Chapter 14

Bottleneck Bandwidth

In this chapter we discuss one of the fundamental properties of a network connection, the *bottleneck bandwidth* that sets the upper limit on how quickly the network can deliver the sender's data to the receiver. In § 14.1 we discuss the general notion of bottleneck bandwidth and why we consider it a fundamental quantity. § 14.2 discusses "packet pair," the technique used in previous work, and § 14.3 discusses why for our study we gain significant benefits using "receiver-based packet pair," in which the measurements used in the estimation are those recorded by the receiver, rather than the ack "echoes" that the sender later receives.

While packet pair often works well, in § 14.4 we illustrate four difficulties with the technique, three surmountable and the fourth fundamental. Motivated by these problems, we develop a robust estimation algorithm, "packet bunch modes" (PBM). To do so, we first in § 14.5 discuss an alternative estimation technique based on measurements of the "peak rate" (PR) achieved by the connection, for use in calibrating the PBM technique, which we then develop in detail in § 14.6. In § 14.7, we analyze the estimated bottleneck bandwidths for the Internet paths in our study, and in § 14.8 we finish with a comparison of the efficacy of the various techniques.

14.1 Bottleneck bandwidth as a fundamental quantity

Each element in the end-to-end chain between a data sender and the data receiver has some *maximum rate* at which it can forward data. These maxima may arise directly from physical properties of the element, such as the frequency bandwidth of a wire, or from more complex properties, such as the minimum amount of time required by a router to look up an address to determine how to forward a packet. The first of these situations often dominates, and accordingly the term *bandwidth* is used to denote the maximum rate, even if the maximum does not come directly from a physical bandwidth limitation.

Because sending data involves forwarding the data along an end-to-end *chain* of networking elements, the *slowest* element in the entire chain sets the *bottleneck bandwidth*, i.e., the maximum rate at which data can be sent along the chain. The usual assumption is that the bottleneck element is a network *link* with a limited bandwidth, although this need not be the case.

Note that from our data we cannot say anything meaningful about the *location* of the bottleneck along the network path, since our methodology gives us only end-to-end measurements (though see § 15.4). Furthermore, there may be multiple elements along the network path, each limited to the same bottleneck rate. Thus, our analysis is confined to an assessment of the bottleneck bandwidth as an end-to-end path property, rather than as the property of a particular element in the path.

We must make a crucial distinction between *bottleneck* bandwidth and *available* bandwidth. The former gives an upper bound on how fast a connection can *possibly* transmit data, while the less-well-defined latter term denotes how fast the connection in fact *can* transmit data, or in some cases how fast it *should* transmit data to preserve network stability, even though it could transmit faster. Thus, the available bandwidth never exceeds the bottleneck bandwidth, and can in fact be much smaller. Bottleneck bandwidth is often presumed to be a fairly static quantity, while available bandwidth is often recognized as intimately reflecting current network traffic levels (congestion). Using the above terminology, the bottleneck location(s), if we were able to pinpoint them, would generally not change during the course of a connection, unless the network path used by the connection underwent a routing changes. But the networking element(s) limiting the available bandwidth might readily change over the lifetime of a connection.

TCP's congestion avoidance and control algorithms reflect an attempt to confine each connection to the available bandwidth. For this purpose, the bottleneck bandwidth is essentially irrelevant. For connection *performance*, however, the bottleneck bandwidth is a fundamental quantity, because it indicates a limit on what the connection can hope to achieve. If the sender tries to transmit any faster, not only is it guaranteed to fail, but the additional traffic it generates in doing so will either lead to queueing delays somewhere in the network, or packet drops, if the overloaded element lacks sufficient buffer capacity.

We discuss available bandwidth further in § 16.5, and for the remainder of this chapter focus on assessing bottleneck bandwidth.

The bottleneck bandwidth is further a fundamental quantity because it determines what we term the *self-interference time-constant*, Q_b. Q_b measures the amount of time required to forward a given packet through the bottleneck element. Thus, Q_b is identical to the service time at the bottleneck element; we use the term "self-interference time-constant" instead because of the central role Q_b plays in determining when packet transit times are necessarily correlated, as discussed below.

If a packet carries a total of b bytes and the bottleneck bandwidth is ρ_B byte/sec, then:

$$Q_b = \frac{b}{\rho_B} \tag{14.1}$$

in units of seconds. We use the term "self-interference" because if the sender transmits two b-byte packets with an interval $\Delta T_s < Q_b$ between them, then the second one is guaranteed to have to wait behind the first one at the bottleneck element (hence the use of "Q" to denote "queueing").

We use the notation Q_b instead of the more functional notation $Q(b)$ because we will assume unless otherwise stated that, for a particular trace pair, b is fixed to the maximum segment size (MSS; § 9.2.2). We note that full-sized packets are *larger* than MSS, due to overhead from transport, network, and link-layer headers. However, while it might at first appear that this overhead is known (except for the link-layer) and can thus be safely added into b, if the bottleneck link along a path uses *header compression* (§ 13.3) then the header as transmitted might take much less data than would appear from tallying the number of bytes in the header. Since many of the most significant bottleneck links in our study also use header compression, we decided to perform all of our analysis of the bottleneck bandwidth in terms of the maximum rate at which a connection can transmit *user data*.

For our measurement analysis, accurate assessment of Q_b is critical. Suppose we observe a sender transmitting p_1 and p_2, both b bytes in size, and that they are sent an interval ΔT_s apart. If

$$\Delta T_s < Q_b,$$

then we know that p_2 had to wait a time $Q_b - \Delta T_s$ at the bottleneck element B while p_1 was being forwarded across B. (This assumes that p_1 and p_2 take the same path through the network, a point we address in detail later in this chapter.)

Thus, if $\Delta T_s < Q_b$, the delays experienced by p_1 and p_2 are *perforce correlated*. If $\Delta T_s \geq Q_b$, then if p_2 experiences greater delay than p_1, the increase is not due to self-interference but some other source (such as additional traffic from other connections, or processing delays).

We use Q_b to analyze packet timings and remove self-interference effects in Chapter 16. In this chapter, we focus on sound estimation of Q_b, as we must have this in order for the subsequent timing analysis to be likewise sound.

14.2 Packet pair

The fundamental idea behind the *packet pair* estimation technique is that, if two packets are transmitted by the sender with an interval $\Delta T_s < Q_b$ between them, then when they arrive at the bottleneck they will be spread out in time by the transmission delay of the first packet across the bottleneck: after completing transmission through the bottleneck, their spacing will be exactly Q_b. Barring subsequent delay variations (due to downstream queueing or processing lulls), they will then arrive at the receiver spaced not ΔT_s apart, but $\Delta T_r = Q_b$. The size b then enables computation of ρ_B via Eqn 14.1.[1]

The principle of the bottleneck spacing effect was noted in Jacobson's classic congestion paper [Ja88], where it in turn leads to the "self-clocking" mechanism (§ 9.2.5). Keshav subsequently formally analyzed the behavior of packet pair for a network in which all of the routers obey the "fair queueing" scheduling discipline, and developed a provably stable flow control scheme based on packet pair measurements [Ke91].[2] Both Jacobson and Keshav were interested in estimating *available* rather than *bottleneck* bandwidth, and for this *variations* from Q_b due to queueing are of primary concern (§ 16.5). But if, as for us, the goal is to estimate ρ_B, then these variations instead become noise we must deal with.

To use Jacobson's self-clocking model to estimate bottleneck bandwidth requires an assumption that delay variation in the network is small compared to Q_b. Using Keshav's scheme requires fair queueing. Internet paths, however, often suffer considerable delay variation (Chapter 16), and Internet routers do not employ fair queueing. Thus, efforts to estimate ρ_B using packet pair must deal with considerable noise issues. The first step in dealing with measurement noise is

[1] If the two packets in the pair have different sizes b_1 and b_2, then which to use depends on how we interpret the timestamps for the packets. If the timestamps reflect when the packet *began* to arrive at the packet filter's monitoring point, then b_1 should be used, since that is how much data was transmitted between the timestamps of the two packets. If the timestamps reflect when the packet *finished* arriving, then b_2 should be used. In practice, a packet's timestamp is recorded some time *after* the packet has finished arriving, per § 10.2, and so if $b_1 \neq b_2$, `tcpanaly` uses b_2.

[2] Keshav also coined the term "packet pair."

to analyze as large a number of pairs as feasible, with an eye to the tradeoff between measurement accuracy and undue loading of the network by the measurement traffic.[3]

Bolot used a stream of packets sent at fixed intervals to probe several Internet paths in order to characterize delay and loss behavior [Bo93]. He measured round-trip delay of UDP echo packets and, among other analyses, applied the packet pair technique to form estimates of bottleneck bandwidths. He found good agreement with known link capacities, though a limitation of his study is that the measurements were confined to a small number of Internet paths. One of our goals is to address this limitation by determining how well packet pair techniques work across diverse Internet conditions.

Recent work by Carter and Crovella also investigates the utility of using packet pair in the Internet for estimating bottleneck bandwidth [CC96a]. Their work focusses on `bprobe`, a tool they devised for estimating bottleneck bandwidth by transmitting 10 consecutive ICMP echo packets and recording the arrival times of the corresponding replies. `bprobe` then repeats this process with varying (and carefully chosen) packet sizes. Much of the effort in developing `bprobe` concerns how to filter the resulting raw measurements in order to form a solid estimate. `bprobe` currently filters by first widening each estimate into an interval by adding an (unspecified) error term, and then finding the point at which the largest number of intervals overlap. The authors also undertook to calibrate `bprobe` by testing its performance for a number of Internet paths with known bottlenecks. They found in general it worked well, though some paths exhibited sufficient noise to sometimes produce erroneous estimates. Finally, they note that measurements made using larger echo packets yielded more accurate estimates than those made using smaller packets, which bodes well for our interest in measuring Q_b for $b = \text{MSS}$.

One limitation of both studies is that they were based on measurements made only at the data sender (§ 9.1.3). Since in both studies the packets echoed back from the remote end were the same size as those sent to it, neither analysis was able to distinguish whether the bottleneck along the forward and reverse paths was the same, or whether it was present in only one direction. The bottleneck could differ in the two directions due the packets traversing different physical links because of asymmetric routing (§ 8), or because some media, such as satellite links, can have significant bandwidth asymmetries depending on the direction traversed [DMT96].

For the study in [CC96a], this is not a problem, because the authors' ultimate goal was to determine which Web server to pick for a document available from a number of different servers. Since Web transfers are request/response, and hence bidirectional (albeit potentially asymmetric in the volume of data sent in each direction), the bottleneck for the combined forward and reverse path is indeed a figure of interest. For general TCP traffic, however, this is not always the case, since for a unidirectional transfer—especially for FTP transfers, which can sometimes be quite huge [PF95]—the data packets sent along the forward path are much larger than the acks returned along the reverse path. Thus, even if the reverse path has a significantly lower bottleneck bandwidth, this is unlikely to limit the connection's maximum rate. However, for estimating bottleneck bandwidth by measuring TCP traffic a second problem arises: if the only measurements available are those at the sender, then ack compression (§ 16.3.1) can significantly alter the spacing of the small ack packets as they return through the network, distorting the bandwidth estimate. We investigate the degree of this problem below.

[3] Gathering large samples, however, can conflict with another goal, that of forming an estimate *quickly*, briefly discussed at the end of the chapter.

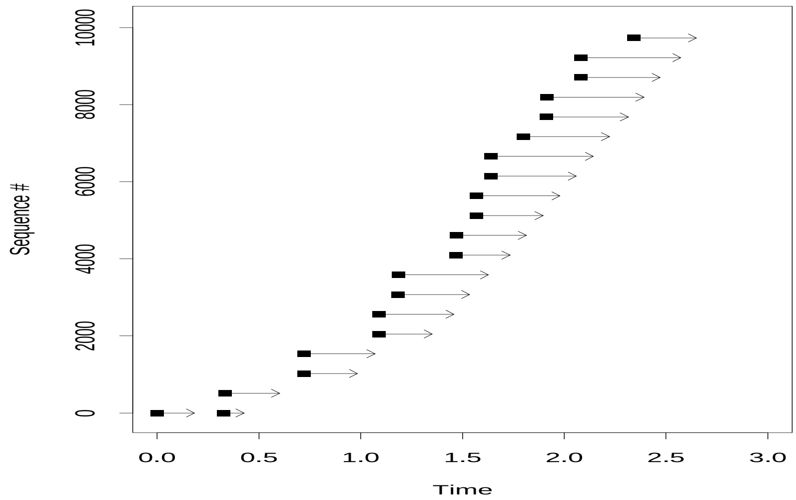

Figure 14.1: Paired sequence plot showing timing of data packets at sender (black squares) and when received (arrowheads)

14.3 Receiver-based packet pair

For our analysis, we consider what we term *receiver*-based packet pair (RBPP), in which we look at the pattern of data packet arrivals at the receiver. We also utilize knowledge of the pattern in which the data packets were originally sent, so we assume that the receiver has full timing information available to it. In particular, we assume that the receiver knows when the packets sent were *not* stretched out by the network, and can reject these as candidates for RBPP analysis.

RBPP is considerably more accurate than sender-based packet pair (SBPP; cf. § 14.2), since it eliminates the additional noise and possible asymmetry of the return path, as well as noise due to delays in generating the acks themselves (§ 11.6.4). Figure 14.1 shows a *paired sequence plot* for data transferred over a path known to have a 56 Kbit/sec bottleneck link. The centers of the filled black squares indicate the times at which the sender transmitted the successive data packets, and the arrowheads point to the times at which they arrived at the receiver. (We have adjusted the relative clock offset per the methodology given in § 12.5). The packet pair effect is quite strong: while the sender tends to transmit packets in groups of two back-to-back (due to slow start opening the congestion window), this timing structure has been completely removed by the time the packets arrive, and instead they come in at a nearly constant rate of about 6,200 byte/sec.

Figure 14.2 shows the same trace pair with the acknowledgements added. They are offset slightly lower than the sequence number they acknowledge for legibility. The arrows start at the point in time at which the ack was generated by the receiver, and continue until received by the sender. We can see that some acks are generated immediately, but others (such as 4,096) are delayed. Furthermore, there is considerable variation among the transit times of the acks, even though *they* are almost certainly too small to be subject to stretching at the bottleneck link along the return path. If we follow the ack arrowheads by eye, it is clear that the strikingly smooth pattern in Figure 14.1

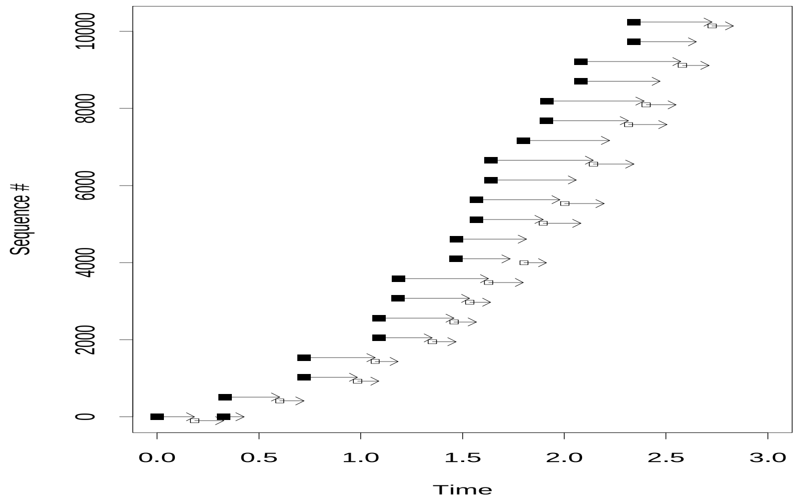

Figure 14.2: Same plot with acks included

has been blurred by the ack delays, which have nothing to do with the quantity of interest, namely Q_b on the forward path.

14.4 Difficulties with packet pair

As shown in the Bolot and Carter/Crovella studies ([Bo93, CC96a]), packet pair techniques often provide good estimates of bottleneck bandwidth. We are interested both in estimating the bottleneck bandwidth of the Internet paths in our study, and, furthermore, whether the packet-pair technique is robust enough that an Internet transport protocol might profitably use it in order to make decisions based on Q_b.

A preliminary investigation of our data revealed four potential problems with packet pair techniques, even if receiver-based. Three of these can often be satisfactorily addressed, but the fourth is more fundamental. We discuss each in turn.

14.4.1 Out-of-order delivery

The first problem stems from the fact that, for some Internet paths, out-of-order packet delivery occurs quite frequently (§ 13.1). Clearly, packet pairs delivered out of order completely destroy the packet pair technique, since they result in $\Delta T_r < 0$, which then leads to a negative estimate for ρ_B. The receiver sequence plot in Figure 14.3 illustrates the basic problem. (Compare with the clean arrivals in Figure 14.1.)

Out-of-order delivery is symptomatic of a more general problem, namely that the two packets in a pair may not take the same route through the network, which then violates the assumption that the second queues behind the first at the bottleneck. In a sense, out-of-order delivery is a blessing, because the receiver can usually *detect* the event (based on sequence numbers, and

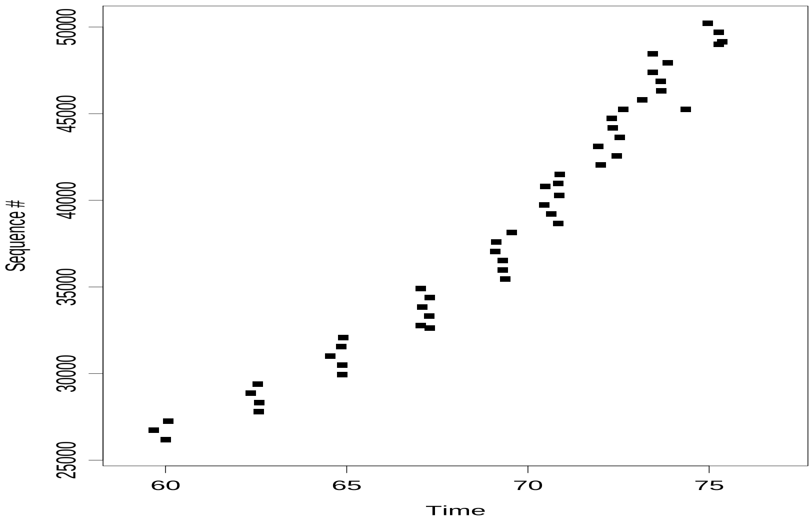

Figure 14.3: Receiver sequence plot illustrating difficulties of packet-pair bottleneck bandwidth estimation in the presence of out-of-order arrivals

possibly IP "id" fields for retransmitted packets; cf. § 10.5). More insidious are packets pairs that traverse different paths but still arrive in order. The interval computed from their arrivals may have nothing to do with the bottleneck bandwidth, and yet it is difficult to recognize this case and discard the measurement from subsequent analysis. We discuss a particularly problematic instance of this problem in § 14.4.4 below.

14.4.2 Limitations due to clock resolution

Another problem relates to the receiver's clock resolution, C_r (§ 12.3). C_r can introduce large margins of error around estimates of ρ_B. Suppose two b-byte packets arrive at the receiver with a spacing of ΔT_r. We want to estimate ρ_B from Eqn 14.1 using

$$\begin{aligned} \Delta T_r &= Q_b \\ &= \frac{b}{\rho_B}, \end{aligned}$$

and hence

$$\rho_B = \frac{b}{\Delta T_r}. \tag{14.2}$$

However, we cannot measure ΔT_r exactly, but only estimate an interval in which it lies, using:

$$\max(\Delta\widetilde{T}_r - C_r, 0) \leq \Delta\widehat{T}_r \leq \Delta\widetilde{T}_r + C_r, \tag{14.3}$$

where $\Delta\widetilde{T}_r$ is the value reported by the receiver's clock for the spacing between the two packets. Combining Eqn 14.2 with Eqn 14.3 gives us:

$$\begin{aligned} \widehat{\rho}_b &= \frac{b}{\Delta\widehat{T}_r}, \\ \frac{b}{\Delta\widetilde{T}_r + C_r} \leq \ \widehat{\rho}_b &\leq \frac{b}{\max(\Delta\widetilde{T}_r - C_r, 0)}. \end{aligned} \tag{14.4}$$

In the case where $\Delta\widetilde{T}_r \leq C_r$, i.e., the two packets arrived with the clock advancing at most once, we cannot provide any upper bound on $\widehat{\rho}_b$ at all. Thus, for example, if $C_r = 10$ msec, a common value on older hardware (§ 12.4.2), then for $b = 512$ bytes, from the arrival of a single packet pair we cannot distinguish between

$$\rho_B = \frac{512}{0.010 \text{ sec}} = 51,200 \text{ byte/sec},$$

and

$$\rho_B = \infty.$$

This means we cannot distinguish between a fairly pedestrian T1 link of under 200 Kbyte/sec, and a blindingly fast (today!) OC-12 link of about 80 Mbyte/sec.

For $C_r = 1$ msec, the threshold rises to 512,000 byte/sec, still much too low for meaningful estimation for high-speed networks. For today's networks, $C_r = 100$ μsec almost allows us to distinguish between T3 speeds of a bit over 5 Mbyte/sec and higher speeds. Since some of the clocks in our study had finer resolution, we view this problem as tractable with today's (better) hardware. It is not clear, however, whether in the future processor clock resolution will grow finer at a rate to match how network bandwidths grow faster (and thus Q_b decreases).

While some of today's hardware provides sufficient resolution for packet-pair analysis, other platforms do not, so we still need to find a way to deal with low-resolution clocks. In line with the argument in the previous paragraph, doing so also potentially benefits measurement of future networks, since their bandwidth growth may outpace that of clock resolution.

A basic technique for coping with poor clock resolution is to use packet *bunch* rather than packet pair.[4] The idea behind packet bunch, in which $k \geq 2$ back-to-back packets are used, is that bunches should be less prone to noise, since individual packet variations are smoothed over a single large interval rather than $k - 1$ small intervals. This idea has not been thoroughly tested, and one might argue the opposite: if packets are occasionally subject to large transient delays due to bursts of cross traffic, than the larger k is, the greater the likelihood that a bunch will be disrupted by a significant delay, leading to underestimation of ρ_B. We investigate this concern below. However, another benefit of packet bunch is that the overall time interval ΔT_r^k spanned by the k packets will be about $k - 1$ times larger than that spanned by a single packet pair. Accordingly, by choosing sufficiently large k we can diminish the adverse effects of poor clock resolution, except for the problem mentioned above of encountering spurious delays and underestimating ρ_B as a result.

[4] The term "packet bunch" has been in informal use for at least several years; however, we were unable to find any appearance of it in the networking literature. The *notion* appears in [BP95a], in the discussion of the "Vegas-*" variant, which attempts to estimate available bandwidth using a four-packet version of packet pair; and in [Ho96], which uses an estimate derived from the timing of three consecutive acks.

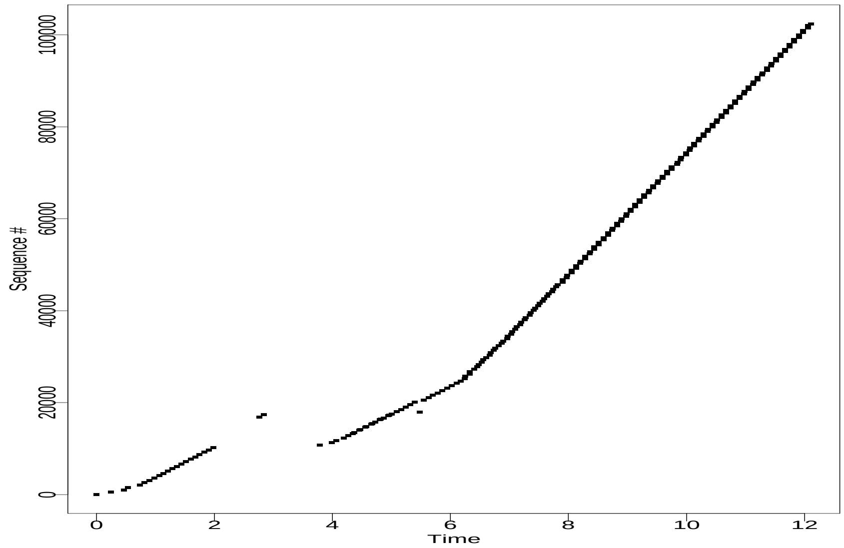

Figure 14.4: Receiver sequence plot showing two distinct bottleneck bandwidths

14.4.3 Changes in bottleneck bandwidth

Another problem that *any* bottleneck bandwidth estimation must deal with is the possibility that the bottleneck *changes* over the course of the connection. Figure 14.4 shows a trace in which this happened. We have shown the entire trace, but only the data packets and not the corresponding acks. While the details are lost, the eye immediately picks out a transition between one overall slope to another, just after $T = 6$. The first slope corresponds to about 6,600 byte/sec, while the second is about 13,300 byte/sec, and increase of about a factor of two.

For this example, we know enough about one of the endpoints (`lbli`) to fully describe what occurred. `lbli`'s Internet connection is via an ISDN link. The link has two *channels*, each nominally capable of 64 Kbyte/sec. When `lbli` initially uses the ISDN link, the router only activates one channel (to reduce the expense). However, if `lbli` makes sustained use of the link, then the router activates the second channel, doubling the bandwidth.

While for this particular example the mechanism leading to the bottleneck shift is specific to the underlying link technology, the *principle* that the bottleneck can change with time is both important and general. It is important to detect such an event, because it has a major impact on the ensuing behavior of the connection. Furthermore, bottlenecks can shift for reasons other than multi-channel links. In particular, routing changes might alter the bottleneck in a significant way.

Packet pair studies to date have focussed on identifying a *single* bottleneck bandwidth [Bo93, CC96a]. Unfortunately, in the presence of a bottleneck shift, any technique shaped to estimate a single, unchanging bottleneck will fail: it will either return a bogus compromise estimate, or, if care is taken to remove noise, select one bottleneck and reject the other. In both cases, the salient fact that the bottleneck shifted is overlooked. We attempt to address this problem in the development of our robust estimation algorithm (§ 14.6).

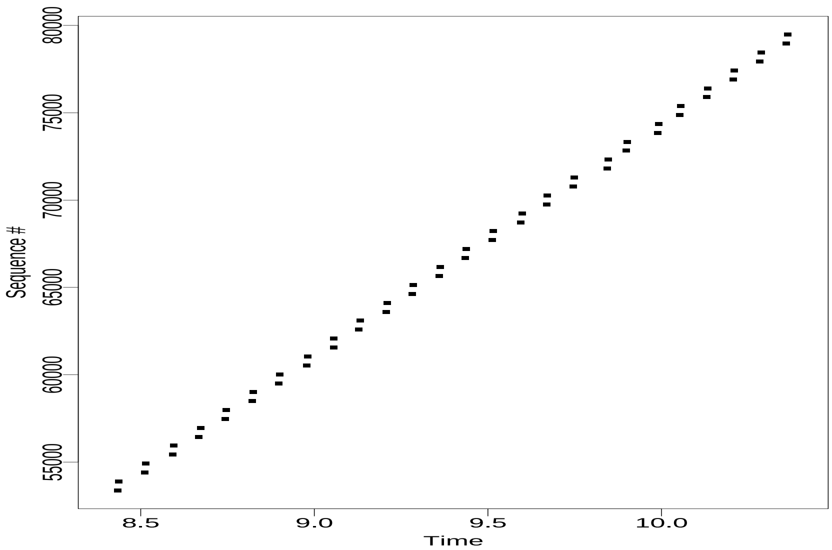

Figure 14.5: Enlargement of part of the previous figure

14.4.4 Multi-channel bottleneck links

We now turn to a more fundamental problem with packet-pair techniques, namely bottleneck estimation in the face of multi-channel links. Here we do not concern ourselves with the problem of detecting that the bottleneck has *changed* due to the activation or deactivation of the link's additional channel (§ 14.4.3). We instead illustrate a situation in which packet pair yields *incorrect overestimates* even in the absence of any delay noise.

Figure 14.5 expands a portion of Figure 14.4. The slope of the large linear trend in the plot corresponds to 13,300 byte/sec, as earlier noted. However, we see that the line is actually made up of pairs of packets. Figure 14.6 expands the plot again, showing quite clearly the pairing pattern. The slope between the pairs of packets corresponds to a data rate of about 160 Kbyte/sec, even though we know that the ISDN link has a hard limit of 128 Kbit/sec = 16 Kbyte/sec, a factor of ten smaller! Clearly, an estimate of

$$\widehat{\rho}_b \approx 160 \text{ Kbyte/sec}$$

must be wrong, yet that is what a packet-pair calculation will yield.

The question then is: where is the spacing corresponding to 160 Kbyte/sec coming from? A clue to the answer lies in the number itself. It is not far below the user data rates achieved over T1 circuits, typically on the order of 170 Kbyte/sec. It is as though every other packet were immune to queueing behind its predecessor at the known 16 Kbyte/sec bottleneck, but instead queued behind it at a downstream T1 bottleneck.

Indeed, this is exactly what is happening. As discussed in § 14.4.3, the bottleneck ISDN link has two channels. These operate in *parallel*. That is, when the link is idle and a packet arrives, it goes out over the first channel, and when another packet arrives shortly after, it goes out over the *other* channel. If a third packet then arrives, it has to wait until one of the channels becomes free. Effectively, it is queued behind not its immediate predecessor but its predecessor's predecessor, the

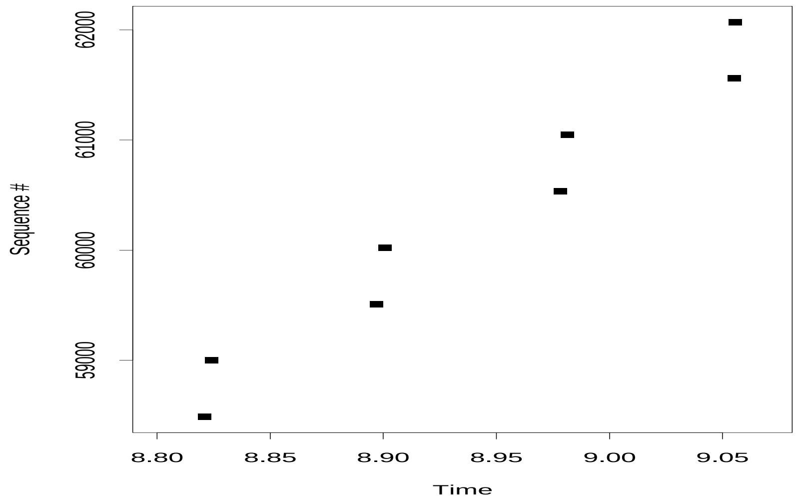

Figure 14.6: Enlargement of part of the previous figure

first packet in the series, and it is queued not for a 16 Kbyte/sec link but for an 8 Kbyte/sec channel making up just part of the link.

As queues build up at the router utilizing the multi-channel link, often both channels will remain busy for an extended period of time. In this case, additional traffic arriving at the router, or processing delays, can alter the "phase" between the two channels, meaning the offset between when the first begins sending a packet and when the second does so. Thus, we do not always get an arrival pattern clearly reflecting the downstream bottleneck as shown in Figure 14.6. We can instead get a pairing pattern somewhere between the downstream bottleneck and the true bottleneck. Figure 14.7 shows an earlier part of the same connection where a change in phase quite clearly occurs a bit before $T = 8$. Here the pair slope shifts from about 23 Kbyte/sec up to 160 Kbyte/sec. Note that the overall rate at which new data arrives at the receiver has not changed at all during this transition, only the fine-scale timing structure has changed.

We conclude that, in the presence of multi-channel links, packet-pair techniques can give completely misleading estimates for ρ_B. Worse, these estimates will often be much too high. The fundamental problem is the assumption with packet pair that there is only a single path through the network, and that therefore packets queue behind one another at the bottleneck.

We should stress that the problem is more general than the circumstances shown in this example, in two important ways. First, while in this example the parallelism leading to the estimation error came from a single link with two separate (and parallel) physical channels, the exact same effect could come from a router that balances its outgoing load across two different links. If these links have different propagation times, then the likely result is out-of-order arrivals, which can be detected by the receiver and removed from the analysis (§ 14.4.1). But if the links have equal or almost equal propagation times, then the parallelism they offer can completely obscure the true bottleneck bandwidth.

Second, it may be tempting to dismiss this problem as correctable by using packet bunch

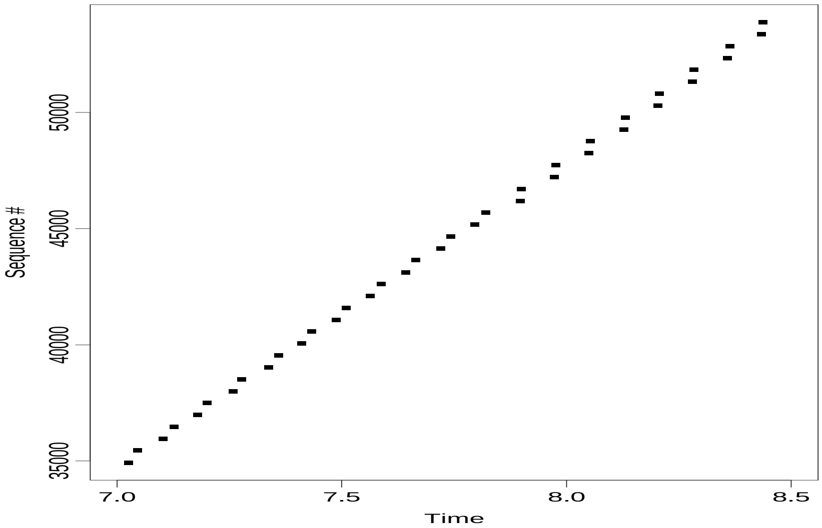

Figure 14.7: Multi-channel phasing effect

(§ 14.4.2) with $k = 3$ instead of packet pair. This argument is not compelling without further investigation, however, because packet bunch is potentially more prone to error; and, more fundamentally, $k = 3$ only works if the parallelism comes from *two* channels. If it came from *three* channels (or load-balancing links), then $k = 3$ will still yield misleading estimates.

We now turn to developing techniques to address these difficulties.

14.5 Peak rate estimation

In this section we discuss a simple, cheap-to-compute, and not particularly accurate technique for estimating the bottleneck bandwidth along a network path. We term this technique *peak rate* and subsequently refer to it as PR. Our interest in PR lies in providing *calibration* for the robust technique developed in the next section, based on packet-bunch modes ("PBM"). We develop two PR-based estimates, a "conservative" estimate, $\widehat{\mathrm{PR}}^c$, very unlikely to be an overestimate, and an "optimistic" estimate, $\widehat{\mathrm{PR}}^o$, which is more likely to be accurate but is also prone to overestimation. Armed with these estimates, we then can compare them with results given by PBM. If the robust technique yields an estimate less than $\widehat{\mathrm{PR}}^c$, or higher than $\widehat{\mathrm{PR}}^o$, then the discrepancy merits investigation. If they generally agree, then perhaps we can use the simpler PR techniques instead of PBM without losing accuracy (though it would be surprising to find that PR techniques suffice, per the discussion below).

PR is based on the observation that the peak rate the connection ever manages to transmit along the path should give a lower bound on the bottleneck rate. PR is a necessarily *stressful* technique in that it requires loading the network to capacity to assure accuracy. As such, we would prefer not to use PR as an active measurement methodology, but it works fine for situations in which the measurements being analyzed are due to traffic initiated for some reason other than bottleneck measurement. Thus, PR makes sense as a candidate algorithm for adding to a transport protocol.

In contrast, packet pair and PBM do not necessarily require stressing the network for accuracy, so they are attractive both as additions to transport protocols to aid in their decision-making, and as independent network analysis tools.

At its simplest, PR consists of just dividing the amount of data transferred by the duration of the connection. This technique, however, often grossly underestimates the true bottleneck bandwidth, because transmission lulls due to slow-start, insufficient window, or retransmission timeouts can greatly inflate the connection duration.

To reduce the error in PR requires confining the proportion of the connection on which we calculate the peak rate to a region during which none of these lulls impeded transmission. Avoiding slow-start and timeout delays is easy, since these regions are relatively simple to identify. Identifying times of insufficient window, however, is more difficult, because the correct window is a function of both the round-trip time (RTT) and the available bandwidth, and the latter is shaped in part by the bottleneck bandwidth, which is what we are trying to estimate.

If the connection was at some point not window-limited, then by definition it achieved a sustained rate (over at least one RTT) at or exceeding the available capacity. Since the hope embodied in PR is that at some point the available capacity matched the bottleneck bandwidth, we address the problem of insufficient window by forming our estimate from the maximum rate achieved over a single RTT.

`tcpanaly` computes a PR-based estimate by advancing through the data packet arrivals at the TCP receiver as follows. For each arrival, it computes the amount of data (in bytes) that arrived between that arrival and the next data packet coming just *beyond* the edge of a temporal window equal to the minimum RTT, $\mathrm{RTT}_{\mathrm{min}}$. ($\mathrm{RTT}_{\mathrm{min}}$ is computed as the smallest interval between a full-sized packet's departure from the sender and the arrival at the sender of an acknowledgement for that packet.) Suppose we find B bytes arrived in a total time $\Delta T_r > \mathrm{RTT}_{\mathrm{min}}$, and that the interval spanned by the departure of the packets when transmitted by the sender is ΔT_s.[5] Finally, if any of the packets arrived out of order, then we exclude the group of packets from any further analysis.

Otherwise, we compute the *expansion factor*

$$\xi_{s,r} = \frac{\Delta T_r + C_r}{\Delta T_s + C_s}, \tag{14.5}$$

where C_s and C_r are the resolutions of the sender's and receiver's clocks (§ 12.3). $\xi_{s,r}$ measures the factor by which the group of packets was spread out by the network. If less than 1, then the packets were *not* spread out by the network and hence not shaped by the bottleneck. Thus, calculations based on their arrival times should not be used in estimating the bottleneck. In practice, however, two effects complicate the simple rule of rejecting timings if $\xi_{s,r} < 1$. The first is that, if C_s is considerably different (orders of magnitude larger or smaller) than C_r, then $\xi_{s,r}$ can vary considerably, even if the magnitudes of ΔT_r and ΔT_s are close. The second problem is that sometimes due to "self-clocking" (§ 9.2.5), a connection rapidly settles into a pattern of transmitting packets at very close to the bottleneck bandwidth, in which case we might find $\xi_{s,r}$ slightly less than 1 even though it allows for a solid estimate of ρ_B. To address these concerns, we use a slightly different definition

[5] Here, B does *not* include the bytes carried by the first packet of the group, since we assume that the packet timestamps reflect when packets *finished* arriving, so the first packet's bytes arrived before the point in time indicated by its timestamp. Also see the footnote in § 14.2.

of $\xi_{s,r}$ than that given by Eqn 14.5:

$$\widetilde{\xi}_{s,r} = \frac{\Delta T_r + C_r}{\Delta T_s + C_r}, \tag{14.6}$$

namely, C_r is used in both the numerator and the denominator, which eliminates large swings in $\xi s, r$ due to discrepancies between C_r and C_s. This is a bit of a "fudge factor," and in retrospect a better solution would have been to use $C_r + C_s$; but, we find it works well in practice. The other fudge factor is that `tcpanaly` allows estimates for $\widetilde{\xi}_{s,r} \geq 0.95$, to accommodate self-clocking effects.

After taking into account these considerations, we then form the PR-based estimate:

$$\widehat{\text{PR}}^{c} = \frac{B}{\Delta T_r + C_r}. \tag{14.7}$$

The ^{c} superscript indicates that the estimator is *conservative*. Since it requires $\Delta T > \text{RTT}_{\text{min}}$, it may be an underestimate if the connection never managed to "fill the pipe," which we illustrate shortly.

For the same group of packets, `tcpanaly` also computes an "optimistic" estimate corresponding to the group minus the final packet (the one that arrived more than RTT_{min} after the first packet):

$$\widehat{\text{PR}}^{o} = \frac{B^-}{\Delta T_r^- + C_r}, \tag{14.8}$$

where B^- is the number of bytes received after subtracting those for the last packet in the group, and ΔT_r^- is likewise the interval over which the group arrived, excluding the final packet. (Thus, we always have $\Delta T_r^- \leq \text{RTT}_{\text{min}}$.) `tcpanaly` does not place any restriction on the expansion factor for the packets used in this estimate, because sometimes the data packets were in fact compressed by the network ($\xi_{s,r}^- < 1$) but still give reliable estimates, because they queued at the bottleneck link behind earlier packets transmitted by the sender. `tcpanaly` does require, however, that either $\Delta T_r^- > \frac{1}{2}\text{RTT}_{\text{min}}$, or that B is equal to the offered window (i.e., the connection was certainly window-limited), to ensure that compression of a small number of packets does not skew the estimate.[6]

We compute the final estimates as the maxima of $\widehat{\text{PR}}^{c}$ and $\widehat{\text{PR}}^{o}$. Note that the algorithms described above work best with cooperation between the sender and the receiver, in order to detect out-of-order arrivals, and to form a good estimate for RTT_{min}, which can be quite difficult to assess from the receiver's vantage point because it cannot reliably infer the sender's congestion window.

Figure 14.8 illustrates the difference between computing $\widehat{\text{PR}}^{c}$ and $\widehat{\text{PR}}^{o}$ for a window-limited connection. RTT_{min} is about 110 msec. 8 packets arrive, starting at $T = 1.5$. The optimistic estimate is based on the 3,584 bytes arriving 22 msec after the first packet, for a rate of about 163 Kbyte/sec. The conservative estimate includes the 9th packet arriving significantly later than the first 8 (due to the window limit). The corresponding estimate is 4,096 bytes arriving in 115 msec, for a rate of about 36 Kbyte/sec. In this case, the optimistic estimate is much more accurate, as the limiting bandwidth is in fact that of a T1 circuit, corresponding to about 170 Kbyte/sec of user data. In this example, the connection is limited by the *offered* window, which is easy to detect. Very often, however, connections are instead limited by the congestion window, due earlier retransmission

[6]The precise method used is a bit more complicated, since it includes the possibility of different-sized packets arriving.

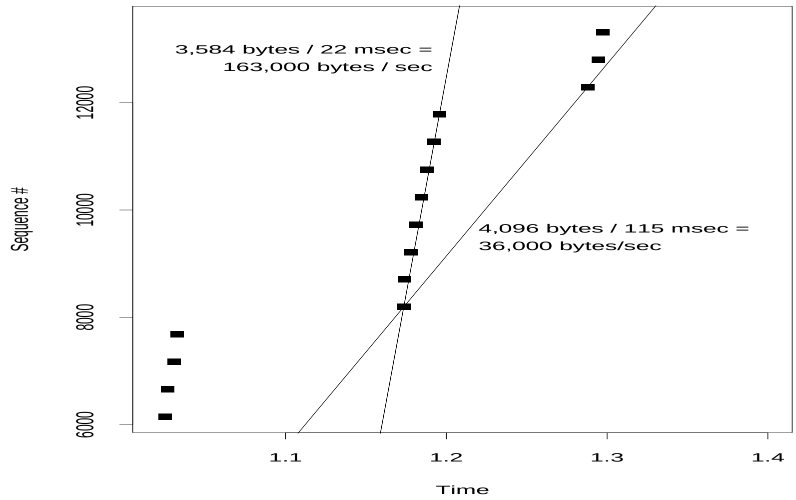

Figure 14.8: Peak-rate optimistic and conservative bottleneck estimates, window-limited connection

events. This limit is more difficult for the receiver to detect. Thus, $\widehat{\mathrm{PR}}^c$ often forms a considerable underestimate.

On the other hand, Figure 14.9 shows an instance in which $\widehat{\mathrm{PR}}^o$ is a large overestimate. The optimistic and conservative estimates for this trace both occurred for the group of packets arriving at time $T = 1.5$, in the middle of the figure. As can be seen from the surrounding groups, the true bottleneck capacity is about 170 Kbyte/sec (T1). The packet group at $T = 1.5$, however, has been *compressed* by the network (cf. § 16.3.2), and it all arrives at *Ethernet* speed. Thus, PR forms a gross overestimate for $\widehat{\mathrm{PR}}^o$. Furthermore, *even if* $\xi^-_{s,r}$ *were checked when forming this estimate*, the estimate would have been accepted, since the packets *left* the sender at Ethernet speed, too! In addition, $\widehat{\mathrm{PR}}^c$ is again a serious underestimate because the connection is again window-limited.

Thus, while PR is fairly simple to compute, it often fails to provide reliable estimates. We need a more robust estimation technique.

14.6 Robust bottleneck estimation

Motivated by the shortcomings of packet pair and PR estimation techniques, we developed a significantly more robust procedure, "packet bunch modes" (PBM). The main observation behind PBM is that dealing with the shortcomings of the other techniques involves both forming a range of estimates based on different packet bunch sizes, and to analyze the result with the possibility in mind of finding more than one bottleneck value.

By considering different bunch sizes, we can accommodate limited receiver clock resolutions (§ 14.4.2) and the possibility of multiple channels or load-balancing across multiple links (§ 14.4.4), while still avoiding the risk of underestimation due to noise diluting larger bunches, or window limitations (§ 14.5), since we also consider small bunch sizes.

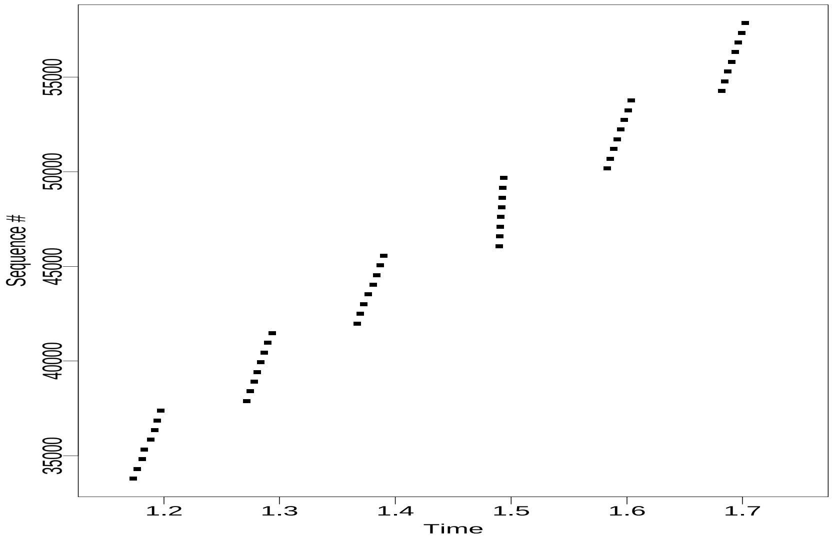

Figure 14.9: Erroneous optimistic estimate due to data packet compression

By allowing for finding multiple bottleneck values, we both again accommodate multi-channel (and multi-link) effects, and also the possibility of a bottleneck *change* (§ 14.4.3). Furthermore, these two effects can be distinguished from one another: multiple bottleneck values due to multi-channel effects *overlap*, while those due to bottleneck changes fall into separate regions in time.

In the remainder of this section we discuss a number of details of PBM. Many are heuristic in nature and evolved out of an iterative process of refining PBM to avoid a number of obvious estimation errors. It is unfortunate that PBM has a large heuristic component, as it makes it more difficult to understand. On the other hand, we were unable to otherwise satisfactorily deal with the considerable problem of noise in the packet arrival times. We hope that the basic ideas underlying PBM—searching for multiple modes and interpreting the ways they overlap in terms of bottleneck changes and multi-channel paths—might be revisited in the future, in an attempt to put them on a more systematic basis.

14.6.1 Forming estimates for each "extent"

PBM works by stepping through an increasing series of packet bunch sizes, and, for each, computing from the receiver trace all of the corresponding bottleneck estimates. We term the bunch size as the *extent* and denote it by k. For each extent, we advance a window over the arrivals at the receiver. The window is nominally k packets in size, but is extended as needed so that it always includes $k \cdot$ MSS bytes of data (so we can include less-than-full packets in our analysis). We do not, however, do this extension for $k = 1$, as that can obscure multi-channel effects.[7]

[7]For higher extents ($k > 1$), this extension does not obscure multi-channel effects, because we detecte multi-channel bottlenecks based on comparing estimates for $k = 1$ with estimates for $k = m$, where m is the number of multiple channels. Thus, the main concern is to not confuse the $k = 1$ estimate.

We also extend the window to include more packets if $\Delta T_r < C_r$, that is, if all the arrivals occurred without the receiver's clock advancing.

If any of the arrivals within the window occurred out of order, or if they were transmitted due to a timeout retransmission, we skip analysis of the group of packets, as the arrival timings will likely not reflect the bottleneck bandwidth.

If when the last packet in the group was sent, the sender had fewer than k packets in flight, then some unusual event occurred during the flight (such as retransmission or receipt of an ICMP source quench), and we likewise skip analysis of the group.

We next compute bounds on ΔT_r, using Eqn 14.3:

$$\begin{aligned}\Delta T_r^- &= \max(\Delta T_r - C_r, 0)\\ \Delta T_r^+ &= \Delta T_r + C_r.\end{aligned}$$

We also compute two *expansion factors* associated with the group, similar to that in Eqn 14.6. The first is more conservative:

$$\xi^c_{s,r} = \frac{\Delta T_r - C_r}{\Delta T_s + C_r}, \tag{14.9}$$

where ΔT_s again is the difference in time between the departure of the last packet and that of the first. The additional conservatism comes from using $\Delta T_r - C_r$ in the numerator. The second is likely to be overall the more accurate, but subject to fluctuations due to limited clock resolution:

$$\xi^o_{s,r} = \frac{\Delta T_r}{\Delta T_s + C_r}.$$

We term it "optimistic" since it yields expansion factors larger than $\xi^c_{s,r}$.

If the last packet group we inspected spanned an interval of $\Delta T'_r$, then we perform a heuristic test. If:

$$\frac{\Delta T_r + C_r}{\Delta T'_r + C_r} > 2, \tag{14.10}$$

then this group was spaced out more than twice as much as the previous group, and we skip the group (after assigning $\Delta T'_r \leftarrow \Delta T_r$), because it is likely to reflect sporadic arrivals. In some cases, this decision will be wrong; in particular, after a compression event such as that shown in Figure 14.9, we will often skip the immediately following packet group. However, this will be the only group we skip after the event, so, unless a trace is riddled with compression, our estimation does not suffer.

We then test whether $\xi^o_{s,r} \geq 0.95$ (where use of 0.95 rather than 1 is again an attempt to accommodate the self-clocking effect, per the discussion of Eqn 14.6). If so, we "accept" the group, meaning we treat it as providing a reliable estimate. (We will further analyze the accepted estimates, as discussed below.) Let B denote the number of bytes in the group (excluding those in the first packet, as also done in § 14.5). With the ith such estimate (corresponding to the ith acceptable group), we associate six quantities:

1. p_i^f, an index identifying the first packet in the group;
2. p_i^l, an index identifying the last packet in the group;
3. $\rho_i = B/\Delta T_r$, the bandwidth estimate;

4. $\rho_i^- = B/\Delta T_r+$, the lower bound on the estimate due to the clock resolution C_r;

5. $\rho_i^+ = B/\Delta T_r-$, the upper bound on the estimate; and

6. ξ_i^c, the conservative expansion factor corresponding to that given by Eqn 14.9.

We will refer to this set of quantities collectively as ψ_i.

One unusual, additional heuristic we use is that, if $\xi_{s,r}^o < 0.2$, i.e., the data packets were grossly compressed, then we *also* accept the estimate given by the corresponding group. (So we reject the estimate if $0.2 \leq \xi_{s,r}^o < 0.95$.) This reasoning behind this heuristic is the same as that accompanying the discussion of Eqn 14.8, namely, that data packets can be highly compressed but still reflect the bottleneck bandwidth due to queueing at the bottleneck behind earlier packets transmitted by the sender. Finally, we note that this heuristic does not generally lead to problems accepting estimates based on compressed data that would otherwise be rejected, because the compression needs to be rampant for PBM to erroneously accept it as a bona fide estimate.

Finally, from a computational perspective, we would like to have an upper bound on the maximum extent k for which we do this analysis. The nominal upper bound we use is $k = 4$. If, however, the bounds on the estimates obtained for $k < 4$ are unsatisfactorily wide due to limited clock resolution, or if we found a new candidate bottleneck for $k = 4$, then we continue increasing k until both the bounds become satisfactory and we have not produced any new bottleneck candidates. These issues are discussed in more detail in the next section.

14.6.2 Searching for bottleneck bandwidth modes

In this section we discuss how we reduce a set of bottleneck bandwidth estimates into a small set of one or more values. Let $\Psi(k)$ be the set of bottleneck estimates formed using the procedure outlined in the previous section, for an extent of k packets. Let n_k denote the number of estimates, and N the total number of packets that arrived at the receiver. If:

$$n_k < \max(\frac{N}{4}, 5),$$

then we reject further analysis of $\Psi(k)$ because it consists of too few estimates. Otherwise, consider $\Psi(k)$ as comprising a sound set of estimates, and turn to the problem of extracting the best estimate from the set.

Previous bottleneck estimation work has focussed on identifying a single best estimate [Bo93, CC96a]. As discussed at the beginning of § 14.6, we must instead accommodate the possibility of forming multiple estimates. This then rules out the use of the most common robust estimator, the median, since it presupposes unimodality. We instead turn to techniques for identifying *modes*, i.e., local maxima in the density function of the distribution of the estimates. Using modal techniques gives PBM the ability to distinguish between a number of situations (bottleneck changes, multi-channel links) that previous techniques cannot.

Clustering the estimates

Because modes are properties of density functions, in trying to identify them we run into the usual problem of estimating density from a finite set of samples drawn from an (essentially) continuous distribution. [PFTV86] gives one procedure for doing so, based on passing a size-k window over sorted samples $X_{(i)}$ to see where $X_{(i+k-1)} - X_{(i)}$ is minimal. $[X_{(i)}, X_{(i+k-1)}]$ then corresponds to the region of highest density, since it packs the most datapoints into the least change in X. We experimented with this algorithm but found the results it produced for our estimation unsatisfactory, because there is no obviously correct choice for k, and different values yield different estimates.

We then devised an algorithm based on a similar principle of conceptually passing a window over the sorted data. Instead of parameterizing the algorithm with a window size k, we use an "error-factor," σ, for $\sigma > 1$. We then proceed through the sorted data, and, for each $X_{(i)}$, we search for an l satisfying $i \leq l < n$ such that:

$$X_{(l)} \leq \sigma X_{(i)} < X_{(l+1)}.$$

In other words, we look ahead to find two estimates that straddle the value of a factor σ larger than $X_{(i)}$. The first estimate, with index (l), is within a factor σ of $X_{(i)}$, while the second, $(l+1)$, is beyond it. If there is no such l (which can only happen if $X_{(n)} \leq \sigma X_{(i)}$), then we consider $X_{(n)}$ as the end of the range of the modal peak.

We term $C_i = l - i + 1$ the *cluster size*, as it gives us the number of points that lie within a factor of σ of $X_{(i)}$. If $C_i \leq 3$, then we consider the cluster *trivial*, and disregard it. Otherwise, we take as the cluster's mode its central observation, i.e., $X_{(i+\frac{C_i}{2})}$. If this is identical to that of a previously observed cluster, we *merge* the two clusters.[8] We then continue advancing the window until we have defined m cluster tuples. The final step is to prune out any clusters that overlap with a larger cluster.

We now turn to how to select σ. We decided to regard as consistent any bottleneck estimates that fall within $\pm 20\%$ of the central bottleneck estimate. We found that using smaller error bars (less than $\pm 20\%$) can lead to PBM finding spurious multiple peaks, while larger ones can wash out true, separate peaks.

Consequently, we will accept as falling within the estimate's bounds

$$X_{(i)} = 0.8 \cdot X_{(i+\frac{C_i}{2})},$$

and

$$X_{(l)} = 1.2 \cdot X_{(i+\frac{C_i}{2})}.$$

However, σ is in terms of the ratio between $X_{(l)}$, the high end of the bottleneck estimate's range, and $X_{(i)}$, the low end. It is easy to show that the above two relationships can hold if $\sigma = 1.5$, so that is the value we choose. Note, though, that we do not define the estimate's bounds in terms of $\pm 20\%$, but as

$$[\min(X_{(i)}, \rho_c^-), \ldots, \max(X_{(l)}, \rho_c^+)], \tag{14.11}$$

where ρ_c^- is the minimum bound on $X_{(i+\frac{C_i}{2})}$ due to clock resolution limits, and ρ_c^+ is the maximum such bound. In the absence of clock resolution limits, the bounds will often be tighter than $\pm 20\%$; but in the presence of such limits, they will often be wider.

The final result is $\Phi(k)$, a list of disjoint, non-trivial clusters associated with $\Psi(k)$, sorted by descending cluster size, and each with associated error bars given by Eqn 14.11.

[8] This can happen because of repeated observations yielding the same bottleneck estimates, due to clock resolution granularities and constant packet sizes.

Reducing the clusters

It is possible that $\Phi(k)$ is empty, because $\Psi(k)$ did not contain any non-trivial clusters. This can happen even if n_k is large, if the individual estimates differ sufficiently. In this case, we consider the extent-k analysis as having failed, and proceed to the next extent, or stop if $k \geq 4$.

Otherwise, we inspect the estimate reflected by each cluster to determine its suitability, as follows. First, we compute $\xi_i^{c(50)}$ and $\xi_i^{c(95)}$ as the 50th and 95th percentiles of the conservative expansion factors ξ_i^c associated with each of the estimates ψ_i within the cluster (per Eqn 14.9).

We next examine all of the estimates that fall within the cluster's error bars (nominally, $\pm 20\%$), to determine the cluster's *range*: where in the trace we first and last encountered packets leading to estimates consistent with the cluster. When determining the cluster's range, we only consider estimates for which $\xi_i^c \geq \min(\xi_i^{c(50)}, 1.1)$, to ensure that we base the cluster's range on sound estimates (those derived from definite expansion, if present very often; otherwise, those in the upper 50% of the expansions). Without this filtering, a cluster's range can be artificially inflated due to self-clocking and spurious noise, which in turn can mask a bottleneck change.

We next inspect all of the extent-k estimates derived from packets falling within ψ_i's inner range, to determine η_i, the proportion of these estimates consistent with the cluster (within the error bars given by Eqn 14.11). η_i is the cluster's *local proportion*, and reflects how well it captures the behavior within its associated range. A value of η_i near 1 indicates that, over its range, the evidence was very consistent for the given bottleneck estimate, while a lower value indicates the evidence for the bottleneck was diluted by the presence of numerous inconsistent measurements. If $\eta_i < 0.2$, or if $k = 2$ (i.e., we are looking at packet pair estimates) and $\eta_i < 0.3$, we reject the estimate reflected by the cluster as too feeble. This heuristic prunes out the vast majority of estimates that have made it this far in the process, since most of them are due to spurious noise effects. It keeps, however, those that appear to dominate the region over which we found them.

It at first appears that a threshold of 0.2 or 0.3 is considerably too lenient, but in fact it works well in practice, and using a higher threshold runs the risk of failing to detect multi-channel effects, which can split the estimates into two or three different regions. For example, in Figure 14.7 we can readily see that a number of different slopes emerge.

An estimate that has made it this far is promising. The next step is to see whether we have already made essentially the same estimate. We do so by inspecting the previously accepted ("solid") estimates to see whether the new estimate overlaps. If so, we consolidate the two estimates. The details of the consolidation are numerous and tedious.[9] We will not develop them here, except to note that this is the point where a solid estimate with a large error interval ($\rho_i^- \ll \rho_i^+$) can tighten its error interval based on the observation that we have independent evidence for the same estimate at a different extent, and the new evidence has a smaller associated error (due to the higher extent). This is also the point where we determine whether to increase the *maximum extent* associated with an estimate. Doing so is important when hunting for multi-channel bottleneck links, as these should exhibit one bandwidth estimate with a maximum extent exactly equal to the number of parallel channels.

If we do not consolidate a new estimate with any previous solid ones, then we add it to the set of solid estimates.

[9] And can be gleaned from the `tcpanaly` source code.

Forming the final estimates

After executing the process outlined in the previous two subsections, we have produced Υ, a set of "solid" estimates. It then remains to further analyze Υ to determine whether the estimates indicate the presence of a multi-channel link or a bottleneck change. Note that in the process we may additionally merge some of the estimates; we have not yet constructed the set of "final" estimates!

If Υ is empty, then we failed to produce any solid bandwidth estimates. This is rare but occasionally happens, for one of the following reasons:

1. so many packet losses that too few groups arrived at the receiver to form a reliable estimate;

2. so many retransmission events that the connection never opened its congestion window sufficiently to produce a viable stream of packet pairs;

3. such a small receiver window that the connection could never produce a viable stream of packet pairs; or,

4. the trace of the connection was so truncated that it did not include enough packet arrivals (§ 10.3.4).

In $\mathcal{N}_1$, we encountered 37 failures; in $\mathcal{N}_2$, only 1, presumably because the bigger windows used in $\mathcal{N}_2$ (§ 9.3) gave more opportunity of observing a packet group spaced out by the bottleneck link. Interestingly, no estimation failed on account of too many out-of-order packet deliveries. Even those with 25% of the arrivals occurring out of order provided enough in-order arrivals to form a bottleneck estimate.

Assuming Υ is not empty, then if it includes more than one solid estimate, we compare the different estimates as follows. First, we define the *base estimate*, ρ^*, as the first solid estimate we produced. No other estimate was formed using a smaller extent than ρ^*, since we generated estimates in order of increasing extent.

If ρ^* was formed using an extent of $k = 2$, and if Υ includes additional estimates that were only observed for $k = 2$ (i.e., for higher extents we never found a compatible estimate with which to consolidate them), then we assess whether these estimates are "weak." An estimate is weak if it is low compared to ρ^*; the overall proportion of the trace in accordance with the estimate is small; and the estimate's expansions $\xi_i^{c(50)}$ and $\xi_i^{c(95)}$ are low. If these all hold, then the estimate fits the profile of a spurious bandwidth peak (due, for example, to the relatively slow pace at which duplicate acks clock out new packets during "fast recovery", per § 9.2.7), and we discard the estimate.

We now can (at last!) proceed to producing a set of final bandwidth estimates. We begin with the base estimate, ρ^*. We next inspect the other surviving estimates as follows. For each estimate, we test to see whether its range overlaps any of the final estimates. If so, then we check whether the two estimates might reflect a two-channel bottleneck link, which requires:

1. One of the estimates must have a maximum extent of $k = 2$ and the other must have a minimum extent of $k \geq 3$. Call these E_2 and E_3. This requirement splits the estimates into one that reflects the downstream bottleneck, which is only observed for packet pairs ($k = 2$, since for $k > 2$ the effect cannot be observed for a two-channel bottleneck), and the other that reflects the true link bandwidth (which can only be observed for $k > 2$, since $k = 2$ is obscured by the multi-channel effect).

2. E_3 must span at least as much of the trace as E_2. It may span more due to phase effects, as illustrated in Figure 14.7.

3. Unless E_3 spans almost the entire trace, we require that:

$$\xi_3^{c(95)} \geq \min(\frac{3}{4}\xi_2^{c(95)}, 2).$$

This requirement assures that E_3 was at least occasionally observed for a considerable expansion factor, or, if not, then neither was E_2. The goal here is to not be fooled by an E_3 that was only generated by self-clocking (i.e., no opportunity to observe a higher bandwidth for an extent $k > 2$).

4. The bandwidth estimate corresponding to E_3 must be at least a factor of 1.5 different than that from E_2, to avoid confusing a single very broad peak with two distinct peaks.

If the two estimates meet these requirements, then we classify the trace as exhibiting a multi-channel bottleneck link.

We originally performed the same analysis for (E_3, E_4), that is, for overlapping estimates, one with extent $k = 3$ and one with $k \geq 4$. A three-channel bottleneck would produce estimates for both. We did not find any traces that plausibly exhibited three-channel bottleneck links, though, and did endure a number of false findings, so we omit three-channel analysis from PBM. If we have the opportunity in the future to obtain traces from paths with known three-channel bottlenecks, then we presume we could devise a refinement to the present methodology that would reliably detect their presence.

If two estimates overlap but fail the above test for a multi-channel bottleneck, and if either has both a higher bandwidth estimate and accords with twice as many measurements as the other, then we discard the weaker estimate and use the stronger in its place.

If they overlap but neither dominates, then if one has a minimum extent larger than the other's maximum extent, and larger than $k = 3$ (to avoid erroneously discarding multi-channel estimates), then we discard it as almost certainly reflecting spurious measurements.

If two estimates overlap and none of the three procedures above resolve the conflict, then PBM reports that it has found conflicting estimates. This never happened when analyzing $\mathcal{N}_1$. For $\mathcal{N}_2$, we found only 10 instances. 7 involve `lbli`, which frequently exhibits both a bottleneck change and a multi-channel bottleneck, per Figures 14.4 and 14.5. The other three all exhibit a great deal of delay variation, leading to the conflicting estimates.

If the newly considered estimate does not overlap, then, after some final sanity checks to screen out spurious measurements (which can otherwise remain undetected, if they happen to occur at the very beginning or end of the trace, and thus do not overlap with the main estimate), we add it to the collection of final estimates. At this point, we conclude that the trace exhibits a bottleneck change.

Completing the above steps results in one or more final estimates. For each final estimate ρ_B, we then associate bounds:

$$\rho_B^- < \rho_B < \rho_B^+, \tag{14.12}$$

where ρ_B^- and ρ_B^+ reflect Eqn 14.11, i.e., the smallest and largest estimates within $\pm 20\%$ of ρ_B, or the bounds on ρ_B itself due to limited clock resolution (§ 14.4.2), if larger. In the latter case, we term the estimate as *clock-limited*.

Results of estimation	$\mathcal{N}_1$ #	$\mathcal{N}_1$ %	$\mathcal{N}_2$ #	$\mathcal{N}_2$ %
Single bottleneck	2,018	90%	14,483	94%
Estimate failure	37	1.7%	1	—
Broken estimate	46	2.1%	72	0.05%
Ambiguous estimate:	139	6.2%	779	5.1%
change	94	4.2%	594	3.9%
multi-channel	74	3.3%	506	3.3%
conflicting	0	0.0%	11	0.07%
Total trace pairs	2,240	100%	15,335	100%

Table XVIII: Types of results of bottleneck estimation for $\mathcal{N}_1$ and $\mathcal{N}_2$

Results of estimation	$\mathcal{N}_1$ #	$\mathcal{N}_1$ %	$\mathcal{N}_2$ #	$\mathcal{N}_2$ %
Single bottleneck	1,929	95%	14,134	98%
Estimate failure	37	1.8%	1	—
Broken estimate	19	0.9%	61	0.04%
Ambiguous estimate:	48	2.3%	204	1.4%
change	7	0.34%	67	0.47%
multi-channel	41	2.0%	135	0.9%
conflicting	0	0.0%	3	0.02%
Total trace pairs	2,033	100%	14,400	100%

Table XIX: Types of results after eliminating trace pairs with `lbli`

14.7 Analysis of bottleneck bandwidths in the Internet

We applied the bottleneck estimation algorithms developed in § 14.5 and § 14.6 to the trace pairs in $\mathcal{N}_1$ and $\mathcal{N}_2$ for which the clock analysis described in Chapter 12 did not uncover any uncorrectable problems. These comprised a total of 2,240 and 15,335 trace pairs, respectively. Table XVIII summarizes the types of results we obtained. "Single bottleneck" refers to traces for which we found solid evidence for a single, well-defined bottleneck bandwidth. An "estimate failure" occurs when PBM is unable to find any persuasive estimate peaks (§ 14.6.2). "Broken estimate" summarizes traces for which PBM yielded a single uncontested estimate, but subsequent queueing analysis found counter-evidence indicating the estimate was inaccurate. (We describe this self-consistency test in § 16.2.6.) "Ambiguous estimate" means that the trace pair did not exhibit a single, well-defined bottleneck: it included either evidence of a bottleneck change, or a multi-channel bottleneck link, or both; or it had conflicting estimates, already discussed in § 14.6.2.

The ambiguous estimates were clearly dominated by `lbli`, no doubt because its ISDN link routinely exhibited both bottleneck changes and multi-channel effects (since when it activates the second ISDN channel, the bandwidth doubles and a parallel path arises). Table XIX summarizes the types of results after removing all trace pairs with `lbli` as sender or receiver. We see that PBM almost always finds a single bottleneck. The results also exhibit a general trend between $\mathcal{N}_1$ and $\mathcal{N}_2$ towards fewer problematic estimates. We suspect the difference is due to two effects: the lower prevalence of out-of-order delivery in $\mathcal{N}_2$ compared to $\mathcal{N}_1$, and the use of bigger windows in $\mathcal{N}_2$ (§ 9.3), which provides more opportunity for generating tightly-spaced packet pairs and packet bunches.

In the remainder of this section, we analyze each of the different types of estimated bottlenecks.

14.7.1 Single bottlenecks

Far and away the most common result of applying PBM to our traces was that we obtained a single estimated bottleneck bandwidth. Unlike [CC96a], we do not *a priori* know the bottleneck bandwidths for many of the paths in our study. We thus must fall back on self-consistency checks in order to gauge the accuracy of PBM. Figures 14.10 and 14.11 show histograms of the estimates formed for $\mathcal{N}_1$ and $\mathcal{N}_2$, where the histogram binning is done using the logarithms of the estimates, so the ratio of the sizes of adjacent bins remains constant through the plot.

There are a number of readily apparent peaks. In $\mathcal{N}_1$, we find the strongest at about 170 Kbyte/sec, and another strong one at 6.5 Kbyte/sec. Secondary peaks occur at about 100, 330, 80, and 50 Kbyte/sec, with lesser peaks at 30 Kbyte/sec, 500 Kbyte/sec, and at a bit over 1 Mbyte/sec. The pattern in $\mathcal{N}_2$ is a bit different. The 170 Kbyte/sec peak clearly dominates, and the 6.5 Kbyte/sec peak has shifted over to about 7.5 Kbyte/sec. The peaks between 50 and 100 Kbyte/sec are no longer much apparent, and the 330 Kbyte/sec peak has diminished while the 30, 500 and 1 Mbyte/sec peaks have grown. Finally, a new, somewhat broad peak has emerged at 13–14 Kbyte/sec.

We calibrate these peaks using a combination of external knowledge about popular link speeds, and by inspecting which sites tend to predominate for a given peak. Several common slower link speeds are 56, 64, 128, and 256 Kbit/sec. Common faster links are 1.544 Mbit/sec ("T1"—primarily used in North America), 2.048 Mbit/sec ("E1"—used outside North America), and 10 Mbit/sec (Ethernet). Certainly faster links are in use in the Internet, but we omit discussion of them since none of the bottlenecks in our study exceeded 10 Mbit/sec; we note, however, that it is the use of faster wide-area links that enables a local-area limit such as Ethernet to wind up as a connection's bottleneck.

The link speeds discussed above reflect the *raw* capacity of the links. Not all of this capacity is available to carry user data. Often a portion of the capacity is permanently set aside for framing and signaling. Furthermore, transmitting a packet of user data using TCP requires encapsulating the data in link-layer, IP, and TCP headers. The size of the link-layer header varies with the link technology. The IP and TCP headers nominally require at least 40 bytes, more if IP or TCP options are used. Use of IP options for TCP connections is rare, and none of the connections in our study did so. TCP options are common, especially in the initial SYN packets. Thus, we might take 40 bytes as a solid lower bound on the TCP/IP header overhead. An exception, however, is links utilizing *header compression* (§ 13.3), which, depending on the homogeneity of the traffic traversing the link, can greatly reduce the bytes required to transmit the headers. Header compression works by leveraging off of the large degree of redundancy between the headers of a connection's successive packets. For example, under optimal conditions, CSLIP compression can reduce the 40 bytes to

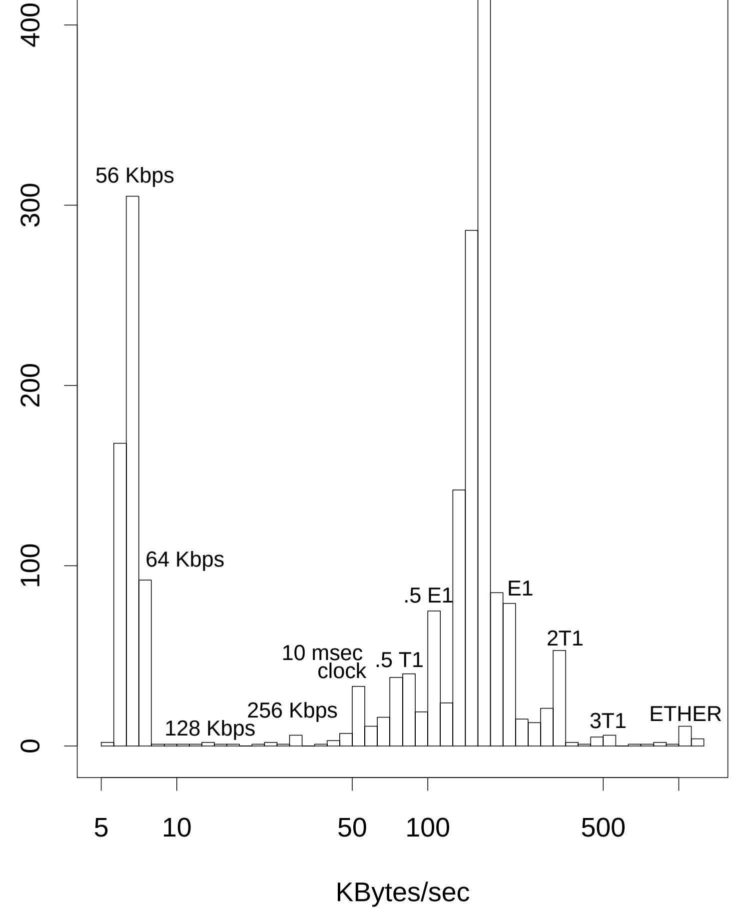

Figure 14.10: Histogram of different single-bottleneck estimates for $\mathcal{N}_1$

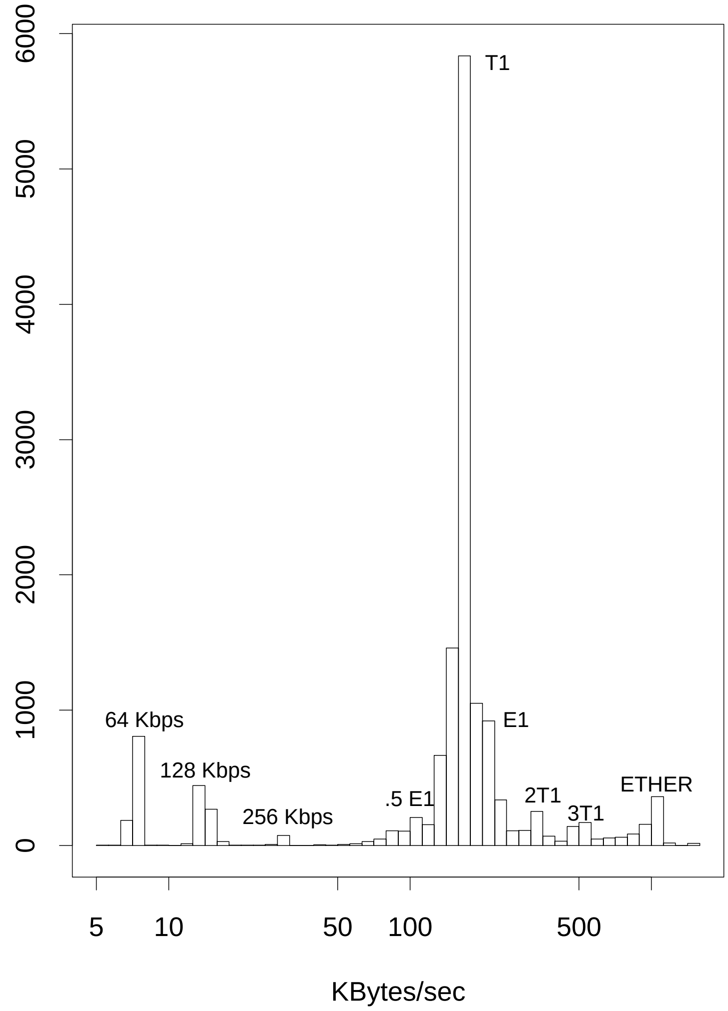

Figure 14.11: Histogram of different single-bottleneck estimates for $\mathcal{N}_2$

Raw rate (ρ_R)	User data rate (ρ_U)	Notes
56 Kbit/sec	$\approx$ 6.2 Kbyte/sec	
64 Kbit/sec	$\approx$ 7.1 Kbyte/sec	
128 Kbit/sec	$\approx$ 14.2 Kbyte/sec	
256 Kbit/sec	$\approx$ 28.4 Kbyte/sec	
1.544 Mbit/sec	$\approx$ 171 Kbyte/sec	T1
2.048 Mbit/sec	$\approx$ 227 Kbyte/sec	E1
10 Mbit/sec	$\approx$ 1.1 Mbyte/sec	Ethernet

Table XX: Raw and user-data rates of different common links

5 bytes. Finally, some links use *data compression* techniques to reduce the number of bytes required to transmit the user data. We presume these techniques did not affect the connections in our study because NPD sends a pseudo-random sequence of bytes (to avoid just this effect).

Given these sundry considerations, we do not hope to nail down a single figure for each link technology reflecting the user data rate it delivers. Instead, we make "ballpark" estimates, as follows. For high-speed links, the framing and signaling overhead consumes about 4.5% of the raw bandwidth [Ta96]. We compromise on the issues of header compression versus additional bytes required for link-layer headers and TCP options by assuming 40 bytes of overhead for each TCP/IP packet. Finally, we assume that a "typical" data packet carries 512 bytes of user data. This is the most commonly observed value in our traces, though certainly not the only one. Given these assumptions, the user data rate available from a link with a raw rate of ρ_R is:

$$\begin{aligned} \rho_U &\approx (.955)(\frac{512}{512+40})\rho_R \\ &\approx .886\rho_R. \end{aligned}$$

Table XX summarizes the corresponding estimated user-data rates for the common raw link rates discussed above. From the table, it is clear that the strong 170 Kbyte/sec peak in Figure 14.10 and Figure 14.11 reflect T1 bottlenecks. Likewise, the 6.5 Kbyte/sec peak reflects 56 Kbit/sec links, and may be slightly higher than the estimate in the Table due to the likely use of header compression. Its shift to 7.5 Kbyte/sec reflects upgrading of 56 Kbit/sec links to 64 Kbit/sec. The 13–14 Kbyte/sec peak reflects 128 Kbit/sec links and the 30 Kbyte/sec peak, 256 Kbit/sec. The 1 Mbyte/sec peaks are clearly due to Ethernet bottlenecks.

These identifications still leave us with some unexplained peaks from the bottleneck estimates. We speculate that the 330 Kbyte/sec peak reflects use of two T1 circuits in parallel, 500 Kbyte/sec reflects three T1 circuits (not half an Ethernet, since there is no easy way to subdivide an Ethernet's bandwidth), and 80 Kbyte/sec comes from use of half of a T1.

We then have only two unexplained peaks remaining: 50 and 100 Kbyte/sec. The 50 Kbyte/sec peak is only prominent in $\mathcal{N}_1$. We find that this peak in fact reflects vagueness due to limited clock resolution: in § 14.4.2 we showed that, for packet pair, the fastest bandwidth a 10 msec clock can yield for 512 byte packets is 51.2 Kbyte/sec. Thus, the 50 Kbyte/sec peak is a measurement artifact, though it also indicates the presence of connections for which PBM was unable to tighten its bottleneck estimate using higher extents (which would reduce uncertainties due to clock resolution), presumably because the connection rarely had more than two packets delivered to the receiver at the bottleneck rate, due to extensive queueing noise.

The 100 Kbyte/sec peak, on the other hand, most likely is due to splitting a single E1 circuit in half. Indeed, we find non-North American sites predominating these connections, as well exhibiting peaks at 200–220 Kbyte/sec, above the T1 rate and just a bit below E1. This peak is, however, absent from North American connections. (See also Figure 14.12 and accompanying discussion, below.)

In summary, we believe we can offer plausible explanations for all of the peaks. Passing this self-consistency test in turn argues that PBM is indeed detecting true bottleneck bandwidths. We next turn to variation in bottleneck rates. We would expect to observe strong site-specific variations in bottleneck rates, since some of the limits arise directly from the speed of the link connecting the site to the rest of the Internet.

Figure 14.12 clearly shows this effect. The figure shows a "box plot" for $\log_{10}$ of the bottleneck estimates for each of the $\mathcal{N}_2$ receiving sites. In these plots, we draw a box spanning the inner two quartiles (that is, from 25% to 75%). A dot shows the median and the "whiskers" extend out to 1.5 times the inter-quartile range. The plot shows any values beyond the whiskers as individual points. The horizontal line marks 171 Kbyte/sec, the popular T1 user data rate (Table XX).

The plot clearly shows considerable site-to-site variation. While all sites reflect some 64 and 128 Kbit/sec bottlenecks, we quickly see that `austr2` has virtually only 128 Kbit/sec bottlenecks, indicating it almost certainly uses a link with that rate for its Internet connection. (`austr`, on the other hand, has at least E1 connectivity.) `lbli` generally does not have a single bottleneck above 64 Kbit/sec (it often has a bottleneck *change* that includes 128 Kbit/sec, but in this section we only consider traces exhibiting a single, unchanged bottleneck). The `lbli` estimates tend to be quite sharply defined. Of those larger than 7 Kbyte/sec, 96% lay within a 30 byte/sec range centered about 7,791 byte/sec. The other site with a narrow bottleneck bandwidth region is `oce`, which has a 64 Kbit/sec link to the Internet, as clearly evidenced by the plot, except for a cluster of outliers at 17 Kbyte/sec. All of the outliers were localized to a 1 day period, perhaps a time when `oce` momentarily enjoyed faster connectivity.

In the main, the plot exhibits a large number of sites with median bottlenecks at T1 rate. A few have slightly higher median bottlenecks, and these tend to be non-North American sites, consistent with E1 links. Two sites have occasional values just below $\log_{10} = 1.5$, corresponding to 256 Kbit/sec links. These sites are `ucl` and `ukc`, both located in Britain, so we suspect these bottlenecks reflect a British circuit or set of circuits. Some sites also exhibit a fair number of bottlenecks exceeding 1 Mbyte/sec: `bnl`, `lbl`, `mid`, `near`, `panix`, and `wustl` (as well as, more rarely, a number of others), indicating these all enjoyed Ethernet-limited Internet connectivity.

We next investigate the stability of bottleneck bandwidth over time. We confine this investigation to $\mathcal{N}_2$, since it includes many more connections between the same sender/receiver pairs, spaced over a large range of time. We begin by constructing for each sender/receiver pair two sequences, $\Delta\mathcal{T}_{s,r}$ and $\mathcal{R}_{s,r}$, giving the difference in time between the beginning of successive connections from the sender to the receiver, and the ratio of the estimated bottleneck rate for the second of the connections to that of the first.

As noted in § 9.3, we varied the mean time between successive connections between sender/receiver pairs, and, in addition, our methodology would sometimes include "revisiting" a pair at a later date. Accordingly, $\Delta\mathcal{T}_{s,r}$ exhibits considerable range: its median is 8 minutes, its 90th

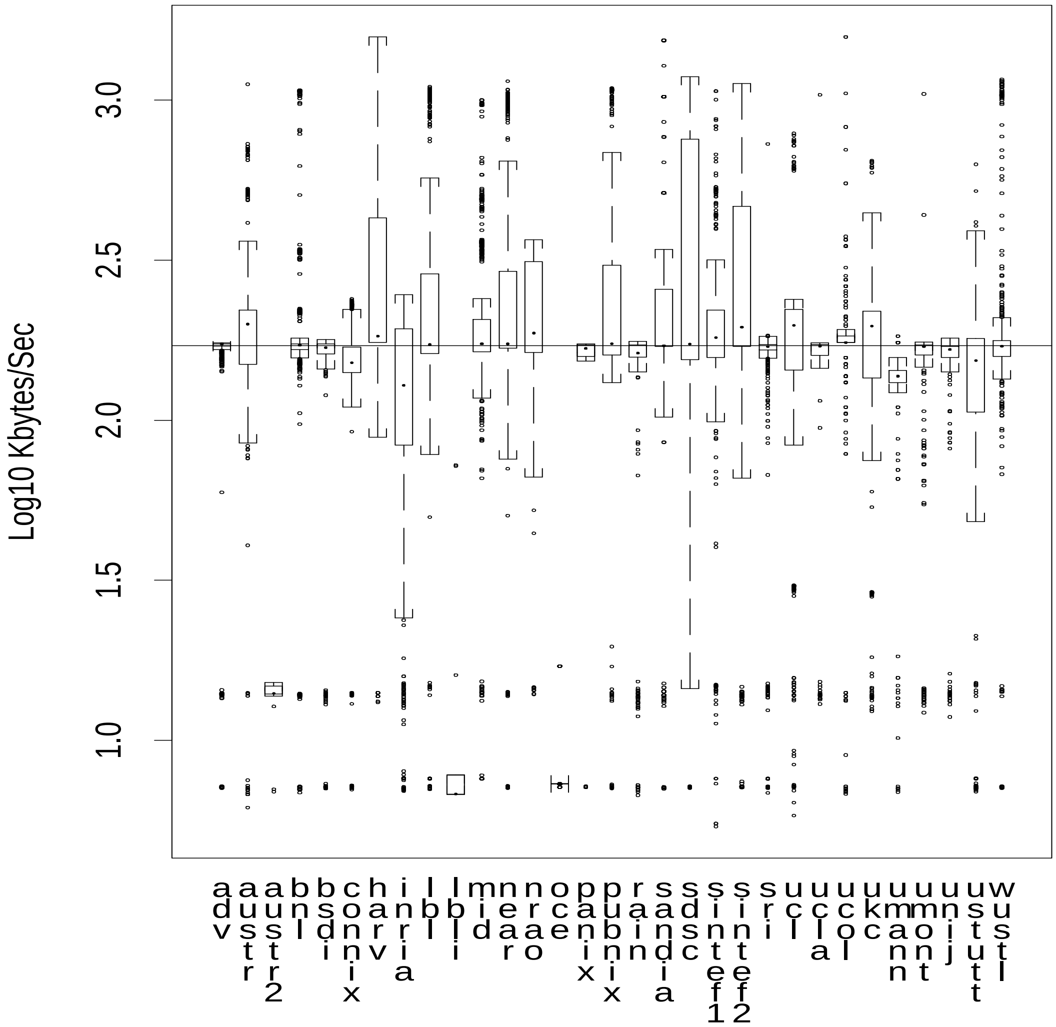

Figure 14.12: Box plots of bottlenecks for different $\mathcal{N}_2$ receiving sites

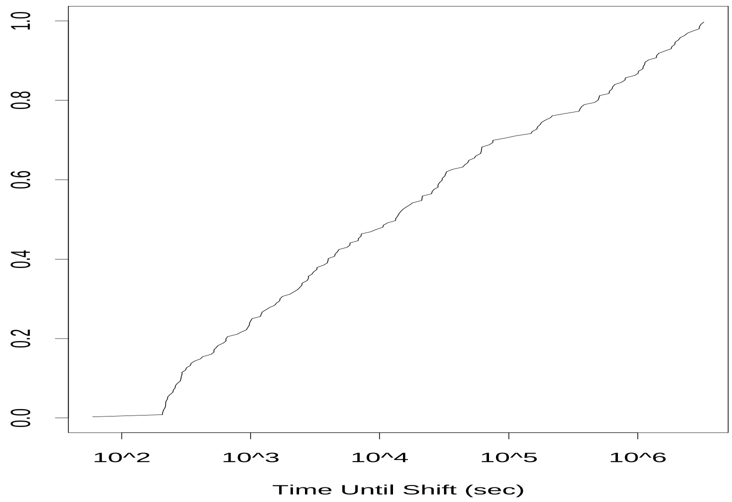

Figure 14.13: Time until a 20% shift in bottleneck bandwidth, if ever observed

percentile is 104 minutes, but its mean is about 7 hours, due to revisiting.

The bottleneck ratio $\mathcal{R}_{s,r}$ overall shows little variation. Its median is exactly 1.0. Evaluating $\mathcal{R}_{s,r}$'s distribution directly can be misleading, because it will tend to be < 1 as often as > 1, depending on whether the second of a pair of estimates was lower or higher than the first. What is more relevant is the "magnitude" of the ratio between successive estimates, which we define as:

$$|\mathcal{R}|_{s,r} \equiv \exp[|\log \mathcal{R}_{s,r}|],$$

that is, the ratio of the larger of the two estimates to the smaller. The median of $|\mathcal{R}|_{s,r}$ is 1.0175, indicating that 50% of the successive estimates differ by less than 1.75% from the previous estimate. We find that 80% of the successive estimates differ by less than 10%, and 98% differ by less than a factor of two.

We consider two different assessments of the stability of the bottleneck rate over time. First, we examine the correlation between $|\mathcal{R}|_{s,r}$ and $\Delta\mathcal{T}_{s,r}$. If bottlenecks fluctuate significantly over time, then we would expect the magnitude of the ratio to correlate with the time separating the connections. If fluctuations are mainly due to measurement imprecision, then the two should be uncorrelated.

For $\Delta\mathcal{T}_{s,r} < 1$ hour (85% of the successive measurements), we find very slight positive correlation between $|\mathcal{R}|_{s,r}$ and $\Delta\mathcal{T}_{s,r}$, with a coefficient of correlation equal to 0.03. We obtain a coefficient of about this size regardless of whether we first apply logarithmic transformations to either or both of $|\mathcal{R}|_{s,r}$ and $\Delta\mathcal{T}_{s,r}$ in an attempt to curb the influence of outliers. For $\Delta\mathcal{T}_{s,r} \geq 1$ hour, the coefficient of correlation rises to about 0.09. This is still not strong positive correlation, and indicates that bottleneck bandwidth is quite stable over periods of time ranging from minutes to days (the mean of $\Delta\mathcal{T}_{s,r}$, conditioned on it exceeding 1 hour, is 52 hours).

We can also assess stability in terms of the time required to observe a significant change. To do so, for each sender/receiver pair we take the first bottleneck estimate as a "base measurement" and then look to see when we find two consecutive later estimates that both differ from the base measurement by more than 20%, and that both agree in terms of the direction of the change (20% larger or smaller). We look for consecutive estimates to weed out spurious changes due to isolated measurement errors. We find that only about a fifth of the sender/receiver pairs *ever* exhibited a shift of this magnitude. Furthermore, the amount of time between the first measurement and the first of the pair constituting the shift has a striking distribution, shown in Figure 14.13. The distribution appears almost uniform, except that the x-axis is logarithmically scaled, indicating that shifts in bottleneck bandwidth occur over a wide range of time scales. This finding qualitatively matches that in Chapter 7 that the time over which different routes persist varies over a wide range of scales. We would expect general agreement since one obvious mechanism for a shift in bottleneck bandwidth is a routing change, though some routing changes will not alter the bottleneck.

The last property of bottleneck bandwidth we study in this section is its symmetry: how often is the bottleneck from host A to host B the same as that from B to A? We know from Chapter 8 that Internet routes often exhibit major routing asymmetries, with the route from A to B differing from the reverse of B to A by at least one city about 50% of the time in $\mathcal{N}_2$. It is quite possible that these asymmetries will also lead to bottleneck asymmetries, an important consideration because sender-based "echo" bottleneck measurement techniques such as those explored in [Bo93, CC96a] will observe the *minimum* bottleneck of the two directions.

Figure 14.14 shows a scatter plot of the *median* bottleneck rate estimated in the two directions for the hosts in our study. The plot uses logarithmic scaling on both axes to accommodate the wide range of bottleneck rates. For each pair of hosts A and B for which we had successful measurements in both directions, we plot a point corresponding to A's median estimate on the x-axis, and B's median estimate on the y-axis. The solid diagonal line has slope one and offset zero. Points falling on it have equal estimates in the two directions. The dashed diagonal lines mark the extent of estimates 20% above or below the solid line. About 45% of the points fall within $\pm 5\%$ of equality, and 80% within $\pm 20\%$ (i.e., within the dashed lines). But about 20% of the estimates differ by considerably more. For example, some paths are T1 limited in one direction but Ethernet limited in the other, a major difference.

Of the considerably different estimates, the median ratio between the two estimates is 40% and the mean is 65%. In light of these variations, we see that sender-based bottleneck measurement provides a good rough estimate, but will sometimes yield quite inaccurate results.

14.7.2 Bottleneck changes

We now turn to analyzing how frequently the bottleneck bandwidth changes during a single TCP connection. From the results in the previous section, we expect such changes to occur only rarely, and indeed this is the case. If we disregard `lbli`, which, as noted in § 14.4.3, frequently exhibits a bottleneck change due to the activation of its second ISDN channel, then, as shown in Table XIX, only about 1 connection in 250 (0.4%) exhibited a bottleneck change. The changes are all large, by definition (since we merge bottleneck estimates with minor differences), with the median ratio between the two bottlenecks in the range 3-6.

Figure 14.15 illustrates one of the smaller changes. At about $T = 2.3$, the bottleneck decreases from an estimated 168 Kbyte/sec to an estimated 99 Kbyte/sec. The effect here is not self-clocking, as the one-way delays of the packets show a considerable increase at $T = 2.3$ as well. Contrast this behavior with that at about $T = 2.1$, where we see a momentary decrease. In this

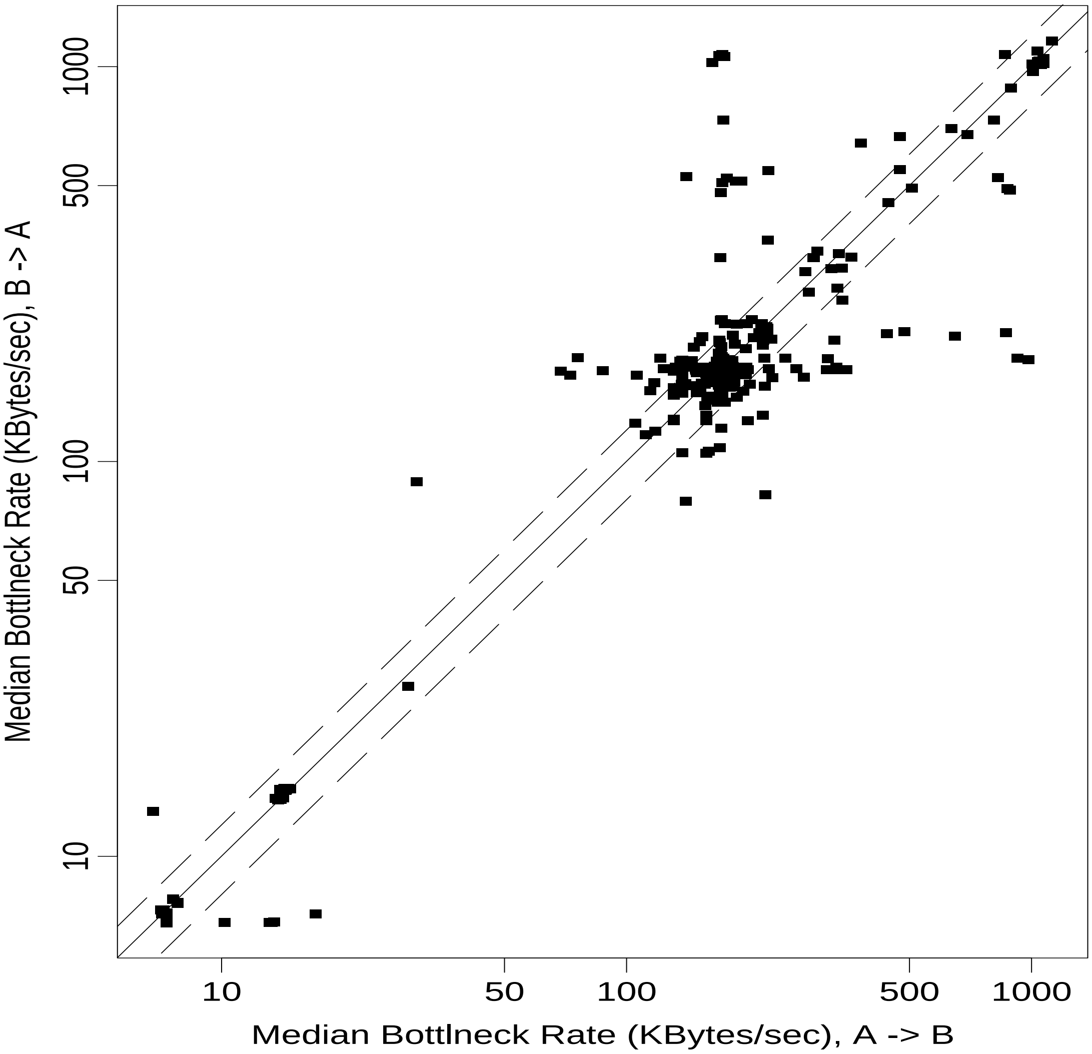

Figure 14.14: Symmetry of median bottleneck rate

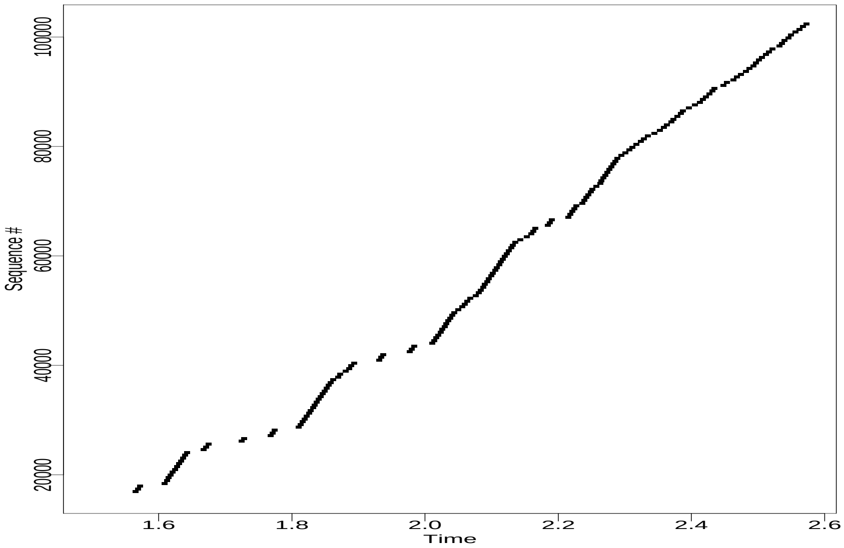

Figure 14.15: Sequence plot reflecting halving of bottleneck rate

case, the slow-down is not accompanied by an increase in transit time, and is instead a self-clocking "echo" of the slow-down at $T = 1.9$.

Since 99 Kbyte/sec is not a particularly compelling link rate vis-a-vis Table XX, we might consider that the bottleneck rate did not in fact change, but instead at $T = 2.3$ a *constant-rate* source of competing traffic began arriving at the bottleneck link, diluting the bandwidth available to our connection and hence widening the spacing between arriving data packets. This may well be the case. We note, however, that *effectively* this situation is the same as a change in the bottleneck rate: if the additional traffic is indeed constant rate, and not adaptive to the presence of our traffic, then we might as well have suffered a reduction in the basic bottleneck link rate, since that is exactly the effect our connection will experience. So we argue that, in this case, we *want* to regard the change as due to a bottleneck shift, rather than due to congestion.

A few of the bottleneck "changes" appear spurious, however. These apparently stem from connections with sufficient delay noise to completely wash out the true bottleneck spacing, and which coincidentally produce a common set of packet spacings that lead to a false bottleneck peak. Most changes, however, appear genuine. In both datasets, about 15% of the changes differ by about a factor of two, suggesting that a link had been split or two sub-links merged following a failure or repair at the physical layer.

14.7.3 Multi-channel bottlenecks

The final type of bottleneck we analyze are those exhibiting the *multi-channel* effect discussed in § 14.4.4. As shown in Table XIX, except for connections involving `lbli`, known to have a 2-channel bottleneck link, we found few multi-channel bottlenecks. However, after excluding `lbli`, we still found a tendency for a few sites to predominate among those exhibiting multi-channel bottlenecks: `inria`, `ukc`, and `ustutt`, in both datasets, and `wustl` in $\mathcal{N}_1$. The presence of this last

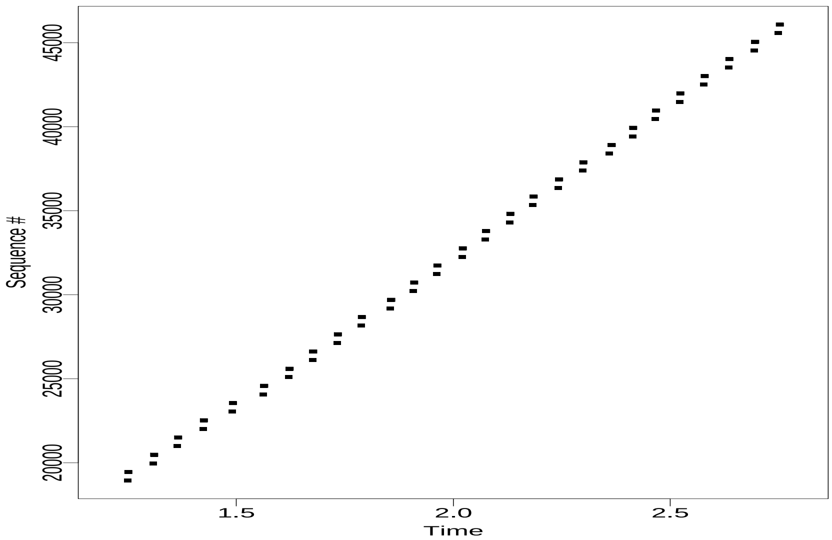

Figure 14.16: Excerpt from a trace exhibiting a false "multi-channel" bottleneck

site in the list is not surprising, since we know that due to route "flutter" many of its connections used two very different paths to each remote site (§ 6.6).

However, we cannot confidently claim that any of the non-`lbli` purported multi-channel bottlenecks are in fact due to multi-channel links, since we find that very often the trace in question is plagued with delay noise, and lacks the compelling pattern shown in Figure 14.6. The ratios between the nominal bandwidths of extent $k = 2$ and $k \geq 3$ bunches also generally tend to be < 2, which from our experience often instead indicates excessive measurement noise smearing out the bottleneck signature.

Even when the measurements appear quite clean, we must exercise caution. Figure 14.16 shows a portion of an $\mathcal{N}_1$ trace from `ukc` to `ucl` with a pattern very similar to that in Figure 14.6. Most of the trace looks exactly like the pattern shown. PBM analyzes this trace as exhibiting a multi-channel bottleneck with an upper rate of 477 Kbyte/sec and a slower rate of 18 Kbyte/sec. However, detailed analysis of the trace reveals a few packet bunches with $k \geq 3$ that arrived spaced at 477 Kbyte/sec, evidence that either the bunches were *compressed* (§ 16.3.2) subsequent to the multi-channel bottleneck, or the bottleneck is in fact not multi-channel. Further analysis reveals that the sending TCP was limited by a sender-window (§ 11.3.2), and that the ack-every-other policy used by the receiver led to almost perfect self-clocking of flights of two packets arriving at the true bottleneck rate, followed by a self-clocking lull, followed by another flight of two, and so on. While PBM includes heuristics based on $\widetilde{\xi}_{s,r}$ (Eqn 14.6) that attempt to discard traces like these as multi-channel candidates, this one passed the heuristic due to some unfortuitous packet bunch expansion early in the trace. Had the sending TCP not been window-limited, it would have continued expanding the window as the self-clocking set in, leading to numerous flights of $k \geq 3$ packets all arriving at the faster link rate, and PBM would have determined that in fact the link was not multi-channel.

In summary, we are not able to make quantitative statements about multi-channel bottle-

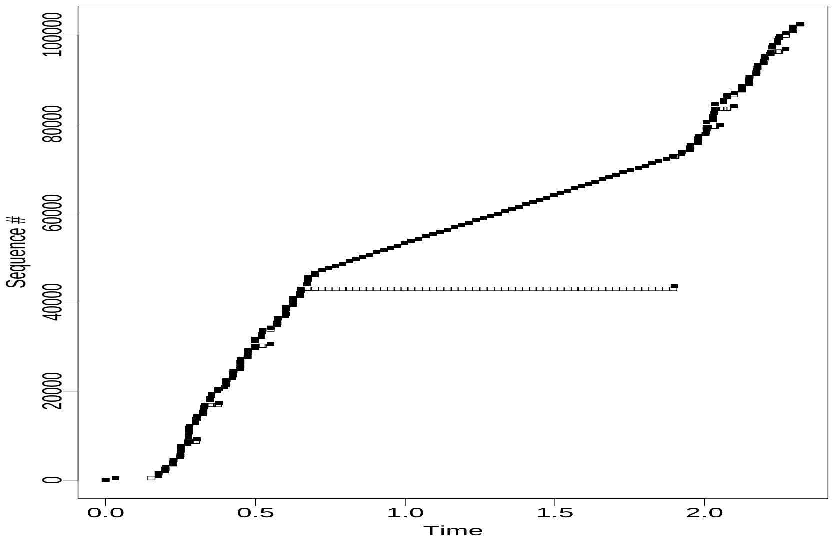

Figure 14.17: Self-clocking TCP “fast recovery”

necks in the Internet, except that in any case they are quite rare; that at least one link technology (ISDN) definitely exhibits them; and that some sites exhibit either true such links, or at least noise patterns resembling the multi-channel signature.

14.7.4 Estimation errors due to TCP behavior

In the previous section, we noted how TCP “self-clocking” can lead to a packet arrival pattern that matches that expected for a multi-channel bottleneck link quite closely, even though the bottleneck link is not in fact multi-channel. In this section we briefly illustrate another form of TCP behavior that can lead to false bottleneck estimates. Figure 14.17 shows a sequence plot of a connection clearly dominated by an unusually smooth and slow middle period.

What has occurred is that a single packet was dropped at about $T = 0.7$. Enough additional packets were in flight that 4 duplicate acks came back to the sender. The first 3 sufficed to trigger “fast retransmit” (§ 9.2.7), and the congestion window was such that the 4th led to the transmission of an additional packet carrying new data via the “fast recovery” mechanism (§ 9.2.7). However, the first packet retransmitted via fast retransmit was also dropped, while the fast-recovery packet carrying new data arrived successfully. This meant that the TCP receiver still had a sequence hole reflecting the original lost packet, so it sent another dup ack. The arrival of that duplicate then liberated another packet via fast recovery, and the cycle repeated 50 more times, until the original lost packet was finally retransmitted again, this time due to a timeout. Its retransmission filled the sequence hole and the connection proceeded normally from that point on.

Since the connection had an RTT of about 22 msec and only one fast recovery packet or dup ack was in flight at any given time, during the retransmission epoch the connection transmitted using "stop-and-go," with an effective rate of:

$$\frac{512 \text{ bytes}}{0.22 \text{ sec}} = 23 \text{ Kbyte/sec}.$$

PBM finds this peak rather than the true bottleneck of 1 Mbyte/sec, because the true bottleneck is obscured by the receiver's 1 msec clock resolution.

The TCP dynamics shown in the figure are quite striking. We note, however, that use of the SACK selective-acknowledgement option [MMFR96], now in the TCP standardization pipeline, will give the sender enough information to avoid situations like this one. We also note that, while this sort of TCP behavior is not exceptionally rare, this was the only such trace that we know PBM to have misanalyzed.

14.8 Efficacy of other estimation techniques

We finish with a look at how other, simpler bottleneck estimation techniques perform compared to PBM. Since PBM is quite complex, it would be useful to know if we can use a simpler method to get comparably sound results. In this context, the development of PBM serves as a way to calibrate the other methods. We confine our analysis to those traces for which PBM found a single bottleneck, as the other techniques all assume such a situation to begin with.

We further associate error bars with each PBM estimate. These either span the range of "consistent" estimates we found, where estimates are considered consistent if they lie within $\pm 20\%$ of the main PBM estimate (§ 14.6.2); or, if larger, the error bars reflect the inherent uncertainty in the PBM estimate due to limited clock resolution (§ 14.4.2). If another technique produces an estimate lying within the error bars, then we consider it as performing as well as PBM, otherwise not.

14.8.1 Efficacy of PR

In this section we evaluate the "conservative" and "optimistic" peak-rate (PR) estimators developed in § 14.5. These estimators were developed primarily as calibration checks for PBM, and we noted in their discussion that they will tend to underestimate the true bottleneck rate. Still, since they are simple to compute, it behooves us to evaluate their efficacy. We only evaluate them for $\mathcal{N}_2$, since they rely on the sending TCP enjoying a large enough window that it could "fill the pipe" and send at a rate equal to or exceeding the bottleneck rate (§ 9.3).

As we might expect, we find that the conservative estimate $\widehat{\text{PR}}^c$ given by Eqn 14.7 often underestimates the bottleneck: 60% of the time in $\mathcal{N}_2$, $\widehat{\text{PR}}^c$ was below the lower bound given by PBM; 39% of the time, it was in agreement; and 2% of the time it exceeded the upper bound, due to packet compression effects (§ 16.3).

Unfortunately, the more optimistic estimate $\widehat{\text{PR}}^o$ given by Eqn 14.8 only fares slightly better, underestimating 43% of the time, agreeing 52%, and overestimating 5% of the time.

We conclude that neither peak-rate estimator is trustworthy: they both often underestimate, because connections fail to fill the pipe due to congestion levels high enough to preclude an RTT's worth of access to the full link bandwidth.

14.8.2 Efficacy of RBPP

Receiver-based packet pair (§ 14.3) is equivalent to PBM with the extent limited to $k = 2$. (That is, it uses PBM's clustering algorithm to pick the best $k = 2$ estimate.) Consequently, we would expect it to do quite well in terms of agreeing with PBM, with disagreement potentially arising only due to clock resolution limitations for $k = 2$ (§ 14.4.2); delay noise on very short time scales such that pairs of packets are perturbed and do not yield a clear bandwidth estimate peak, but larger extents do; and multi-channel bottlenecks (not further evaluated in this section), one of the main motivations for PBM in the first place.

We find the RBPP estimate is almost always within $\pm 20\%$ of PBM's, disagreeing in $\mathcal{N}_1$ and $\mathcal{N}_2$ by more only 2-3% of the time. The two estimates are identical about 80% of the time, indicating PBM was usually unable to further hone RBPP's estimate by considering larger extents. Thus, if (1) PBM's general clustering and filtering algorithms are applied to packet pair, (2) we do packet pair estimation at the *receiver*, (3) the receiver benefits from sender timing information, so it can reliably detect out-of-order delivery and lack of bottleneck "expansion," and (4) we are not concerned with multi-channel effects, then packet pair is a viable and relatively simple means to estimate the bottleneck bandwidth.

14.8.3 Efficacy of SBPP

We finish with an evaluation of one form of *sender*-based packet pair (SBPP). SBPP is of considerable interest because a sender can use it without any cooperation from the receiver. This property makes SBPP greatly appealing for use by TCP in the Internet, because it works with only *partial deployment*. That is, SBPP can enhance a TCP implementation's decision-making for every transfer it makes, even if the receiver is an old, unmodified TCP. We expect SBPP to have difficulties, though, due to noise induced by networking delays experienced by the acks, as well as variations in the TCP receiver's *response delays* in generating the acks themselves (§ 11.6.4).

The bottleneck bandwidth estimators previously studied are both sender-based [Bo93, CC96a]. They differ from how sender-based TCP packet pair would work in that those schemes use "echo" packets. As noted in the discussion of Figure 14.14, Internet paths do not always have symmetric bottlenecks in the forward and reverse directions. Consequently, echo-based techniques will sometimes perforce give erroneous answers for the forward path's bottleneck rate. For TCP's use, however, the "echo" is the acknowledgement of the data packet. Except for connections sending data in both directions simultaneously, which are rare, these echoes are therefore returned in quite small ack packets. Consequently, bottleneck asymmetry will not in general perturb SBPP for TCP. Another significant difference is that, for TCP, usually an echo is only generated for every other data packet (§ 11.6.1). Consequently, the interval between each pair of acks arriving at the sender echoes the difference in time between the arrivals of *two* data packets at the receiver, rather than the arrivals of consecutive data packets. Because of this loss of fine-scaled timing information, TCP SBPP cannot detect the presence of multi-channel links, since doing so requires observing per-packet timing differences. (It will instead see timings corresponding to an extent of $k = 4$, which, for 2-channel and 3-channel links, is in fact the true bottleneck rate.)

To fairly evaluate SBPP, we assume use by the sender of the following considerations for generating "good" bandwidth estimates:

1. The sender always correctly determines how many user data bytes arrived at the receiver

between when it sent the two acks.

2. The sender does not consider pairs of acks if the first ack was for all the outstanding data, as such a pair is guaranteed to have a spurious RTT delay between the first and second ack.

3. The sender never bases an estimate on an ack that is for only a single packet's worth of data (MSS), as these often are delayed acks, and the sender lacks sufficient information to remove the timer-induced additional delay.

4. The sender never bases an estimate on an ack that does not acknowledge new data. This prevents the sender from using inaccurate timing information due to packet loss or reordering.

5. The sender keeps track of the sending times for its data packets, so it can determine the *sender expansion factor* (§ 14.5):
$$\widetilde{\xi}_{s,s} = \frac{\Delta T_a + C_s}{\Delta T_d + C_s},$$
where ΔT_a is the elapsed time between the arrival of successive acks, ΔT_d is the elapsed time between the departure of the first and last data packet being acknowledged, and C_s is the sender's clock resolution.

   The sender rejects an estimate if $\widetilde{\xi}_{s,s} < 0.9$. We use 0.9 instead of 1.0 as a “fudge factor” to account for self-clocking, which sometimes occurs at exactly the bottleneck rate.

The sender also computes “acceptable” estimates, which are those that do not conform to all of the above considerations, but at least conform to the first two. (These estimates will be used if SBPP cannot form enough “good” estimates.)

After collecting “good” and “acceptable” estimates for the entire trace, we then see whether we managed to collect 5 or more “good” estimates. If so, we take their 95th percentile as the bottleneck estimate (allowing for the last 5% to have been corrupted by ack compression, per § 16.3.1). If not, then we take the median of the “acceptable” estimates as our best guess.

We find, unfortunately, that SBPP does not work especially well. In both datasets, the SBPP bottleneck estimate lies within $\pm 20\%$ of the PBM estimate only about 60% of the time. About one third of the estimates are too low, reflecting inaccuracies induced by excessive delays incurred by the acks on their return, with the median amount of underestimation being a factor of two (and the mean, more than a factor of four). The remaining 5–6% are overestimates, reflecting frequent ack compression (§ 16.3.1), with an $\mathcal{N}_1$ median overestimation of 60% and a mean of 175%, though in $\mathcal{N}_2$ these dropped to 45% and 75%.

A final interesting phenomenon in $\mathcal{N}_2$ is that, about 2% of the time, SBPP was unable to form *any* sound estimate. These all entailed connections to receivers that generated only one ack for each entire slow-start “flight” (§ 11.6.1). Since one of the considerations outlined above requires that the first ack of a pair not be an ack for all outstanding data (to avoid introducing a round-trip time lull that has nothing to do with the bottleneck spacing), if the network does not drop any data packets, then such a receiver will *only* generate acks for all outstanding data, so the SBPP algorithm above fails to find any acceptable measurements.

14.8.4 Summary of different bottleneck estimators

In our evaluation of the different bottleneck rate estimators, we found that PBM overall appears quite strong. It produces many bandwidth estimates that accord with known link speeds, and produces few erroneous results, except for a tendency to misdiagnose a multiple-channel bottleneck link in the presence of considerable delay noise.

Using PBM then as our benchmark, we found that the stressful “peak rate” (PR) techniques perform poorly, frequently underestimating the bottleneck, as we surmised they probably would when developing them in § 14.5. They did, however, serve as useful calibration tests when developing PBM, since they pointed up traces for which we needed to investigate why PBM produced an estimate less than that of the conservative PR technique, or greater than that of the optimistic PR technique.

We also found that receiver-based packet pair (RBPP) performs virtually identically to PBM, provided that we observe the requirements outlined in § 14.8.2, and are not concerned with detecting multi-channel bottleneck links. Unfortunately, one requirement for RBPP is sender cooperation in timestamping the packets it sends, so the receiver can detect out-of-order delivery and data packet compression. We have not investigated the degree to which these requirements can be eased, but this would be a natural area for future work.

We unfortunately found that sender-based packet pair (SBPP) does not fare nearly as well as RBPP. Even taking care to use only measurements the sender can deduce should be solid, SBPP suffers from ack arrival timings perturbed by queueing delays and ack compression. As a result, it renders accurate results less than 2/3's of the time.

Thus, receiver-based bottleneck measurement appears to hold intrinsic advantages over sender-based measurement, and fairly simple receiver packet pair techniques, with sender cooperation, gain all of the advantages of the more complex PBM, unless we are concerned with detecting multi-channel bottleneck links.

Finally, a particularly interesting question for future work to address is how *quickly* these techniques can form solid estimates. If we envision a transport connection using an estimate of the bottleneck bandwidth to aid in its transmission decisions, then we would want to form these estimates as early in the connection as possible, particularly since most TCP connections are short-lived and hence have little opportunity to adapt to network conditions they observe [DJCME92, Pa94a].