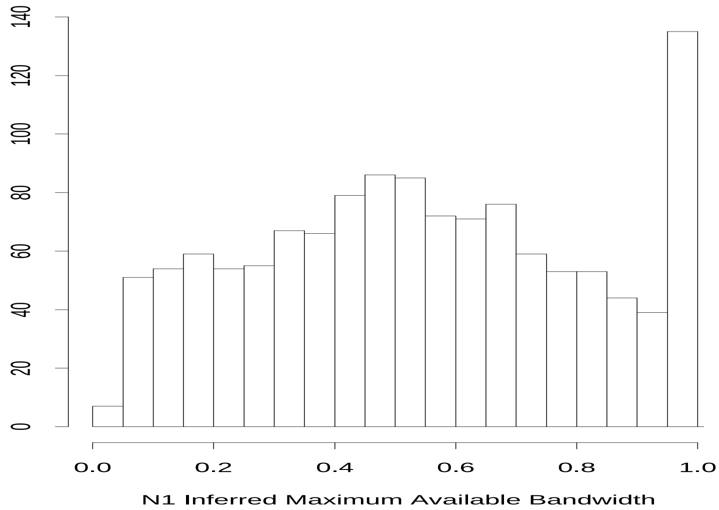

Figure 16.32: Distribution of $\mathcal{N}_1$ maximum inferred available bandwidth (β) for connections with bottleneck rates exceeding 100 Kbyte/sec

and lower bounds, respectively, and from them compute β^- and β^+, lower and upper bounds of the available bandwidth. Figure 16.31 and Figure 16.32 show the densities of β^- and β^+ for the connections in the $\mathcal{N}_1$ dataset with $\rho_B \geq 100$ Kbyte/sec.

The density of β^- fairly closely matches that of β given in Figure 16.28, but shifted about 0.05 to the left, except for the upper regime, which is shifted by about 0.15 to the left. The density of β^+, however, shows roughly the same shape shifted about 0.1 to the right, except for a striking spike at $\beta \approx 1$. This spike is telling: three-quarters of it is for $\beta^+ > 1$, which is an unphysical situation, namely, that the connection's load on the path exceeds the total variation observed on the path. Thus, the spike indicates that ρ_B^-, from which β^+ is derived, is *erroneously too low*. Because it is too low, the corresponding loads, λ_i^+, are too high. Furthermore, the loads can rapidly become *much* too high, due to self-clocking: if the connection is indeed transmitting at exactly the bottleneck rate, which self-clocking will promote in the absence of significant cross-traffic, then each packet's load will be zero, or perhaps will correspond to one additional packet at the bottleneck link if the receiver uses ack-every-other (so the window advances by two packets at a time). In this case, a slightly low estimate of ρ_B^- will result in a determination that the load continually builds up, since the bad estimate will imply that packets are being sent at a rate exceeding the bottleneck's capacity, and hence the queue at the bottleneck grows and grows (per Figure 16.13).

Consequently, we should not trust the variation between β and β^+ as reflecting the true error-bar range in β's density; but that between β and β^- does not suffer from this problem. Based on the latter, then, we conclude that the error in our estimates of β is about ± 0.1, with somewhat lower errors for small values of β, and somewhat higher errors for larger values. This level of error is not large enough to alter any of the conclusions drawn above.

As we might expect, we find that β is inversely correlated with data packet loss rate. For both $\mathcal{N}_1$ and $\mathcal{N}_2$, for connections with $\rho_B \geq 100$ Kbyte/sec, the coefficient of correlation between

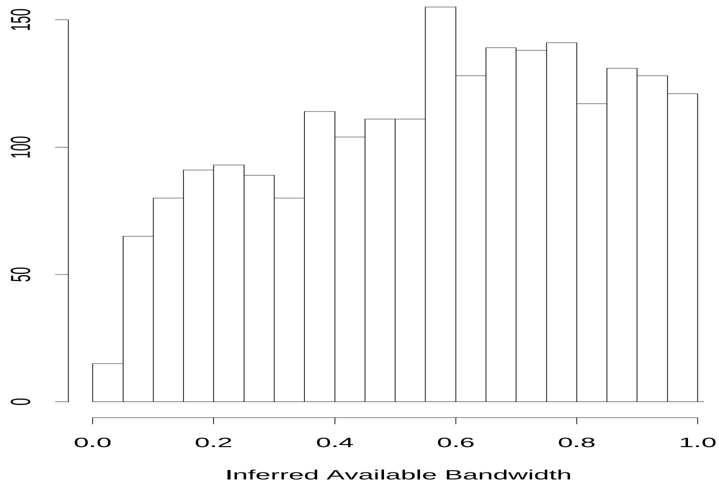

Figure 16.33: Distribution of $\mathcal{N}_2$ inferred available bandwidth (β) for U.S. connections

β and the loss rate is -0.36. This provides us with a solid connection between delay variation and packet loss, which agrees with the widely held assumption that most packet loss in the Internet is due to congestion (which will first lead to delay variations as queues build up). The connection is not overwhelmingly strong, however, which we would also expect, because delay variation need not lead to packet loss if the congested element contains sufficient buffer space to absorb the variation.

That β and loss rate are negatively correlated suggests that we might find significant regional variation in β, much as we did for loss rates in § 15.1. Indeed, we do. Figure 16.33 shows β for connections with $\rho_B \geq 100$ Kbyte/sec and with both sender and receiver sited in the United States. Figure 16.34 shows the same for sender and receiver both sited in Europe. Clearly, European sites suffer from much lower β's than their U.S. counterparts, with the mean (and median) European β at 40%, while for the U.S. connections, it is just under 60%.

The last aspect of available bandwidth we investigate is how it evolves over time. To do so, we group connections with the same source and destination hosts together, after eliminating any with $\rho_B < 100$ Kbyte/sec. For successive connections c and c' in each group, we compute the pair $\langle \Delta T_c, |\Delta\beta_c| \rangle$, where ΔT_c is the time between c and c', and $|\Delta\beta_c|$ is the magnitude of the difference between the computed β's for each connection.

After constructing these pairs, we sort them on ΔT_c and then compute $|\Delta\beta_c|$ smoothed using an exponentially-weighted moving average with $\alpha = 0.01$ and an initial value of 0. Figure 16.35 shows the resulting smoothed evolution for the $\mathcal{N}_2$ dataset. (The $\mathcal{N}_1$ dataset exhibits a similar evolution.) We see that $|\Delta\beta_c|$ almost immediately rises to about 0.12, which is somewhat higher than the error range we estimated for β above, but not greatly higher.[14] This level is sus-

[14]The exponential smoothing, along with starting the averaging with an value of 0, limits how rapidly the plot can reach this level. This is what creates the plotting artifact of what appears to be a rapid climb, falsely suggesting that $|\Delta\beta_c|$ is significantly smaller for very low inter-connection times. A more sound interpretation is that even for very low inter-connection times, we will usually find $|\Delta\beta_c|$ already quite close to 0.12.

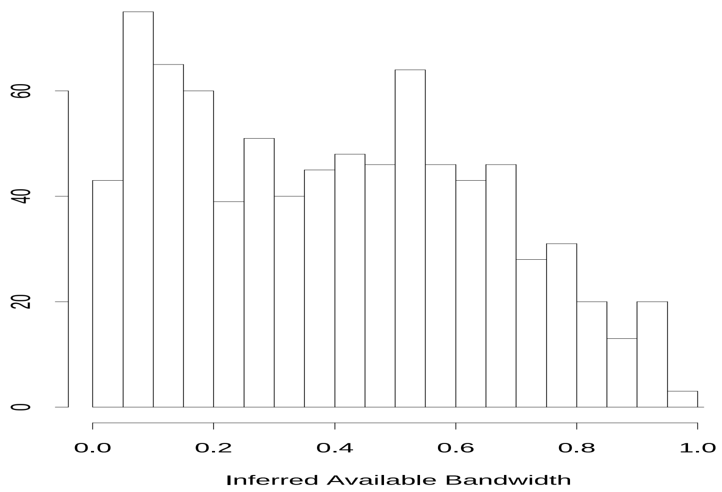

Figure 16.34: Distribution of $\mathcal{N}_2$ inferred available bandwidth (β) for European connections

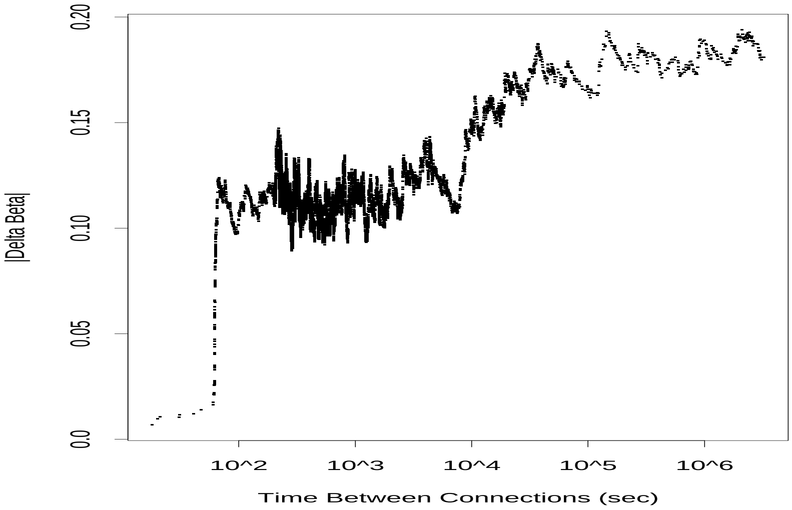

Figure 16.35: Evolution of difference between inferred available bandwidth (β) for successive connections

tained for a number of hours, after which it increases markedly, by about 50%. The transition no doubt coincides with the diurnal cycle we noted in § 15.1: the network is much more congested during working hours than during off hours. Since the predictive power is, qualitatively, fairly good for time scales of several hours, we conclude that transport connections can fruitfully cache information regarding a path's available bandwidth for use in subsequent connections.