CHAPTER 4

Network-Level Parameters and Metrics

If you are distressed by anything external, the pain is not due to the thing itself, but to your estimate of it; and this you have the power to revoke at any moment.

— Marcus Aurelius (121–180)

Reality continues to ruin my life.

— Bill Watterson (1958–), Calvin and Hobbes

The workload of TCP connections represents the demands of applications for sending and receiving data in a reliable, ordered, and congestion-responsive manner. How well TCP can satisfy these demands depends on the conditions of the network path between the two endpoints of each TCP connection, and the way TCP reacts to these conditions. An obvious example of a network condition that affects TCP is congestion that leads to segment loss. When a data segment is lost, TCP must retransmit it, and this implies some reduction in performance (*e.g.*, throughput) as the same data segment (rather than a new one) has to be sent again. In addition, TCP considers loss as an indication of network congestion, and reacts by reducing its sending rate. Different versions of TCP implement different ways of adjusting this sending rate. This means that the characteristics of the set of segments in a TCP connection are not just a function of the source-level behavior of the endpoints. This fact will have profound implications for the validation of our approach to synthetic traffic generation.

Intuitively, demonstrating that synthetic traffic is "realistic" must be based on a comparison of the statistical properties of real and synthetic traffic. If these properties are reasonably approximated, we can argue with confidence that the traffic generation method and its underlying statistical model provide an adequate foundation for experimental networking research. The comparison can be performed at two levels. First, we can compare source-level properties using the a-b-t modeling approach (see for example section 3.5). Second, we can compare network-level properties, *i.e.*, properties of the actual segments that make up individual connections in real and generated traffic. The material in this chapter is concerned with developing methods for making this latter comparison meaningful.

Since network conditions have an important impact on TCP connections, comparing real and synthetic traffic at the network-level is difficult if network conditions are not incorporated to some extent into the traffic generation system. For example, if we generate traffic that is intended to resemble that of some real link, and connections on this link experience substantial loss rates, the characteristics of the synthetic traffic would be rather different if the synthetic traffic did not experience comparable loss rates. Otherwise, the synthetic traffic would experience higher transfer rates, shorter durations, *etc.* The first part of this chapter considers methods for characterizing three important, and perhaps the dominant, network-level properties of TCP connections: round-trip times, receiver window sizes, and loss rates. These three properties will be incorporated in our traffic generation method as input parameters, and will make synthetic traffic more comparable to real traffic. Additionally, we also examine the properties of a number of real traces to illustrate the wide range of network conditions in which TCP operates, and how this range changes from one network link to another.

The second part of the chapter considers the actual problem of comparing traffic at the network-level. The research literature has identified a number of statistical properties of traffic that can serve as metrics for assessing the realism of synthetic traffic. We describe these properties and consider their application in the context of comparing traffic traces. We also examine a number of real traces in light of these metrics. Our analysis reveals important differences between the traces, and uncovers some dependencies between network-level metrics and types of source-level behavior.

4.1 Network-level Parameters

4.1.1 Round-Trip Time

The Round-Trip Time (RTT) between two network hosts is defined as the time required to send a packet from one host to another plus the time required to send a packet in the reverse direction. These two times are often very similar, but may sometimes vary considerably (*e.g.*, in the presence of asymmetric routing). In general, round-trip times are not constant, since queuing delays, switching fabric contention, route lookup times, *etc.*, vary over the lifetime of a connection.

Impact of Round-Trip Time

Round-trip times play a very important role in TCP connections. As indicated in Chapter 3, the exchange of a request ADU and its response ADU (*i.e.*, an epoch) in a TCP connection requires at least one round-trip time. This is independent of the amount of data exchanged. In addition, the speed at which data can be delivered (known as throughput[1]), is also a function of the round-trip time of the TCP connection.

The minimum time between the sending of a data segment and the arrival of its corresponding acknowledgment is exactly one round-trip time. Without TCP's window mechanism, TCP would only be able to send one segment per round-trip time, since it would have to wait for the acknowledgment before sending the next data segment. Therefore, peak throughput would be given by the maximum segment size S divided by the round-trip time R. This would imply that the longer the round-trip time, the lower the throughput S/R of the connection would be. In order to increase performance, a TCP endpoint can send a limited number of segments, a *window*, to the other endpoint before receiving an acknowledgment for the first segment. The number of segments W in the window gives the peak throughput of a TCP connection, $\frac{W*S}{R}$. This peak throughput can be lower if the path between the two endpoints has a capacity C that is lower than $\frac{W*S}{R}$, so the peak throughput of a TCP connection is given by $min(\frac{W*S}{R}, C)$. This implies that if W is not large enough to fill the available capacity C, R is the limiting factor in the peak throughput of a TCP connection.

A new TCP connection is not allowed to reach its peak throughput until it completes a "ramp-up" period known as *slow start* [Pos81]. The throughput of TCP during slow-start is also highly dependent on round-trip time. At the start of each connection, TCP does not make use of the entire window to send data, but rather probes the capacity of the network path between the two endpoints by sending an exponentially increasing number of segments during each round-trip time. This normally means that TCP sends only 1 segment in the first round-trip time, 2 in the second one, 4 in the third one, and so on, doubling the number of segments after each round-trip time until this number reaches a maximum of W segments. The throughput of the slow-start phase is therefore a function of round-trip time and maximum segment size, but it depends little on receiver window size and capacity. For example, an ADU that fits in 4 segments, requires 3 round-trip times to be transferred in the slow-start phase (one segment is sent in the first round-trip time, two in the second one, and one more in the final one), so

[1]More precisely, throughput is the rate of transfer taking into account not only application data but also control headers added by TCP and lower network layers. A related concept, goodput, is the rate of transfer of application data, *i.e.*, TCP payload. This distinction is important, but in the discussion above, throughput and goodput are affected similarly by round-trip times, so we simply talk about throughput.

the throughput of the connection is $\frac{4S}{3R}$. For common values of R and S, $S = 1460$ bytes and $R = 100$ milliseconds, the throughput would be 156 Kbps. This same ADU sent later in the connection using a single window would achieve a much higher throughput (*e.g.*, the four segments could be sent back to back, so they would reach the destination after only one half the round-trip time, $\frac{R}{2}$, achieving a throughput of $\frac{8S}{R} = 934$ Kbps).

Passive Estimation of Round-Trip Times

The dependency between TCP throughput and round-trip time implies that the distribution of round-trip times of the TCP connections found on a link has a substantial impact on the characteristics of a trace. If we intend to compare the throughputs of connections in traces from real links with those in synthetic traces, traffic generation must employ similar round-trip times. This requires us to be able to extract RTTs from a trace by analyzing packet dynamics. Extracting round-trip times from packet traces has received only limited attention in the literature [JD02, AKSJ03]. Nonetheless we can refine some of the existing ideas to obtain the distribution of round-trip times of connections in a trace in a manner that is useful for traffic generation.

Before we describe several methods for characterizing round-trip times, it is important to point out that the round-trip time of a TCP connection is not a fixed quantity. The time required for a segment to travel from one endpoint to another has several components. Transmission and propagation delays are more or less constant for a given segment size, but queuing delays, medium access contention, and router and endpoint processing, introduce variable amounts of extra delay. The TCP segments observed in our traces are exposed to these delays, whose variability is not always negligible, as our later measurement results illustrate. In summary, the segments of a TCP connection are exposed to a distribution of round-trip times, rather than to a fixed round-trip time.

We can think about the segments of a TCP connection as probes that sample the dynamic network conditions along their path, experiencing variable delays. As shown in the previous chapter, most TCP connections carry a small amount of data, providing only a few samples of these underlying conditions. This makes it very difficult to fully characterize the distribution of round-trip times experienced by an individual connection using only passive measurement methods (*i.e.*, only by looking at packet headers). In addition to the low number of samples per connection, TCP's delayed acknowledgment mechanism adds extra delays to some samples. This introduces even more variability, this time unrelated to the path of the connection. As we discuss below, the presence of delayed acknowledgments makes statistics

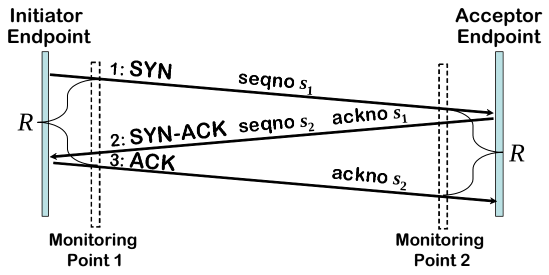


Figure 4.1: A set of TCP segments illustrating RTT estimation from connection establishment.

(such as the mean and standard deviation) computed from RTT samples, grossly overestimate the true mean and standard deviation of the underlying distribution of round-trip times. In our work, we favor more robust statistics, such as the median, or the minimum, which provide a good way of characterizing the non-variable component of a connection's round-trip time. For simplicity, our traffic generation will simulate the minimum round-trip time observed for each connection.

The SYN Estimator

The simplest way of estimating the round-trip time of a connection from its segment header is to examine the segments sent by the initiator endpoint during connection establishment. The use of this *SYN estimator* is illustrated in Figure 4.1. The initial SYN segment sent from the initiator to the acceptor is supposed to be immediately acknowledged by a SYN-ACK segment sent in the opposite direction. The initiator endpoint would then respond to the SYN-ACK segment[2] with an ACK segment. The initiator may or may not piggy-back data on this segment, but this does not affect RTT estimation significantly. The time between the arrival of the SYN segment and the arrival of the ACK segment is the round-trip time R of the connection (more precisely, a sample of the round-trip time). Measuring R using the departure times of the SYN and the ACK segments from the initiator endpoint gives approximately the same result as measuring R using the arrivals of these segments at either the monitoring point or the connection acceptor. In general, the SYN estimator is a good indicator of the minimum round-trip time, *i.e.*, total transmission and propagation delay. This is because TCP endpoints respond immediately[3] to connection establishment segments, and also because the small packets used by the SYN estimator are less likely to encounter queuing delays that the larger ones found later in the connection. The SYN

[2]For simplicity, our illustrations use acknowledgment numbers that refer to the cumulative sequence number accepted by the endpoint, which is one unit below the actual acknowledgment number stored in the TCP header [Pos81].

[3]Endpoints are not required to behave in this manner by any RFC, but it makes little sense to delay the acknowledging of SYN segments. On the contrary, delaying the acknowledging of data segments gives the endpoints a chance to receive a second data segment and acknowledge both data segments using a single acknowledgment.

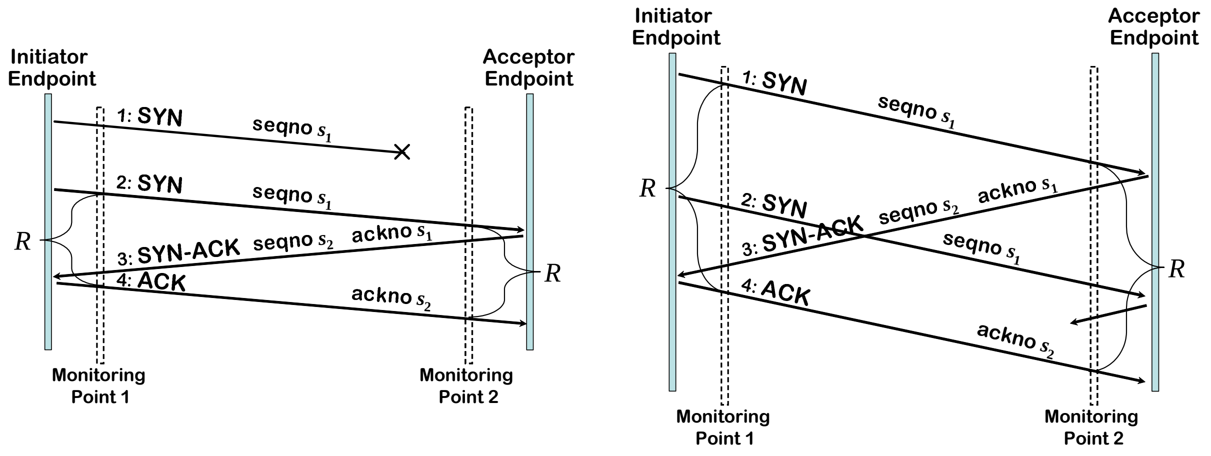


Figure 4.2: Two sets of TCP segments illustrating RTT estimation ambiguities in the presence of loss (left) and early retransmission (right) in connection establishment.

estimator has been a popular means of estimated round-trip times [AKSJ03, CCG$^+$04].

The SYN estimator has a number of shortcomings. First, the estimator provides a single sample of the round-trip time, which may be a poor representative of the average round-trip time of the connection. Second, partially-captured connections in a trace may not include connection establishment segments, so the SYN estimator cannot be used to determine their round-trip times. Third, the round-trip time of a connection with a retransmission of the SYN segment (or of the SYN-ACK segment), cannot be estimated with confidence, since the coupling of the SYN and the ACK segments becomes ambiguous. The problem is that the monitor may see two instances of the SYN segment, and either one could be coupled with the ACK for the purpose of computing the RTT. This difficulty is illustrated in Figure 4.2. The left side of the figure shows an example of connection establishment in which the first SYN segment is lost. In this case R is the time between the arrival of the second SYN segment and the arrival of the ACK segment, and not the time between the first SYN segment and the arrival of the ACK segment. However, it is not always correct to couple the last retransmission of the SYN with the ACK, and this is illustrated in the right side of the figure. The diagram shows a connection with such a large R that the initiator endpoint times out before the receipt of the SYN-ACK and sends an *early* (*i.e.*, unnecessary) retransmission of the SYN before the SYN-ACK reaches the initiator. In this case, R should be computed as the difference between the arrival times of the first SYN and the ACK, and not between the arrival times of the second SYN and the ACK. Note that standard TCP implementations time out and retransmit SYN segments after 3 seconds without receiving an acknowledgment [APS99]. The two cases result in exactly the same sequence of segments observed at the monitoring point, so it is not generally possible to accurately choose the right SYN to couple with the corresponding ACK

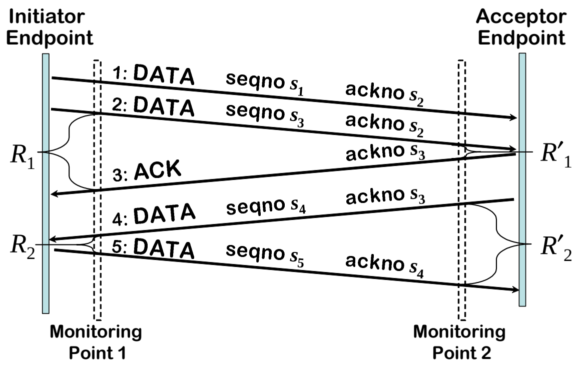

Initiator Endpoint
Acceptor Endpoint
1: DATA seqno s_1 ackno s_2
2: DATA seqno s_3 ackno s_2
ackno s_3
3: ACK
4: DATA seqno s_4 ackno s_3
5: DATA seqno s_5 ackno s_4
R_1
R_2
R'_1
R'_2
Monitoring Point 1
Monitoring Point 2

Figure 4.3: A set of TCP segments illustrating RTT estimation using the sum of two OSTTs.

segment by looking only at the sequence of segments [KP88a, Ste94].

The early retransmission of SYN segments when the RTT is greater than 3 seconds implies that the simple SYN estimator, at least in this basic form, cannot be used to study the tail of the round-trip time distribution (this issue has been overlooked in the literature). In theory, one could disambiguate the case of a timed-out SYN-ACK using the observation that SYN segments are retransmitted only after 3 seconds without receiving the SYN-ACK [Bra89]. However, our empirical observations show that this heuristic is unreliable as the timing of arrivals is imprecise, and not all TCP implementations seem to use the 3-second timeout properly. Detection of an unexpected retransmission of the SYN-ACK (or the ACK) can also be used to develop a heuristic, but cases with multiple losses can be very complicated to disambiguate.

The OSTT Estimator

A second technique for estimating round-trip times is illustrated in Figure 4.3. The location of the monitor divides the path of a connection into two sides, and we can estimate the *One-Side Transit Time* (OSTT) independently for each side. The sum of the two OSTTs gives an estimate of the round-trip time of the connection. The idea is that the arrival times of a data segment and its acknowledgment segment at the monitor provides an estimation of the OSTT from the measurement point to one of the endpoints. Round-trip time estimation using the OSTT method requires the collection of one or more samples of the OSTT between the initiator and the monitoring point, and one or more samples of the OSTT between the acceptor and the monitoring point. In Figure 4.3, a sample R_1 of the OSTT for the right side of the path (*i.e.*, OSTT between the acceptor and the monitoring point) is given by the difference in the arrival times of segments 2 and 3. A sample R_2 of the OSTT for the left side of the path (*i.e.*, between the initiator and the monitoring point) is given by the difference in the arrival times of segments 4 and 5. Thus, a sample of the full round-trip time R is given by $R_1 + R_2$. One way of seeing this graphically is to do the mental exercise of shifting the monitoring point toward the initiator. As we do this, the R_1 increases, while R_2 decreases. When the monitoring point reaches the initiator endpoint, R_1 is exactly the round-trip time of the connection, and R_2 is zero.

The OSTT-based estimation of the RTT is independent of the location of the monitoring point. For example, the arrival of segments at the second monitoring point in Figure 4.3 provides a sample $R'_1 + R'_2$ which is equal to $R_1 + R_2$. This is a substantial improvement over existing methods, since it implies that we can perform RTT estimation for connections observed at any point on their path. Previous work, such as Aikat *et al.* [AKSJ03], constrained itself to traces collected very close the edge of the path, so they could assume that the delay between the monitoring point and local networks was minimal. This results in an estimate in which only R_1 is computed under the assumption that R_2 is very small. The use of the sum of the OSTTs is more flexible, since it makes it possible to extract RTTs from any trace, and not just edge traces. This allowed us to analyze a backbone trace like Abilene-I, making our traffic analysis and generation technique more widely applicable.

There are, however, a number of difficulties with OSTT-based round-trip time estimation. A first problem is that each pair of segments provides a different estimation of the OSTT (due to differences in queuing delay and other sources of delay variability), so we have to decide how to combine the OSTT samples from one side of the connection with those from the other side. In other words, each connection provides a set of OSTT samples for one side, $\{R_{11}, R_{12}, \ldots, R_{1n}\}$, and another set of OSTT samples for the other side, $\{R_{21}, R_{22}, \ldots, R_{2m}\}$, where $n \geq 0$ and $m \geq 0$ are not necessarily equal. The question is then how to combine these samples into a single estimate of R, which we will call $\hat{R}$. If we assume low variability, we could simply sum the means of the two sets of estimates,

$$\hat{R} = \frac{\sum_{i=1}^{n} R_{1i}}{n} + \frac{\sum_{i=1}^{m} R_{2i}}{m}. \tag{4.1}$$

However, as we discuss in the next section, the sum of means can introduce substantial inaccuracy due to TCP's delayed acknowledgment mechanism.

A second problem is that the sum of OSTT samples requires at least one sequence number/acknowledgment number pair for each side of the connection. Otherwise, one of the sets of OSTT samples is empty, and we have no information about the delay on one side of the connection. This prevents us from using the sum of OSTTs estimator for connections that send data only in one direction.

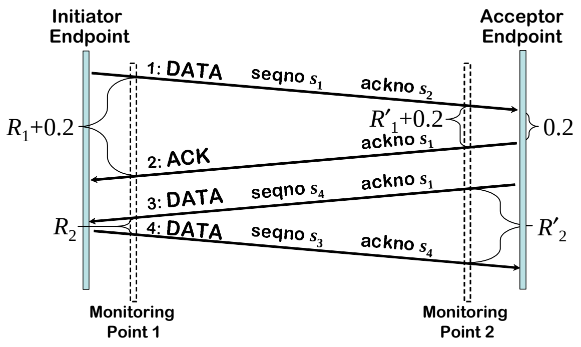


Figure 4.4: A set of TCP segments illustrating the impact of delayed acknowledgments on OSTTs.

Finally, we must note that the time between the arrival of a data segment and its first acknowledgment is not always a good estimator of the OSTT. This is mostly due to two causes: retransmission ambiguity and delayed acknowledgments. Retransmissions may create ambiguous cases in which we cannot match the pair of data and ACK segments. This is the well-known *retransmission ambiguity* problem, which was first discussed by Karn and Partridge [KP88b] in the context of estimation of TCP's retransmission timeout. Whenever a data segment is retransmitted, it is not possible to decide whether to compute the OSTT using the first or the second instance of the data segment. These data segments cannot therefore be used to obtain a new OSTT sample. This retransmission ambiguity is similar to the SYN retransmission problem shown in Figure 4.2.

Delayed acknowledgments can add up to 500 milliseconds[4] [Bra89] of extra delay in the OSTT estimates, whenever a segment is not acknowledged immediately. Figure 4.4 illustrates this problem. The right side OSTT is 200 milliseconds larger than it should be due to the delayed sending of the acknowledgment in segment 2. The distortion of OSTT samples caused by delayed acknowledgments is pervasive, since the number of segments in a window is often an odd number, and TCP implementations are allowed to keep (at most) one unacknowledged segment. An odd number of segments in a window means that the last segment does not trigger an immediate acknowledgment, which adds an extra delay to its corresponding sample. Furthermore, performance enhancement heuristics implemented in modern TCP stacks often add PUSH flags to TCP segments carrying data in the middle of an ADU, and this flag forces the other endpoint to immediately send an acknowledgment [Pos81]. This creates even more cases in which the last segment of the window has to be acknowledged separately using a delayed acknowledgment. The empirical results presented below illustrate the impact of this problem.

[4]Typical values are between 100-200 milliseconds.

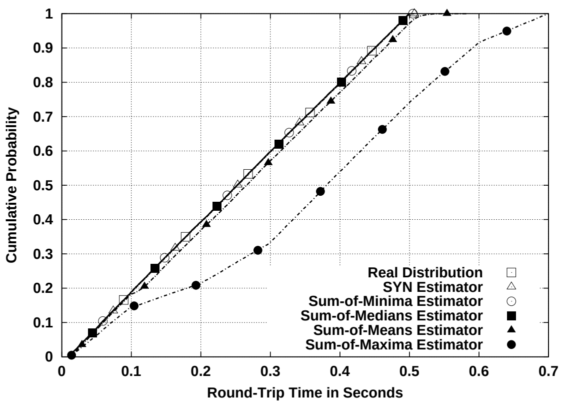


Figure 4.5: Comparison of RTT estimators for a synthetic trace: no loss and enabled delayed acknowledgments.

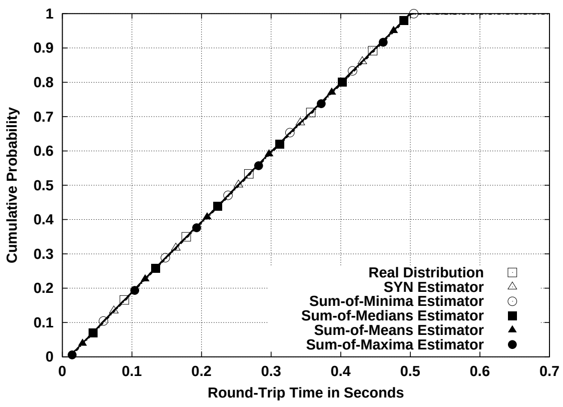


Figure 4.6: Comparison of RTT estimators for a synthetic trace: no loss and disabled delayed acknowledgments.

Validation of Round-Trip Time Estimators

We evaluated the round-trip time estimation techniques proposed above using synthetic traffic in a testbed where RTTs could be controlled precisely. Figure 4.5 shows the results of a first experiment, in which a uniform distribution of round-trip times between 10 and 500 milliseconds was simulated using a modified version of *dummynet* [Riz97]. Each connection had exactly the same round-trip time throughout its lifetime, so every one of its segments was artificially delayed by the same amount. During the experiment, a large number of single epoch connections was created. The sizes of a_1 and b_1 for each connection were randomly sampled from a uniform distribution with values between 10,000 and 50,000 bytes. We collected a segment header trace of the traffic and applied the round-trip time estimation techniques described above. Figures 4.5 and 4.6 compare the results. As shown in Figure 4.5 the SYN estimator can measure the distribution of round-trip times flawlessly in this experiment. The input distribution of RTTs (marked with white squares) exactly matches the distribution computed using the SYN estimator (marked with white triangles).

Figure 4.5 also studies the accuracy of several OSTT-based estimators. As discussed in the previous section, the analysis of the OSTTs in a TCP connection results in two sets of estimations, $\{R_{11}, R_{12}, \ldots, R_{1n}\}$ and $\{R_{21}, R_{22}, \ldots, R_{2m}\}$, for the initiator-to-monitor side and for the acceptor-to-monitor side respectively. For each connection, the estimated round-trip time $\hat{R}$ has to be derived from these collections of numbers. The figure shows the result of computing the distribution of round-trip times using four different methods of deriving $\hat{R}$. The first method is the sum-of-minima, where $\hat{R}$ is the sum of the minimum value in $\{R_{11}, R_{12}, \ldots, R_{1n}\}$ and the minimum value in $\{R_{21}, R_{22}, \ldots, R_{2m}\}$.

In the figure, the sum-of-minima estimation of the distribution of round-trip times (marked with white circles) is exactly on top of the input distribution, so this estimator is exact. The same is also true when the sum of medians is used. This shows that there is no significant variability between the minimum and the median of each set of OSTTs, which is expected in our uncongested experimental environment.

Figure 4.5 shows another two distributions derived from OSTT samples that are less accurate characterizations of the real RTT distribution in the testbed experiment. The distribution (marked with black triangles) of round-trip times obtained using the sum of the mean of the OSTTs, *i.e.*, Equation 4.1, is slightly heavier that the real distribution of round-trip times. This is due to the presence of a few OSTT samples that are above the real OSTT of the connection, which skew the mean but not the median or the minimum. The magnitude of these larger samples is strikingly illustrated by the curve corresponding to the sum of the maximum OSTTs (marked with back circles). This curve is far heavier than the previous one, and certainly a poor representative of the original distribution of round-trip times. The use of the maximum makes this last estimator focus on the largest OSTTs, which are shown to be quite far from the true values of the OSTT. The exact cause of this inaccuracy is the use of delayed acknowledgments in TCP, which was illustrated in Figure 4.4. Delayed acknowledgments make some OSTT samples include extra delays due to the behavior of the TCP stack and not the path between the endpoints. In particular, the distribution computed using the sum-of-maxima is 200 milliseconds heavier than the input distribution for most of its values. This is consistent with the default value of FreeBSD's delayed acknowledgment mechanism, which is 100 milliseconds. Connections where both the initiator-to-monitor and theR_1 acceptor-to-monitor sets of OSTTs have values from delayed acknowledgments result in values of $\hat{R}$ equal to $R + 100 + 100$ milliseconds.

To confirm this hypothesis, we conducted a second experiment, with exactly the same setup, although this time TCP's delayed acknowledgment mechanism was completely disabled. The results of estimating the distribution of round-trip times in this second experiment are shown in Figure 4.6. Every estimation method is accurate in this case, which proves our hypothesis about the impact of delayed acknowledgments. The conclusion is that the first three estimators are preferable, since they are robust to the inaccuracy that delayed acknowledgments introduce when measuring round-trip times from segment header traces. Interestingly, the impact of delayed acknowledgment on passive RTT estimation has been overlooked in the literature [Ost, AKSJ03, JD02].

The discussion of the RTT estimation methods in the previous section pointed out the need to filter out samples from retransmissions. The previous two experiments were run in an uncongested testbed,

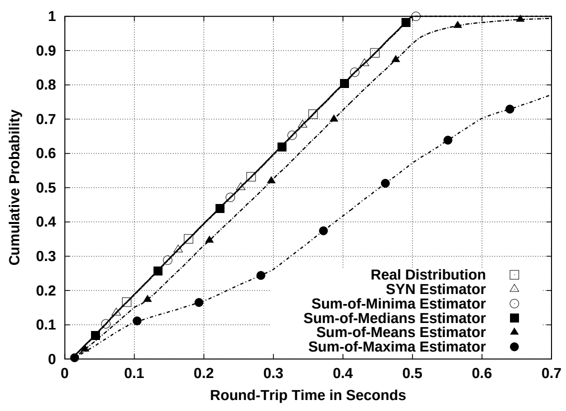


Figure 4.7: Comparison of RTT estimators for a synthetic trace: fixed loss rate of 1% for all connections.

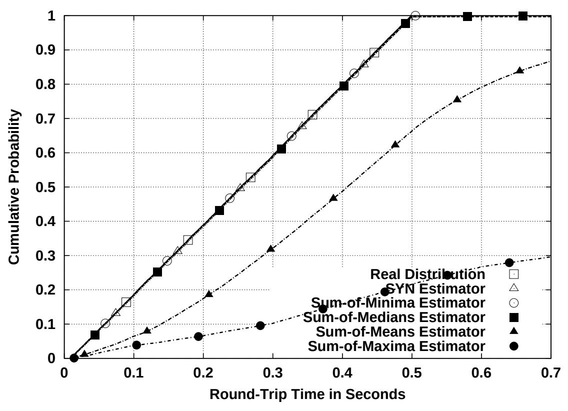
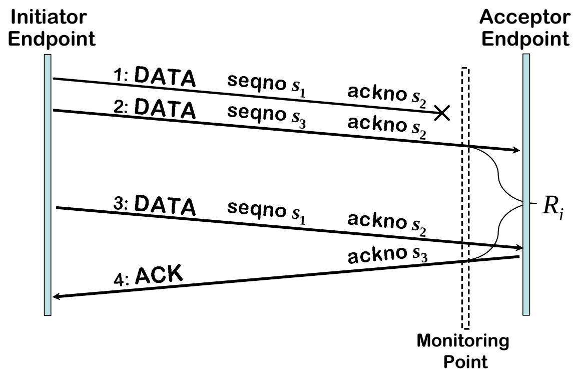


Figure 4.8: Comparison of RTT estimators for a synthetic trace: loss rates uniformly distributed between 0% and 10%.



Initiator Endpoint
Acceptor Endpoint
1: DATA seqno s_1 ackno s_2
2: DATA seqno s_3 ackno s_2
3: DATA seqno s_1 ackno s_2
ackno s_3
4: ACK
R_i
Monitoring Point

Figure 4.9: A set of TCP segments illustrating an invalid OSTT sample due to the interaction between loss and cumulative acknowledgments.

where no losses were expected. Since loss is common in the real traces that we study in this dissertation, we further validated these methods using experiments where *dummynet* was used to introduce artificial loss rates under our control. Figure 4.7 compares the six distributions obtained using the six RTT estimators in an experiment with a fixed loss rate of 1%. Once again the first three estimators measure the distribution of physical round-trip times accurately, while the sum-of-means and the sum-of-maxima overestimate the true distribution. The overestimation is even more pronounced in another experiment in which loss rates were uniformly distributed between 0% and 10%. The estimated RTT distributions are shown in Figures 4.8 and 4.10. The first figure uses the same range in the x-axis as Figure 4.7, while the second figure uses a broader range in the x-axis, between 0 and 5 seconds. The first three estimators are not affected by losses, but the RTT distribution computed by the sum-of-means estimator is substantially heavier than the original. Similarly, the distribution computed by the sum-of-maxima is several times larger than the real distribution.

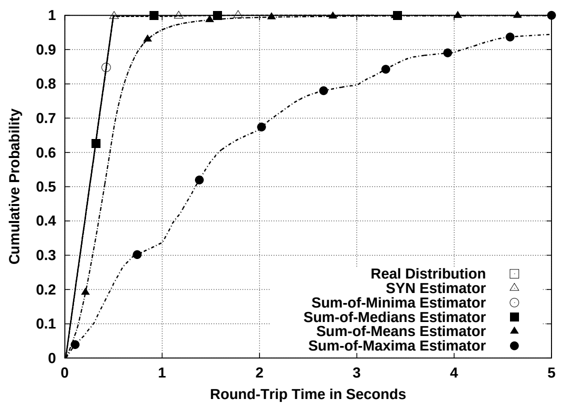


Figure 4.10: Comparison of RTT estimators for a synthetic trace: loss rates uniformly distributed between 0% and 10%.

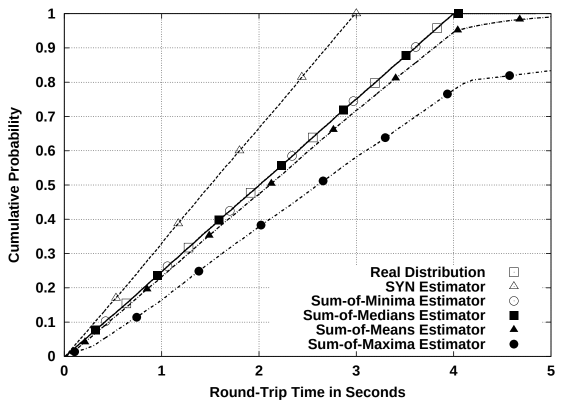


Figure 4.11: Comparison of RTT estimators for synthetic traces: fixed loss rate of 1%; real RTTs up to 4 seconds.

The cause of the additional inaccuracy in the sum-of-means estimator is the interaction between losses and TCP's cumulative acknowledgment mechanism, which prevent us from disambiguating samples from retransmissions. This problem is illustrated in Figure 4.9. Segments 1 and 2 with sequence numbers s_1 and s_2 respectively are sent from the initiator to the acceptor, but segment 1 is lost before the monitor. Since TCP's acknowledgments are cumulative, this means that the acceptor endpoint cannot acknowledge segment 2 alone[5]. Some time later, after the initiator times out, another segment with sequence number s_1 is sent from initiator to acceptor. Upon its arrival, the acceptor can send a cumulative acknowledgment with sequence number s_2. Using the timestamps of segments 2 and 4, we could compute an OSTT R_i. However, R_i is clearly not a good representative of the OSTT between the monitor and the acceptor, and therefore this sample is incorrect. The true value of the OSTT would be the difference between the timestamps of segments 3 and 4, which is much smaller than R_i. In this example, filtering samples from retransmitted sequence numbers does not help, since no retransmission was observed for s_2. In general, it is important to either filter out any sample associated with reordering (*e.g.*, segment 3 which has a lower sequence number than segment 2), or use an estimator, such as the sum-of-medians, that is robust to the distortion created by samples like R_i. Otherwise, OSTTs can be substantially overestimated, as illustrated in Figure 4.8.

Figure 4.11 reports on another experiment in which round-trip times were distributed between 10 and 4,000 milliseconds, and the underlying loss rate was 1%. Unlike the previous experiments, the SYN estimator results in a lighter distribution of round-trip times than the original one. This is due to the SYN retransmission timeout, which is set to 3 seconds [Bra89]. Connections with a round-trip

[5]Some implementations send an ACK whenever an out-of-order data segment is received, like Segment 2 in this case, but this behavior is not mandated by Internet standards. RFC 2581 [APS99] only recommends it.

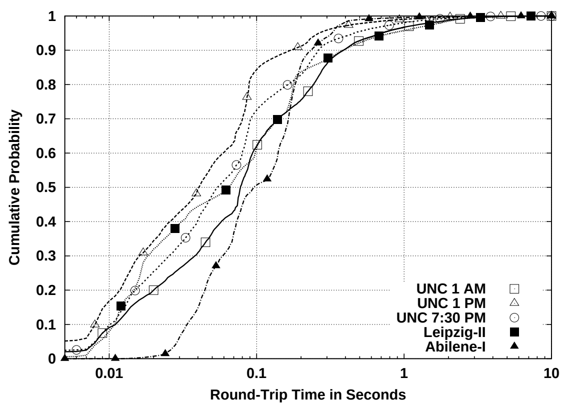


Figure 4.12: Bodies of the RTT distributions for the five traces.

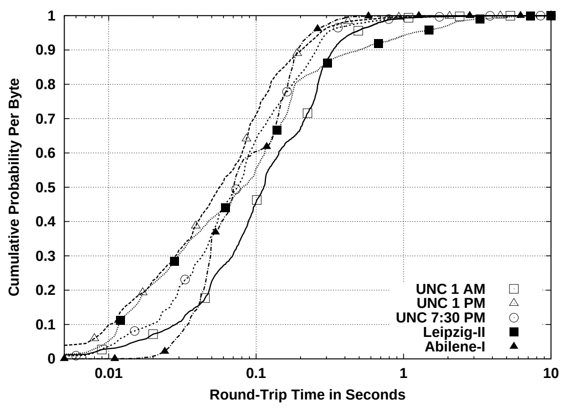


Figure 4.13: Bodies of the RTT distributions with per-byte probabilities for the five traces.

time above 3 seconds always retransmit their SYN segment, and therefore make their SYN estimator invalid. Therefore, these connections provide no samples when the SYN estimator is used, resulting in a distribution of RTTs limited to a maximum of 3 seconds. However, in these cases, the sum-of-minima and the sum-of-medians estimator were again able to estimate the distribution of round-trip times accurately.

Measurement Results

Figure 4.12 shows the distributions of round-trip times computed using the sum-of-minima estimator for the five traces listed in Table 3.1. The first observation is that the distribution of round-trip times is significantly variable across sites and for different times of the day at the same site. While the majority of round-trip times are between 7 milliseconds and 1 second for UNC and Leipzig, they are distributed in a far narrower range, between 20 milliseconds and 400 milliseconds, for Abilene-I. This is probably due to the fact that the Abilene-I trace was collected in the middle of a backbone network that mostly carries traffic between US universities and research centers so intercontinental round-trip times are very uncommon in this trace. This is also a lightly loaded network, so extra delays due to queuing are very uncommon. Note also that the distributions for UNC become lighter as we consider busier times of the day. The cause for this is an open question. The distribution for Leipzig-II does not exactly match any of the ones for UNC, but its body fluctuates within the envelope formed by the UNC distributions.

Figure 4.13 shows the same distributions but the probability of each round-trip time is computed for each byte rather than for each connection. A probability of 0.5 in this plot means that 50% of the bytes were carried in connections with a round-trip time of a given value or less. For example, for the

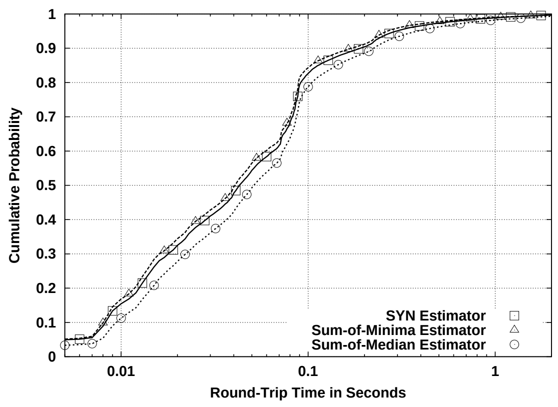


Figure 4.14: Comparison of the sum-of-minima and sum-of-medians RTT estimators for UNC 1 PM.

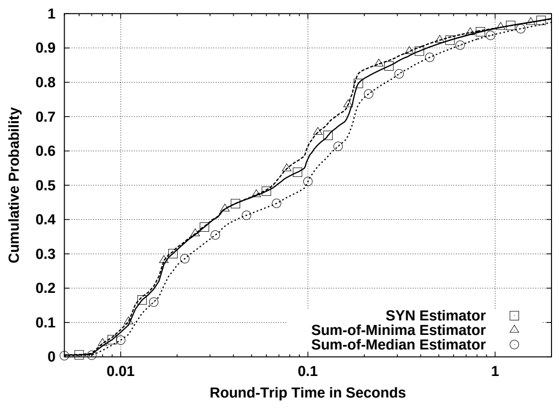


Figure 4.15: Comparison of the sum-of-minima and sum-of-medians RTT estimators for Leipzig-II.

UNC 1 AM trace, 50% of the bytes were carried in connections that experienced round-trip times of 110 milliseconds or less. Previously (*e.g.*, Figure 4.12) a probability of 0.5 meant that 50% of the connections experienced round-trip times of a given value or less. In general, we observe that the smallest round-trip times are somewhat less significant in terms of bytes than they are in terms of connections. Interestingly, Abilene-I does not differ much from the other distributions in this case. Another interesting observation is that a substantial number of bytes in the Leipzig-II traces were carried in connections with round-trip times between 300 milliseconds and 3 seconds, and this phenomenon is not observed for the other distributions. This could be explained by the location of this link in Europe, and the fact that it may carry a significant amount of traffic to distant US servers.

Figures 4.14 and 4.15 compare the variability in the results when different estimators are used for the same trace. In the UNC 1 PM trace and the Leipzig-II trace, the sum-of-medians estimator results in a somewhat heavier distribution of round-trip times but maintains more or less the same shape of the distribution. Given that these estimators were shown to be robust to losses and TCP artifacts in the previous section, the difference between the sum-of-minima and the sum-of-medians seems due to true round-trip time variability. While we are not implementing RTT variability within individual TCP connections in our experiments, it seems possible to reproduce this variability during traffic generation. This could be achieved by combining the distributions from the sum-of-minima and the sum-of-medians to give connections more variable round-trip times. For example, given a connection with a sum-of-minima estimate of $\hat{R}_{min}$ and sum-of-medians estimate of $\hat{R}_{median}$, let $\delta = \hat{R}_{median} - \hat{R}_{min}$. During traffic generation, the segments of this connection could be delayed by a random quantity between $\hat{R}_{median} - \delta$ and $\hat{R}_{median} + \delta$, or some variation of this scheme. Note that this basic method needs to be refined to eliminate segment reordering, which would occur frequently with the described approach.

4.1.2 Receiver Window Size

When a segment is received by a TCP endpoint, its payload is stored in an operating system buffer until the application uses a system call to receive the data. In order to avoid overflowing this buffer, TCP endpoints use a field in the TCP header to tell each other about the amount of free space in this buffer, and they never send more data than can possibly fit in this buffer. This mechanism, known as *flow control*, imposes a limit on the maximum throughput of a TCP connection. A sender can never send more data than the amount of free buffer space at the receiver. We refer to this free space as the receiver window size. The TCP header of each segment includes the size of the receiver window on the sender endpoint at the time the segment was sent. This value is often called the "advertised" window size, and defined as a "receiver-side limit on the amount of outstanding (*i.e.*, unacknowledged) data" by RFC 2581 [APS99]. The size of the advertised window shrinks as new data reach the endpoint (since data are placed in the TCP buffer), and grows when the application using the TCP connection consumes these data (which are removed from the TCP buffer).

A TCP connection with a maximum receiver window of W segments[6], a maximum segment size of S bytes, and a round-trip time of R seconds, can at most send data at $\frac{W*S}{R}$ bytes per second. This peak throughput can be further constrained by the capacity of the path C, so peak throughput is $min(\frac{W*S}{R}, C)$. As we will show, connections often use small receiver window sizes that significantly constrain performance, *i.e.*, $\frac{W*S}{R} << C$, and this should be taken into account during traffic generation.

We can measure the distribution of receiver window sizes by examining segment headers. As pointed out in [CHC$^+$04b], some TCP implementations (*e.g.*, Microsoft Windows) do not report their maximum receiver window size in their first segment (*i.e.*, the SYN or SYN-ACK) as one would expect, but do it in their first data segment. This is because some implementations allocate a small amount of buffering (*e.g.*, 4 KB) to new TCP connections, but increase this amount after connection establishment is successfully completed (*e.g.*, increasing it to 32 KB). In our work, we compute the maximum receiver window sizes as the maximum value of the advertised window size observed in the segments of each TCP connection. This gives us two maximum receiver window sizes per connection, one for each endpoint. There is no reason why the two endpoints must use receiver windows of equal size.

[6]The advertised receiver window size is given in bytes in the TCP header. We describe it here and in section 4.1.1 in terms of segments for convenience when considering the impact of round-trip times.

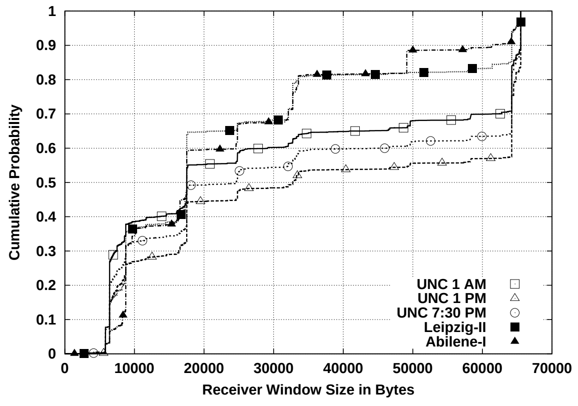


Figure 4.16: Bodies of the distributions of maximum receiver window sizes for the five traces.

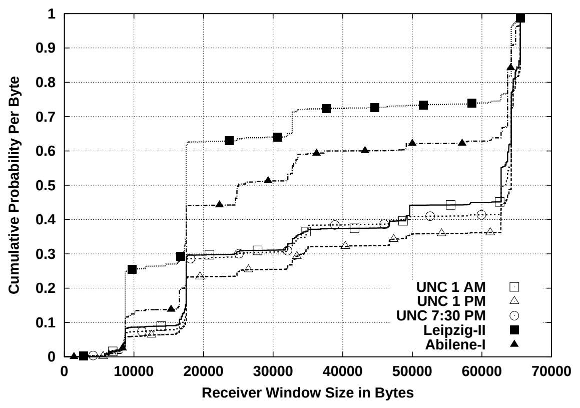


Figure 4.17: Bodies of the distributions of maximum receiver window sizes with per-byte probabilities for the five traces.

Figure 4.16 shows the distribution of maximum receiver window sizes in five traces. In general, window sizes are a multiple of the maximum segment size (usually 1,460 bytes), so we observe numerous jumps in the cumulative distribution function. Notice for example the jumps at 12 segments, $1,460*12 = 17,520$ bytes (approximately 16 KB), and 44 segments, $1,460 * 44 = 64,240$ bytes (approximately 64 KB). The field in the TCP header that specifies the receiver window size is 16 bits long, so the maximum receiver window size is 65,535 bytes[7].

We can make two interesting observations from Figure 4.16. First, a significant fraction of the connections used small receiver window sizes in all traces. For example, between 45% and 65% of the connections had window sizes below 20,000 bytes. Second, we observe a surprising difference between the UNC distributions and those from Leipzig-II and Abilene-I. We see a much larger fraction of the largest windows at UNC, suggesting a different distribution of endpoint TCP implementations, or widespread tuning of the servers located on the UNC campus. This is in sharp contrast to the results for round-trip times, where Leipzig-II and UNC were alike and quite different from Abilene-I.

Figure 4.17 shows an alternative view of the distributions of maximum receiver window sizes by computing the probability that each byte in the traces was carried in a connection with certain maximum receiver window size. The plot shows that connections with the largest window sizes carry many more bytes than those with small sizes. This is likely to be explained by tuning of the TCP endpoint parameters

[7]Some implementations support the window scaling option described in [JBB92], which enables larger windows. These larger windows are specified as the product of the receiver window size encoded in a 16-bit field in the TCP header, and a multiplier encoded in a TCP option (almost always 64 KB). We have not studied this feature in our work. The use of the window scaling is negotiated by the endpoints using a TCP header option, and TCP options are often not included in segment header traces, making the analysis difficult. It would however be possible to study the maximum amount of unacknowledged data in each connection, which would allow us to identify violations of the advertised window. For these cases, we could estimate the scaled window size by multiplying the advertised window by 64 KB.

by administrators and server vendors in environments with large data transfers.

4.1.3 Loss Rate

TCP reacts to loss by retransmitting segments, which makes TCP a *reliable* transport protocol, and reducing its sending rate, a mechanism known as *congestion control*. The reduction in sending rate is implemented using a TCP variable known as the congestion window size G, which further limits the maximum number of packets that can be sent by one endpoint. Throughout the lifetime of a TCP connection, TCP endpoints are only allowed to have a maximum of $min(G, W)$ outstanding (unacknowledged) segments in the network. This limits peak throughput to $min(\frac{min(G,W)*S}{R}, C)$.

The size of the congestion window is reduced every time TCP detects loss, so lossy connections have lower throughput than lossless ones. Numerous papers have developed analytical expressions that consider the impact of loss on average throughput. These papers make use of different analysis techniques and consider different models of TCP behavior and loss patterns. However, the simple relationship between loss and rate given in [MSM97] is enough to illustrate the basic impact of loss. In general, the average throughput of a TCP connection is $\frac{S*K}{R\sqrt{p}}$, where S is the maximum segment size, K is a constant equal to $\sqrt{\frac{3}{2}}$, R is the round-trip time and p is the loss rate. Therefore, average throughput is inversely proportional to the square root of the loss rate p, and it decreases very quickly as p increases. Note that the maximum window size is not part of this equation, but peak throughput is still limited by W (and by round-trip time), as mentioned above.

We define the loss rate of a TCP connection as the number of lost segments divided by the total number of segments sent, l/s. Assuming segments have an equal probability of loss, the loss rate is equal to the probability of losing an individual segment. Measuring the exact loss rate experienced by a TCP connection depends on our ability to count all segments, including those that may be lost before the monitoring point, and detecting all losses, which may occur before or after the monitoring point. The exact calculation of the loss rate of a connection is a very difficult task. In our work, we make use of two heuristics that should provide a good approximation of a connection's loss rate. We make no attempt to address the most difficult and ambiguous cases of loss detection, which our experience leads us to believe are uncommon.

Our measurement of loss rate from traces of segment headers relies on detecting retransmissions and making use of the same indications of loss that TCP employs. For each connection, we compute the total number of segments transmitted s as the total number of *data* segments in the connection. In addition, we compute the total number of lost segments l using the number of retransmitted data segments r, and the number of triple duplicate acknowledgment events d. We need both numbers r and d, since they provide complementary information. Triple duplicates can tell us about losses that occur before the monitoring point, which do not create observable retransmissions. Retransmissions can tell us about losses recovered using the retransmission timer, which do not create triple duplicates.

Estimating the loss rate p of a TCP connection simply as $(r + d)/s$ tends to overestimate loss rate when the monitoring point is located after the point of loss. In the most common situation, when the loss of a segment in one direction happens *before* the monitoring point, the trace collected at the monitoring point includes no retransmission and sends three duplicate acknowledgments in the opposite direction. These acknowledgments share the same sequence number, which corresponds to the sequence number of the segment that preceded (according to TCP's logical data order) the lost segment. However, when the loss happens *after* the monitoring point, the trace includes both a retransmission, in the direction in which the loss occurred, and a triple duplicate acknowledgment event, in the opposite direction. We can therefore compute a better estimate of loss rate by ignoring the triple duplicate events whenever a corresponding retransmission is observed. Doing so means that triple duplicates are used to estimate loss before the monitoring point, while retransmissions are used to estimate loss after the monitoring point. Applying this idea, the estimate of loss rate that we use in our work is therefore $(r + d')/s$, where d' includes only triple duplicate events not associated with observed retransmissions.

Note also that our computation of the loss rate considers only losses of data segments, and not losses of pure acknowledgments. Losses of acknowledgments can also reduce the size of the congestion window G, but measuring acknowledgment loss rate is even harder given the cumulative nature of the acknowledgment numbers and the fact that endpoints may acknowledge every data segment received, or every other data segment. We are not overly concerned by this simplification. This is because under the assumption that losses are caused by congestion, pure acknowledgments are far less likely to be dropped given their much smaller size.

In order to study the accuracy of our estimation of loss rates, we conduct a number of controlled experiments similar to those used to evaluate the different round-trip time estimation techniques. Figure 4.18 shows the results of two laboratory experiments in which an artificial loss rate of 1% was imposed on connections carrying a single epoch with $a_1 = b_1 = 10,000,000$ bytes. Transferring ADUs of this size requires a minimum of 6,850 data segments. Losses were created using *dummynet* , so each connection

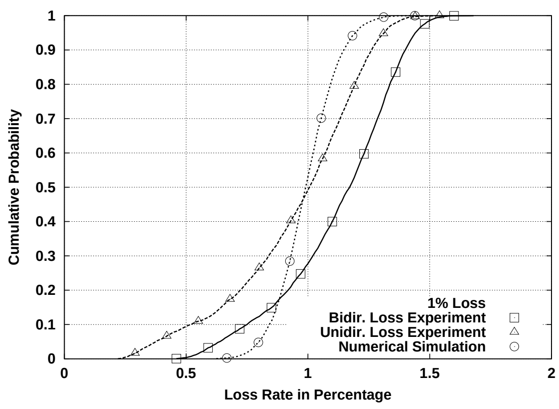


Figure 4.18: Measured loss rates from experiments with 1% loss rates applied only on one direction or on both directions of the TCP connections.

in the experiment had a drop probability equal to 0.01. The figure illustrates several points about our loss rate computation. We first compare the measured loss rates for two scenarios: one where a segment loss probability of 0.01 was applied by *dummynet* only to one direction of the connections, and another one where it was applied to both directions. In Figure 4.18, the first scenario is labeled “unidirectional loss experiment” and the second is labeled “bidirectional loss experiment”. The mean value of the data segment loss rate in the unidirectional loss experiment (marked with white triangles) was 1%, exactly the intended value. We also observe that 90% of the connections experienced loss rates between 0.5% and 1.5%. The bidirectional loss experiment (results marked with white squares) illustrates the dependency between the two directions of a TCP connection. The mean of the CDF is substantially higher for this experiment, and the distribution shows a fixed positive offset of 20%. This is because losses of acknowledgments in one direction also triggered retransmissions in the other, increasing the measured (data segment) loss rate. In other words, loss of acknowledgments inflated the estimated loss rates, since data was not really lost.

Our second observation about Figure 4.18 is that the range of the two distributions is quite wide, showing substantial variability around the target loss rate of 1%. This is partly explained by the random sampling in *dummynet* 's implementation of per-flow loss rates. *Dummynet* drops segments in an independent manner, by generating a random number between 0 and 1 for each segment, and only dropping a segment if its corresponding random number is between 0 and 0.01. This means that even with large ADUs, the drop probability rate experienced by the connection in the testbed experiments was not exactly 0.01.

In order to study the impact of this random sampling, we conduct a numerical simulation, and the result is illustrated using the third CDF in Figure 4.18. This distribution comes from simulating each connection by sampling a uniform distribution (with a range between 0 and 1) 6,850 times (the number of data segments in 10 MB). Each sample is meant to simulate one segment that may or may not be lost. If the value of the sample is equal to or greater than 0.01, the segment is not counted as a loss. If the sampled value is less than 0.01, the segment is counted as a loss. In this case, we continue to sample the uniform distribution until the value obtained is equal to or greater than 0.01. These extra samples are used to simulate the possibility of losing retransmissions, which can also be dropped by *dummynet* with the same probability. The result of this sampling process is two counts:

- the total number of segments s^* in the simulated connection, which is the number of times that the uniform distribution was sampled, and
- the total number of loss events l^*, which is the number of times that the samples from the uniform distribution were less than 0.01.

The ratio l^*/s^* is the simulated loss rate p^* for one connection. We repeated this process 4,200 times, which was the number of connections in the testbed experiments, and constructed a CDF of the resulting loss rates which is shown in Figure 4.18. The CDF exhibits substantial variability around 1%. Therefore, sampling variability partially explains the variability observed in the loss rates that we measured from the testbed experiments. Note that the variability in our lab experiments and in the numerical simulation is tied to the size of the ADU (10 MB in a minimum of 6,850 segments). Increasing this size would reduce the variability of the measured loss rates in proportion to the square root of the number of segments (a basic probability result for sample means). However, our illustration of the sampling variability using 10 MB ADUs is already conservative, since most TCP connections carry far less data and therefore need fewer segments.

The CDF from the numerical simulation provides us with a gold standard for our measurements, since our loss rate estimates should reflect the actual drop rates that *dummynet* imposed to the connections in the testbed. Still, further work is need to explain the remaining difference, and possibly refine our measurement technique. In any case, the experiments serve to confirm that our loss rate estimate is reasonably close to the true loss.

The distributions of loss rates in our collection of real traces is shown in Figure 4.19. Between 92.5% and 96.2% of the connections did not experience any losses, while the remaining connections did experience quite significant loss rates. This is consistent across all measured sites. The result is quite

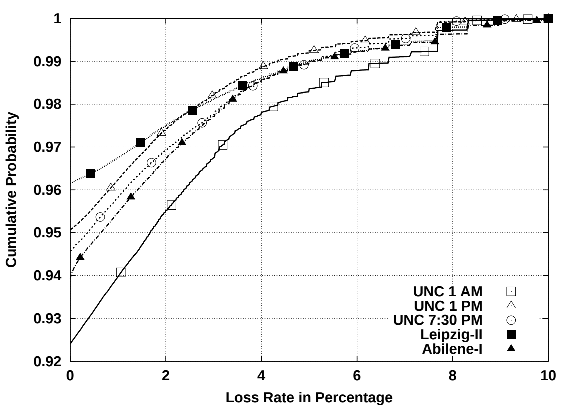


Figure 4.19: Bodies of the distributions of loss rates for the five traces.

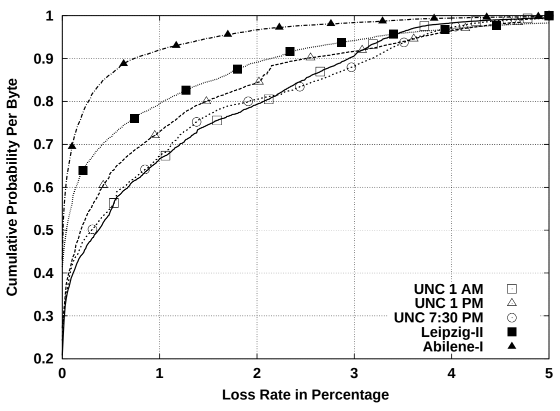


Figure 4.20: Bodies of the distributions of loss rates with per-byte probabilities for the five traces.

different when the probability is computed in terms of bytes rather than in terms of connections, as shown in Figure 4.20. Most bytes were carried in connections that experience at least some loss. We can also observe that Abilene-I's connections were substantially less affected by loss than the connections in the other traces, since Abilene-I's CDF shows higher cumulative probabilities. For example, the distribution for the Abilene-I trace shows that only 8% of the bytes were carried in connections with 1% loss or more. The distributions for the rest of the traces show that between 20% and 34% of the bytes were carried in connections with 1% loss or more. Loss is specially significant at UNC, where 34% of the bytes were carried by connections with loss rates above 1% (a high loss rate). Interestingly, the UNC trace with the highest load (UNC 1 PM) had a lighter distribution of per-byte loss rates.

4.2 Network-level Metrics

The previous section considered methods for characterizing network-level properties of traffic that can be incorporated into traffic generators as input parameters. Here we consider other network-level properties that can be used to compare traces, providing a way to assess the realism of synthetic traffic. In order to make the distinction between these two types of network-level properties clearer, we apply the term *network-level parameter* to those properties that are part of the input of the traffic generation method, and the term *network-level metric* to those properties that are not part of the input but are still useful for characterizing the output (*i.e.*, the synthetic traffic). The key idea, demonstrated in Chapter 6, is that synthetic traffic can closely approximate real traffic in terms of these network-level metrics, as long as source-level and network-level parameters are incorporated into the traffic generation method.

The success of this approach confirms that the parameters we have incorporated in our approach are significant, and that the data acquisition methods we propose are sufficiently accurate to achieve high realism in traffic generation.

4.2.1 Aggregate Throughput Time Series

A basic property of the performance of a network link is the number of bytes and packets[8] that traverse the link per unit time. We will call this property aggregate throughput, since it is the result of multiplexing the throughputs of the individual connections that form the traffic carried by a network link. Accurately reproducing aggregate throughput will be an important part of our evaluation.

Aggregate throughput is generally very variable, so researchers (and practitioners) usually study the time series of aggregate throughputs in order to understand the dynamics of network traffic. Formally, an aggregate throughput time series at scale t is defined as a vector $X^t = (X_1^t, X_2^t, \ldots, X_n^t)$ where X_i^t is the number of bytes (or packets) observed at a measurement point between time $t(i-1)$ and time ti for some constant interval t. This constant integral t is called the *scale* of the time series. X_i is often referred to as the i-th bin of the time series, which is sometimes called a time series of bin counts.

We consider three ways of studying aggregate throughput time series in this dissertation. First, we make use of plots of aggregate throughput against time, "throughput plots", which provide a simple yet informative visualization of the dynamics of the traffic throughout the entire trace. Second, we examine the marginal distribution of the time series using a CDF, which enables us to study the fine scale characteristics of the throughput process. These two methods are described in more detail below. While they are useful, they are sensitive to the scale t at which the time series is analyzed (*e.g.*, throughput per minute is much smoother than throughput per millisecond). For this reason, we complement our analysis with a third method, wavelet analysis. Wavelet analysis is a multi-resolution method particularly suitable to study how the statistical nature of X^t changes as a function of t. This type of study, often known as "traffic scaling", is specially important for Internet traffic, which exhibits strong long-range dependence. We employ both plots of the wavelet spectrum of a throughput time series, and wavelet-based estimates of Hurst parameters with confidence intervals.

When it comes time to validate synthetic traffic generation methods, an important aspect of the

[8]In this section, we will often use the term packet rather than segment. In the context of TCP traffic, a time series of packets per unit time and a time series of segments per unit time are the same thing. However, the traffic measurement literature generally talks about packet throughput (not segment throughput), often using the unit Kilo packet per second (Kpps).

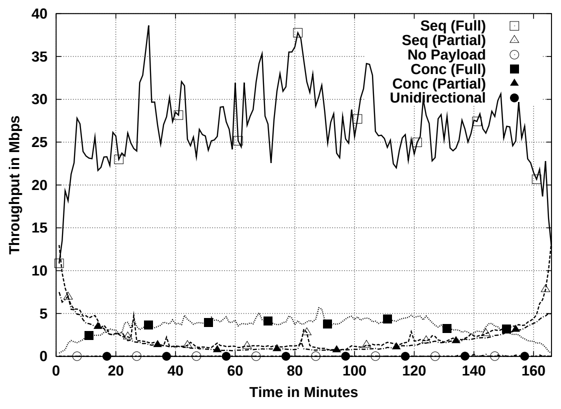


Figure 4.21: Breakdown of the byte throughput time series for Leipzig-II inbound.

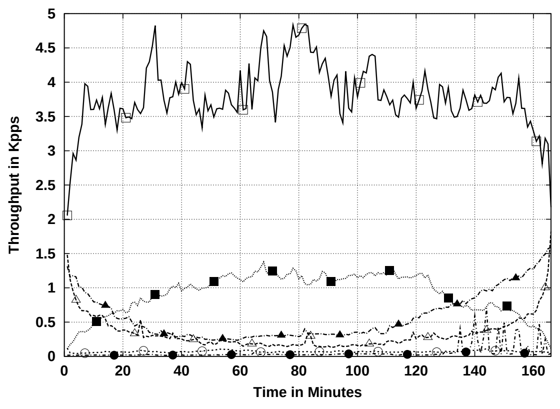


Figure 4.22: Breakdown of the packet throughput time series for Leipzig-II inbound.

validation will be a qualitative comparison of plots of throughput time series and plots of their marginal distributions and wavelet spectra. Here we describe the use of these visualizations to understand the nature of throughput on the links we have measured.

Throughput Plots

Figure 4.21 shows a breakdown of the aggregate byte throughput of the Leipzig-II trace in the inbound direction (*i.e.*, TCP traffic coming into the University of Leipzig). The scale of the time series (the bin size) is one minute. The time series of all byte arrivals has been partitioned into six time series according to the type of abstract source-level behavior (sequential, concurrent or no payload[9]), depending on whether the start and the end of the connection were observed (fully or partially captured connections), and whether the connection was observed only in one direction of the link (unidirectional connections) or in both. The analysis of abstract source-level behavior described in Section 3.5 was used to classify connections into these categories, and then the original segment header traces were partitioned according to this classification. This type of analysis complements the one performed at the source-level in Section 3.5, giving us a sense of the relative importance of sequential and concurrent connections. Also, our traffic generation will only make use of connections that were fully captured, *i.e.*, fully characterized, so it is important to understand the importance of the traffic in the rest of the connections (so we know what we are missing).

Figure 4.21 shows that sequential connections that were fully captured account for the vast majority of the bytes to Leipzig-II inbound. Since connections observed near the boundaries of the trace are more

[9]A connection without any useful TCP payloads has an empty connection vector since no ADU is sent.

likely to be observed only partially, the time series shows a much smaller number of bytes in the first and in the last ten minutes of the trace. On the contrary, the time series of partially-captured sequential connections has a much larger number of bytes in the first and the last ten minutes. This is because the probability of observing only part of a connection increases as we get closer to the trace boundaries. For this reason, in the first ten minutes we see many more connections that started before the start of the trace, and in the last ten minutes we see many more connections that ended after the end of the trace. We will refer to this increased likelihood of finding partially-captured connections near trace boundaries as the *connection sampling bias*.

The solid line with white squares in Figure 4.21 shows the time series of fully-captured sequential connection. When we examine the stable region of this time series (*i.e.*, ignoring the first and last 10 minutes), we can see substantial variability between the minimum of 22 Mbps and the maximum of 38 Mbps. The rest of the time series in this plot are far less “bursty”. The average throughput of the time series for concurrent connections is much smaller, and partially-captured connections only account for a tiny fraction of the bytes. The number of bytes in connections without any useful data payload is insignificant, as one would expect in a properly working network in which little malicious activity is taking place[10].

Figure 4.22 shows the time series of packet arrivals for the inbound direction of the Leipzig-II trace. As in the previous figure, fully-captured sequential connections account for the majority of the packets, and the time series exhibits substantial variability. Notice however that the number of packets in fully-captured concurrent connections is more significant in terms of packets than in terms of bytes (the percentage of packets was higher than the percentage of bytes). The time series of the number of packets in “no payload” connections and in unidirectional connections is also more significant. Notice the large spikes at the end of the time series of unidirectional connections. These spikes could be related to some malicious activity, like network or port scanning[11], or connection attempts to a popular server that is temporarily offline[12]. This feature was not present in the corresponding time series of byte arrivals.

The same time series for the reverse direction of the Leipzig link are shown in Figures 4.23 and 4.24. The magnitude of the throughput time series is significantly smaller in this case, suggesting that the

[10]The “no payload” time series would have been much more significant if, for example, a denial-of-service attack using SYN segments had taken place. These segments, and the likely SYN-ACK segments sent in response by the victim, would have not carried any (useful) payloads (no application-level communication would have taken place), and would have been classified as “no payload” traffic.

[11]This type of activity creates unidirectional traffic whenever the target host is firewalled, or otherwise unreachable, or the target IP does not exist. The location of these spikes at the end of the trace is purely accidental.

[12]In this case, clients would try to open a connection by sending a SYN segment (and several retransmissions), which will receive no response since the destination server is not online. These types of connection attempts to offline hosts show up as unidirectional connections in segment header traces.

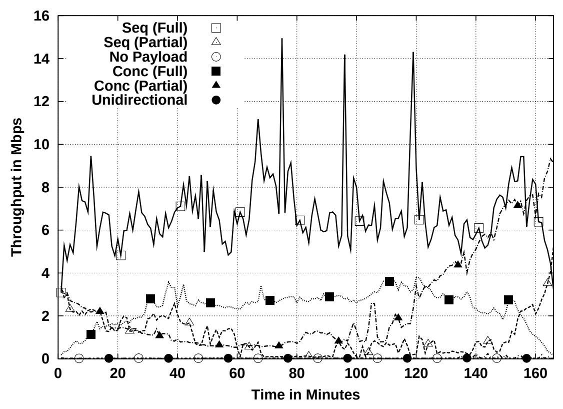


Figure 4.23: Breakdown of the byte throughput time series for Leipzig-II outbound.

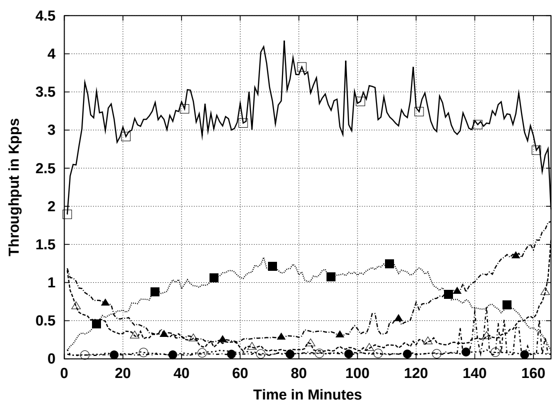


Figure 4.24: Breakdown of the packet throughput time series for Leipzig-II outbound.

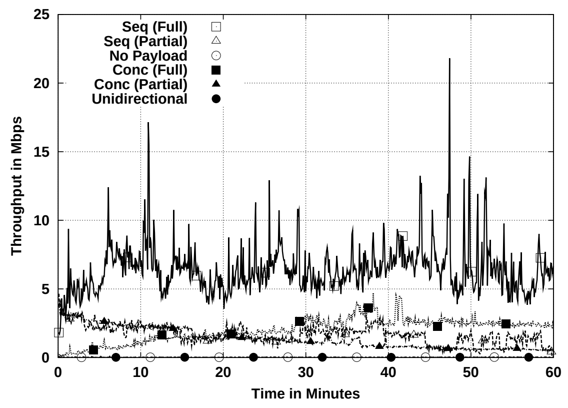


Figure 4.25: Breakdown of the byte throughput time series for Leipzig-II outbound.

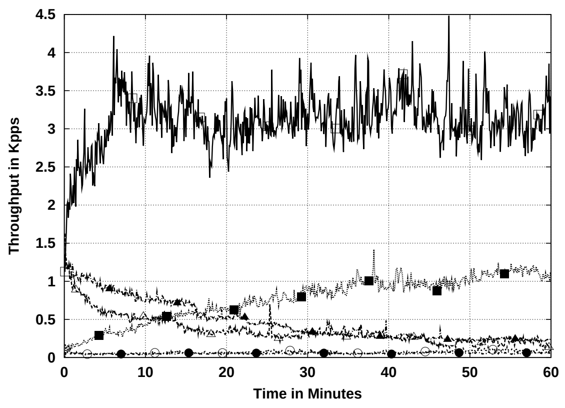


Figure 4.26: Breakdown of the packet throughput time series for Leipzig-II outbound.

University of Leipzig is mostly a consumer of content from the rest of the Internet. Sequential connections that were fully captured exhibit some sharp spikes in three short time intervals. A closer look revealed that only a few large connections with small round-trip times (in particular, three connections in the first spike in minute 78) created this sudden increase in throughput. As in the inbound direction, partially-captured sequential connections are only significant for the first and the last few minutes of the trace. Concurrent connections also show the same pattern, but partially-captured connections exhibit a steady increase for the last 50 minutes of the trace. Interestingly, the separation between the time series for fully and partially captured sequential connections is much larger for packets than for bytes. This suggests that the packets in this direction are very small, and mostly consist of acknowledgment segments that do not have a payload. This is another confirmation of the content-consumer nature of the university of Leipzig.

Figures 4.25 and 4.26 illustrate the impact of scale on the throughput time series for the Leipzig-

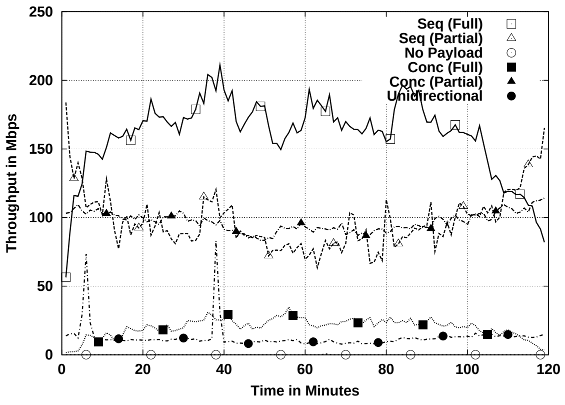


Figure 4.27: Breakdown of the byte throughput time series for Abilene-I Ipls/Clev.

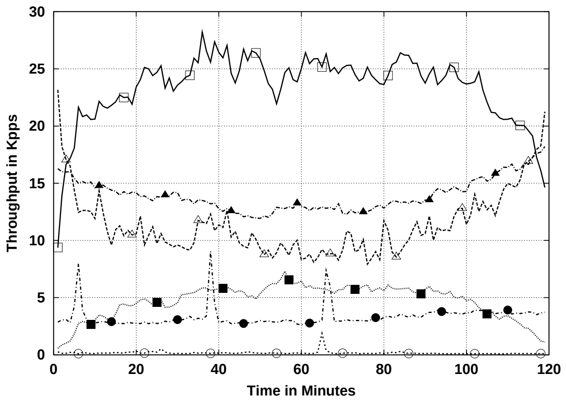


Figure 4.28: Breakdown of the packet throughput time series for Abilene-I Ipls/Clev.

II outbound trace. These plots have a scale of 5 seconds, and only the first 60 minutes (rather than entire 166 minutes) are shown to reduce the amount of over-plotting. Both byte and packet throughputs are clearly more bursty at this scale. The largest spikes of time series for fully-captured sequential connections are even larger (and therefore narrower) than those in the 1-minute time series. For example, the spike in the eleventh minute reaches 17 Mbps in the 5-second scale, while the corresponding region in the 1-minute scale plot in Figure 4.23 did not go above 9 Mbps. Decreasing the scale provides a better picture of the burstiness of traffic, but it increases over-plotting, and does not change the overall view (*i.e.*, the relative magnitude of the different time series). The same lesson holds for packet arrivals, but notice that the largest byte throughput spikes do not appear to have corresponding packet throughput spikes. This shows that a relatively small number of full packets sent in short periods created the observed throughput spikes (and not a large number of small packets).

The structure of the throughput time series for the Abilene-I trace is remarkably different. Figure 4.27 shows the time series of byte arrivals at a 1-minute scale for the Abilene-I traffic sent from Indianapolis to Cleveland. As in the Leipzig-II case, fully-captured sequential connections account for the largest percentage of the traffic. However, bytes from partially-captured sequential connections are much more significant here, with a mean throughput that is roughly half of the mean throughput for fully-captured sequential connections. While we still observe much larger throughputs in the first and last few minutes of the time series, the middle part still accounts for a very large number of bytes. This is in sharp contrast to the Leipzig-II trace, and cannot be explained by the duration of the trace, which is almost as long (2 hours *vs.* 2 hours and 46 minutes). Note also that the partially-captured connection time series is almost as bursty as the time series for fully-captured connections.

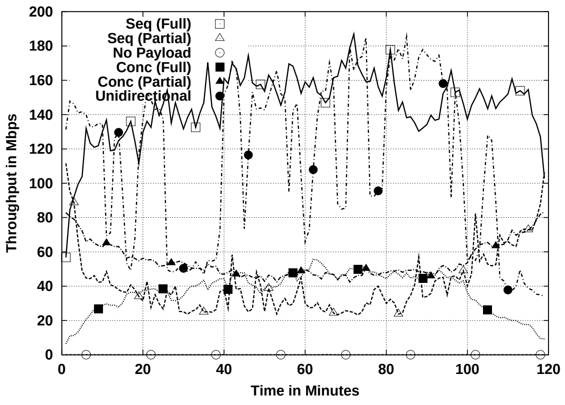


Figure 4.29: Breakdown of the byte throughput time series for Abilene-I Clev/Ipls.

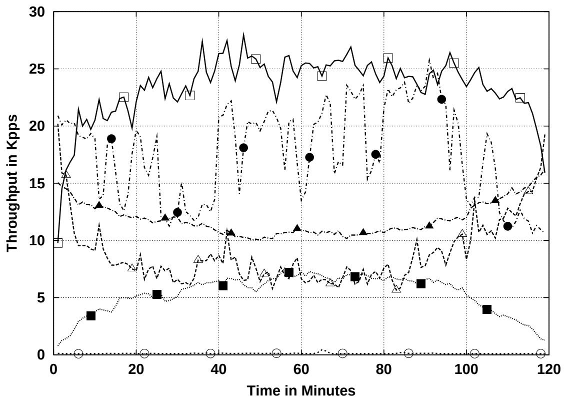


Figure 4.30: Breakdown of the packet throughput time series for Abilene-I Clev/Ipls.

Concurrent connections in this direction of the Abilene-I trace show a surprising structure. The number of bytes in concurrent connections that were partially-captured was much larger than the number of bytes in connections that were fully-captured. This suggests that concurrent connections in this trace tend to have extremely long durations. Both time series are much smoother than those for sequential connections, and trace boundaries have very little impact on them. Connections with no payload carried an insignificant number of bytes, but, unlike the Leipzig-II trace, unidirectional traffic is non-negligible. Rather than some malfunction or malicious activity, this is explained by asymmetric routing in the Abilene backbone. Only one direction of these connections goes through the measured link, and hence these connections appear in our trace as unidirectional. We also observe two major throughput spikes at the 6th and the 38th minutes that could also be explained by transient routing changes, but malicious traffic cannot be ruled out without further analysis. Both spikes reach throughputs as high as 350 Mbps when the time series is examined at the 5-second scale.

The packet throughput time series for the Abilene-I trace shown in Figure 4.28 has a similar structure, in which partially-captured connections also account for a large percentage of the trace. It is interesting to note that fully-captured concurrent connections carry a larger percentage of packets than bytes, so packets in these connections are likely to be small. We also observe a third spike in the time series for unidirectional connections that did not show up in the byte throughput time series, and a smaller spike in the "no payload" time series.

The reverse direction, Cleveland to Indianapolis, of the Abilene-I trace offers a rather different view in Figure 4.29. Partially-captured sequential connections are much less significant in this case, although this time series still exhibits remarkable variability. Similarly, the number of bytes in partially-

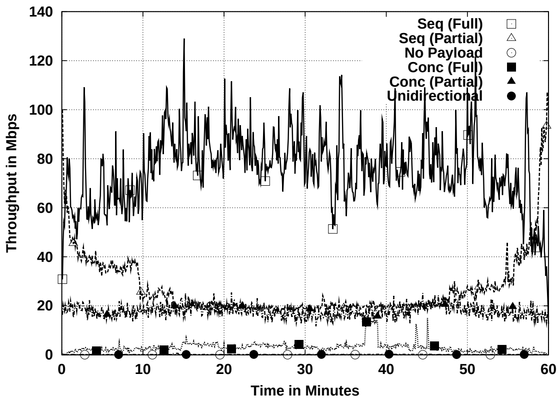


Figure 4.31: Breakdown of the byte throughput time series for UNC 1 PM inbound.

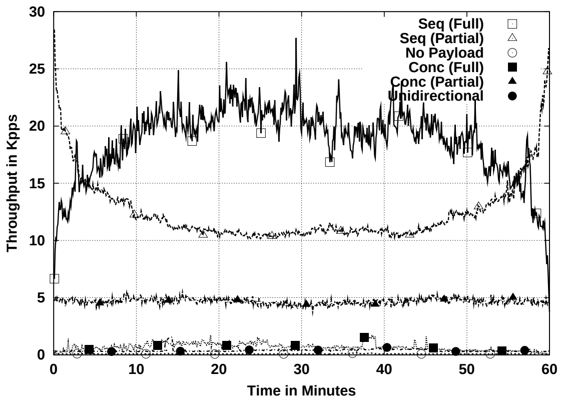


Figure 4.32: Breakdown of the packet throughput time series for UNC 1 PM inbound.

captured concurrent connections is much lower in relative terms, and quite close to the number of bytes in fully-captured concurrent connections. The most striking feature of this plot is the time series of unidirectional connections. The byte throughput of these connections shows enormous variability, and even reaches the magnitude of fully-captured sequential connections. This is either a strong indication of substantial instability in the routing of the Abilene backbone, or the existence of flows with extremely high throughput that only show up in one direction of the measured link. In any case, byte throughput is always above 50 Mbps, so part of the traffic that is routed asymmetrically did not experience any major routing changes.

The packet throughput time series from Cleveland to Indianapolis shown in Figure 4.30 offer yet another pattern in the breakdown of traffic per connection type. The sharp changes in the throughput of the time series of unidirectional traffic have a smaller magnitude, suggesting that large packets dominate traffic in this direction of the Abilene-I trace. Partially-captured concurrent connections carried significantly more packets than fully-captured connections. This is the opposite of the phenomenon in Figure 4.28 and can be explained by an asymmetry in the sizes of the ADUs of the connections. This asymmetry results from connections with a majority of data segments in the same direction and a majority of acknowledgments in the other direction.

The byte throughput time series for the UNC 1 PM trace in the inbound direction resembles that of Leipzig-II (which is also an edge trace). Figure 4.31 shows that fully-captured sequential connections carry the vast majority of the bytes, although the relative percentage of bytes in partially captured connections is larger. This is can be explained by the shorter duration of this trace (1 hour). The most significant difference, however, is in the time series for partially-captured concurrent connections.

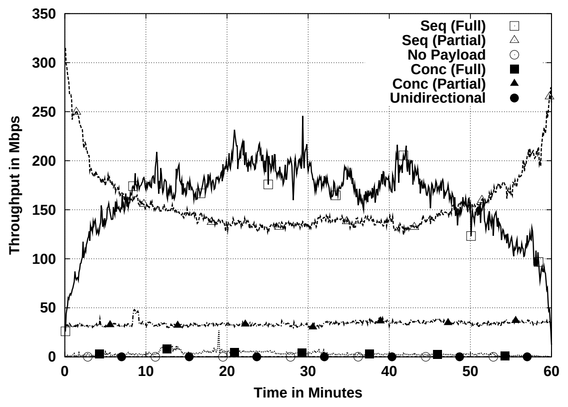


Figure 4.33: Breakdown of the byte throughput time series for UNC 1 PM outbound.

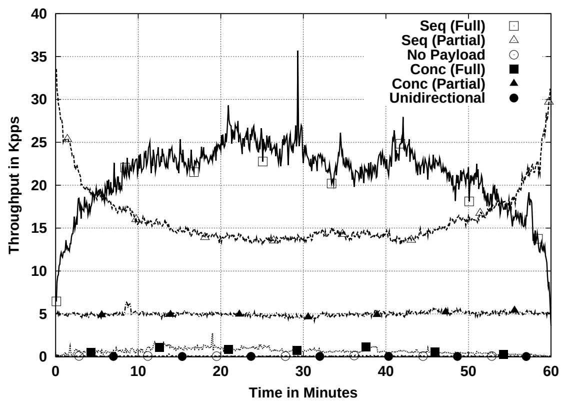


Figure 4.34: Breakdown of the packet throughput time series for UNC 1 PM outbound.

In this case, we find a very stable throughput of 20 Mbps without any clear boundary effects. This is similar to the type of concurrent traffic found in Abilene-I. Fully-captured concurrent connections show an interesting jump between the 37th and the 39th minutes, and a couple of spikes around the 45th minute. This could be explained by a single connection with significant throughput (10 Mbps). Packet throughput time series are similar, but we observe a significantly higher percentage of packets in partially-captured sequential connections. As the analysis of the other direction will suggest, this is due to the presence of many pure acknowledgment segments.

The byte throughput time series for the outbound direction of UNC 1 PM are shown in Figure 4.33. They are remarkably different from those of the Leipzig-II trace, where throughput on the inbound direction (created by local users downloading content from the Internet) was much higher than the throughput in the outbound direction. We observe the opposite here. The mean overall utilization in the outbound direction is much higher than the inbound direction, 325 Mbps versus 100 Mbps. Also, partially-captured sequential connections are much more significant. The obvious explanation is the presence at UNC of `ibiblio.org`, a popular repository of software and other content. Hosts outside UNC retrieve large amounts of data from the `ibiblio.org` servers, making the load in the outbound direction of the UNC link much higher than the load on the inbound link. Furthermore, `ibiblio.org` clients often download large objects, and this requires long connections that are more likely to be only partially captured. This provides a good explanation for the extreme boundary effects in the first and last 10 minutes of the throughput time series. The high throughput in the stable region of this time series could be due to long connections that carry large amounts of data, although further analysis is needed to verify this claim.

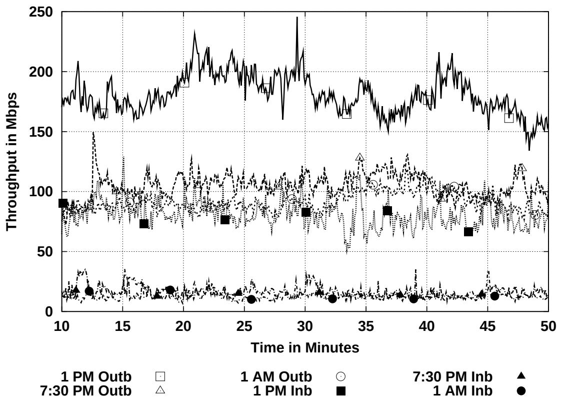


Figure 4.35: Breakdown of the byte throughput time series for the three UNC traces.

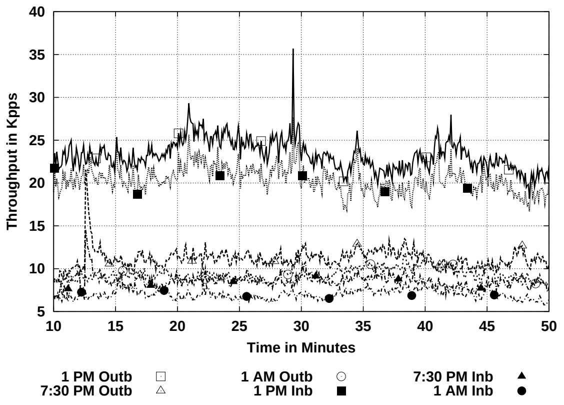


Figure 4.36: Breakdown of the packet throughput time series for the three UNC traces.

Concurrent traffic in the outbound direction appears similar to the inbound direction, showing remarkable load symmetry for partially-captured concurrent connections. We do not observe much variation in the time series for packet throughput (shown in Figure 4.34). Partially-captured sequential connections carried a smaller number of packets than bytes, and this agrees with the idea that large numbers of bytes are downloaded from `ibiblio.org`. These downloads show up as large data packets in the outbound direction and small acknowledgment packets in the inbound direction. In the most common case, a full TCP segment has a size of 1500 bytes, while an empty one (no payload) is only 40 bytes. This means that the ratio of bytes in a connection carrying a large file is 1500:40. Furthermore, since most TCP implementations acknowledge only every other data segment, we have a ratio of 3000:40 for bytes and a ratio of 2:1 for packets. A link that is dominated by large file downloads should show similar byte and packet ratios. If large file downloads from `ibiblio.org` were the only cause of the large fraction of bytes in partially captured connection, then we would expect similar ratios between the two directions of the UNC link. However, this is not so clear in Figures 4.31 to 4.34, suggesting that phenomena other than `ibiblio.org` also contribute to making UNC a source rather than a sink of content. As an example, file-sharing activity from campus dorms could also play a significant role. Since file-sharing implies both uploading and downloading, it usually tends to make the byte and packet ratios more balanced.

Network activity usually follows a cyclic daily pattern, in which traffic increases throughout the morning and decreases in the evening, being at its lowest during night hours. This diurnal pattern in the time series for the UNC traces is evident in Figures 4.35 and 4.36. These time series correspond to fully-captured sequential connections. Byte throughputs for the 1 PM traces are much higher in both directions and we observe that even the 1 AM trace has a large throughput in the outbound direction.

This suggests that content from UNC is downloaded throughout the day, although a diurnal pattern is still present. On the contrary, UNC clients are much less active later in the day. A similar plot (not shown) for partially-captured concurrent connections shows little reduction in throughput between the 1 PM and the 7:30 PM traces, and a reduction of only 15 Mbps in the 1 AM trace. The packet throughput time series illustrate again the dichotomy between the large data segments in the outbound direction, carrying UNC content, and the small segments in the inbound direction, carrying acknowledgments. The difference is substantially less significant later in the day.

4.2.2 Throughput Marginals

Plots of throughput time series provide a good overview of the coarse-scale characteristics of the traffic trace. However, they are not very practical for studying finer-scale features. Our traces are long enough that any plot at a scale of 1 second or below is dominated by over-plotting, and does not provide any useful information. This is specially true when the goal of the plot is to compare two time series that are already rather similar. Finer-scale differences can be of great importance for certain experiments. For example, two traces could have exactly the same average throughput, and appear identical at the 1-minute scale, but be completely different at the 1-second scale. One of them could show a sequence of sharp spikes and ditches, while the other one could remain completely smooth. If we were to expose a router queue to these two traces, we could obtain two completely different distributions of router queue length (and therefore of packet delay through the router). This would for example be the case if the spikes exceed the output rate of the queue (creating a backlog), while the smooth trace always remains below output rate. In the first case, packets would experience variable queuing delay, while in the second case no queuing delay would occur.

There are several ways in which we can compare traces at finer time scales. The most obvious one is to examine throughput for a limited period. While this approach is useful in some situations, it does not scale for comparing entire traces, especially as we decrease the scale and the number of possible periods to examine grows. In this dissertation, we will make use of two alternative methods for studying the throughput of our traces at finer granularities. Our first method is to examine the marginal distributions of throughput time series at a rather fine time scale, 10 milliseconds. Plots of the bodies of marginal distributions help us to understand the most common fine-scale throughputs in a trace, while plots of the tails of marginal distributions explore the episodes of highest throughput in a trace. We describe this type of analysis in the rest of this section. Our second method is to study the way throughput

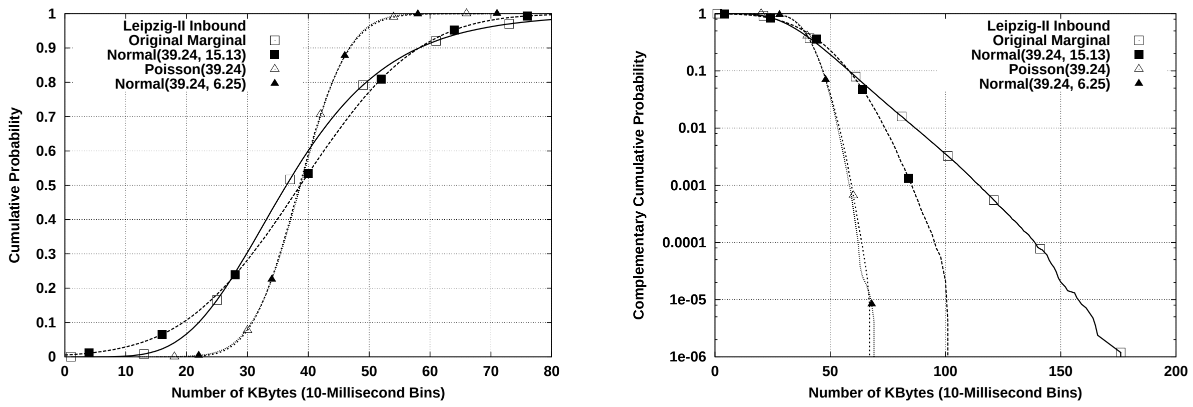


Figure 4.37: Byte throughput marginals of Leipzig-II inbound, its normal distribution fit, the marginal distribution of its Poisson arrival fit, and the normal distribution fit of this Poisson arrival fit.

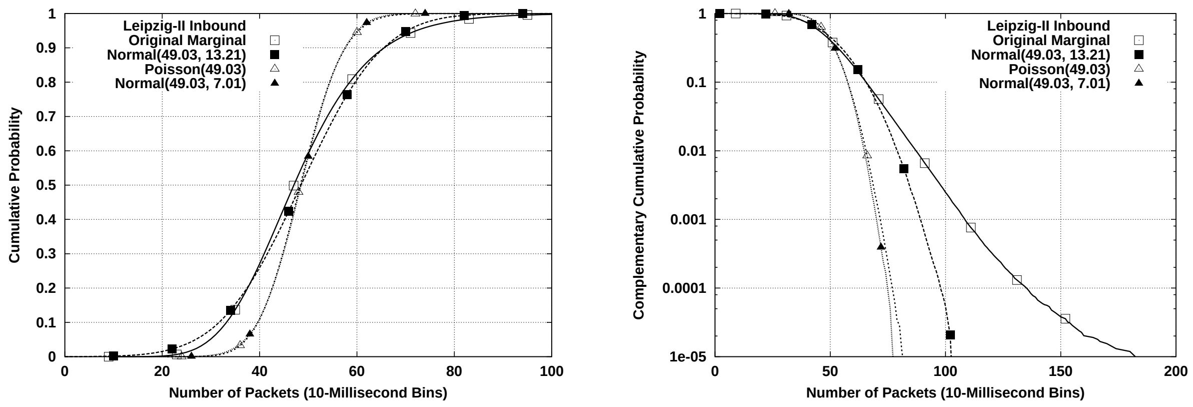


Figure 4.38: Packet throughput marginals of Leipzig-II inbound, its normal distribution fit, the marginal distribution of its Poisson arrival fit, and the normal distribution fit of this Poisson arrival fit.

variance changes with scale, which we will approach using the concepts of self-similarity and long-range dependence. We discuss this type of analysis in the next section.

Our analysis of throughput marginals examines the time series of throughput at the 10-millisecond time-scale, constructing the empirical distribution of the values of the time series. This empirical distribution assigns a certain probability to each observed value of the time series equal to the fraction that this value represents of the total set of values in the time series. As in previous cases, we will study the bodies of the marginal distributions using plots of CDFs and the tails using plots of CCDFs. For example, Figures 4.37 and 4.38 show respectively the marginal distribution of byte and packet throughput for the inbound direction of the Leipzig-II trace (depicted using solid lines marked by white squares). The CDF in the left plot provides a good overview of the body of the marginal distribution using linear axes. The CCDF in the right plot shows the tail of the distribution using a logarithmic y-axis. These

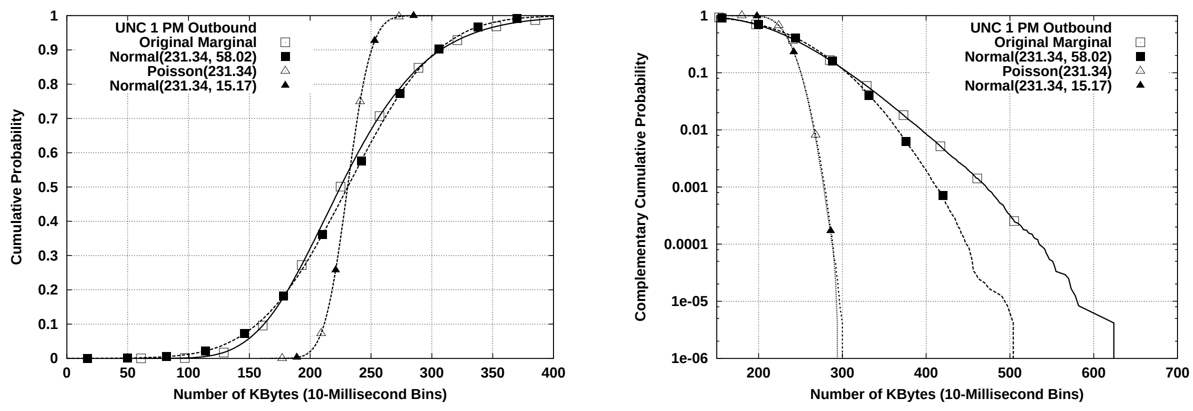


Figure 4.39: Byte throughput marginals of UNC 1 PM outbound, its normal distribution fit, the marginal distribution of its Poisson arrival fit, and the normal distribution fit of this Poisson arrival fit.

visualizations of the marginals are particularly useful for comparing multiple distributions, and we will use them extensively in Chapter 6.

As stated before, our goal with the analysis of marginal distributions is to understand fine-scale characteristics of throughput. We will use this type of analysis to determine whether our proposed traffic generation method results in synthetic traffic whose distribution of fine-scale throughputs is "realistic". By construction, and as explained in Chapter 5, the determination of this realism is accomplished by directly comparing the marginal distributions of an original trace and its synthetic version. This non-parametric analysis is consistent with other methods used in this dissertation.

We have also considered doing some parametric analysis of the marginal distributions of throughput time series. When modeling an arrival process, the first approach that comes to mind is the Poisson modeling framework. Poisson arrivals are very convenient from an analytical perspective, and concisely describe an arrival process using a single parameter. As pointed out by Floyd and Paxson [PF95], empirical studies do not support the use of this model, primarily because Poisson arrivals are far less "bursty" than Internet packet and byte arrivals. This important issue is discussed in the next section. In addition, we show here that Poisson arrivals have marginal distributions that are very far from the ones in our traces.

Given a throughput time series, we can fit a Poisson arrival model simply by computing the mean of the time series and using it as the rate of the Poisson model. From this fitted model, we can easily obtain a marginal distribution using Monte Carlo simulation. Figure 4.37 shows the marginal distribution of byte throughput in the inbound direction of the Leipzig-II trace, and the marginal distribution of the Poisson fit of this throughput process (depicted using a dashed line with white triangles). Both marginal

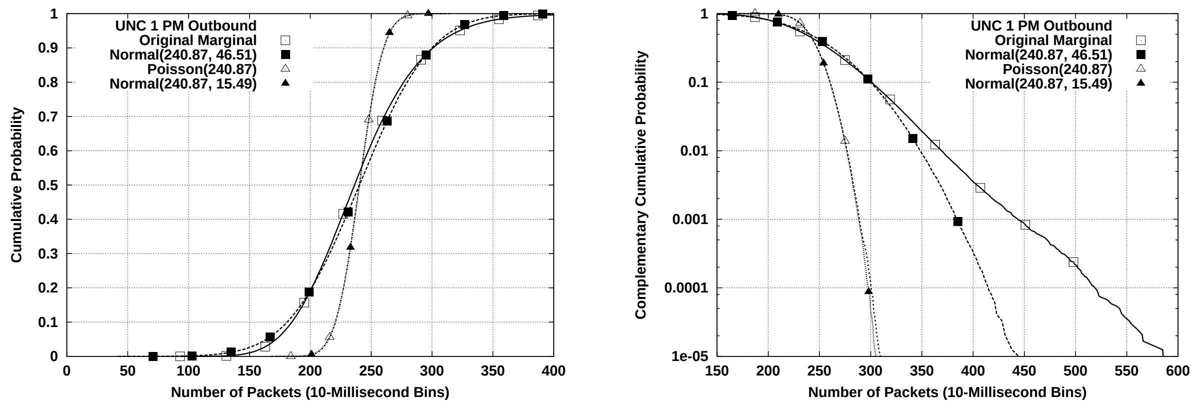


Figure 4.40: Packet throughput marginals of UNC 1 PM outbound, its normal distribution fit, the marginal distribution of its Poisson arrival fit, and the normal distribution fit of this Poisson arrival fit.

distributions have the same mean, 39.24 Kilobytes per 10-millisecond interval. As shown in the figure, the two marginals are very different, with the Poisson fit exhibiting a far narrower body. The standard deviation of the Poisson model is only 6.25, while the one for the real marginal distribution is 15.13, more than twice as large. In addition, the tail of the marginal distribution from Poisson arrivals is far lighter than the one from the trace. Intuitively, this means that the real traffic is far more aggressive on the network, consistently reaching far higher throughput values. Poisson arrivals are equally inadequate for modeling packet arrivals, at least in terms of their marginal distributions, as shown in Figure 4.38. Figures 4.39 and 4.40 repeat the same analysis for the outbound direction of UNC 1 PM. The plots confirm the poor fit from the Poisson arrival model, even for a trace with a throughput that is eight times higher. The same is true for every other trace examined in this dissertation.

The empirical results in Fraleigh *et al.* [FTD03] and the analysis in Appenzeller *et al.* [AKM04] support the idea that throughput values are normally distributed in Internet traffic, as long as sufficient traffic aggregation exists. If this were true, studying (and comparing) marginal distributions of throughput could easily be accomplished by looking at means and variances. Our analysis of the throughput marginals in our traces shows that they do resemble a normal distribution, but that this model is not completely satisfactory.

We can easily fit a normal model to the marginal distributions from our traces by computing their means and standard deviations. Figures 4.37, 4.38, 4.39 and 4.40 compare the real marginals and their fits, consistently showing the following two differences:

- The bodies of the marginal distributions from the real traces appear somewhat narrower. The

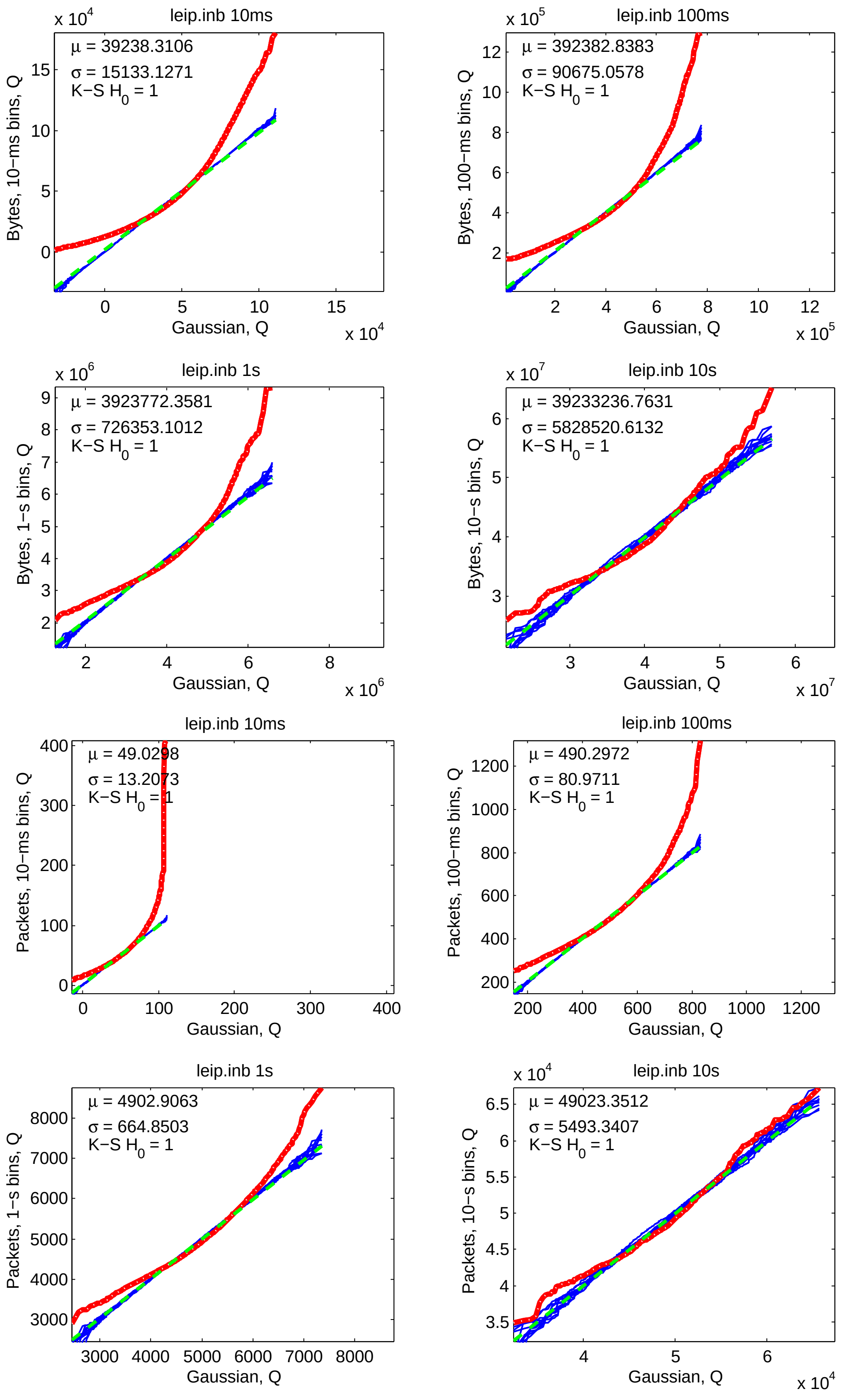


Figure 4.41: Quantile-quantile plots with simulation envelops for the marginal distribution of Leipzig-II inbound. The top four plots show byte throughput, while the four bottom plots show packet throughput.

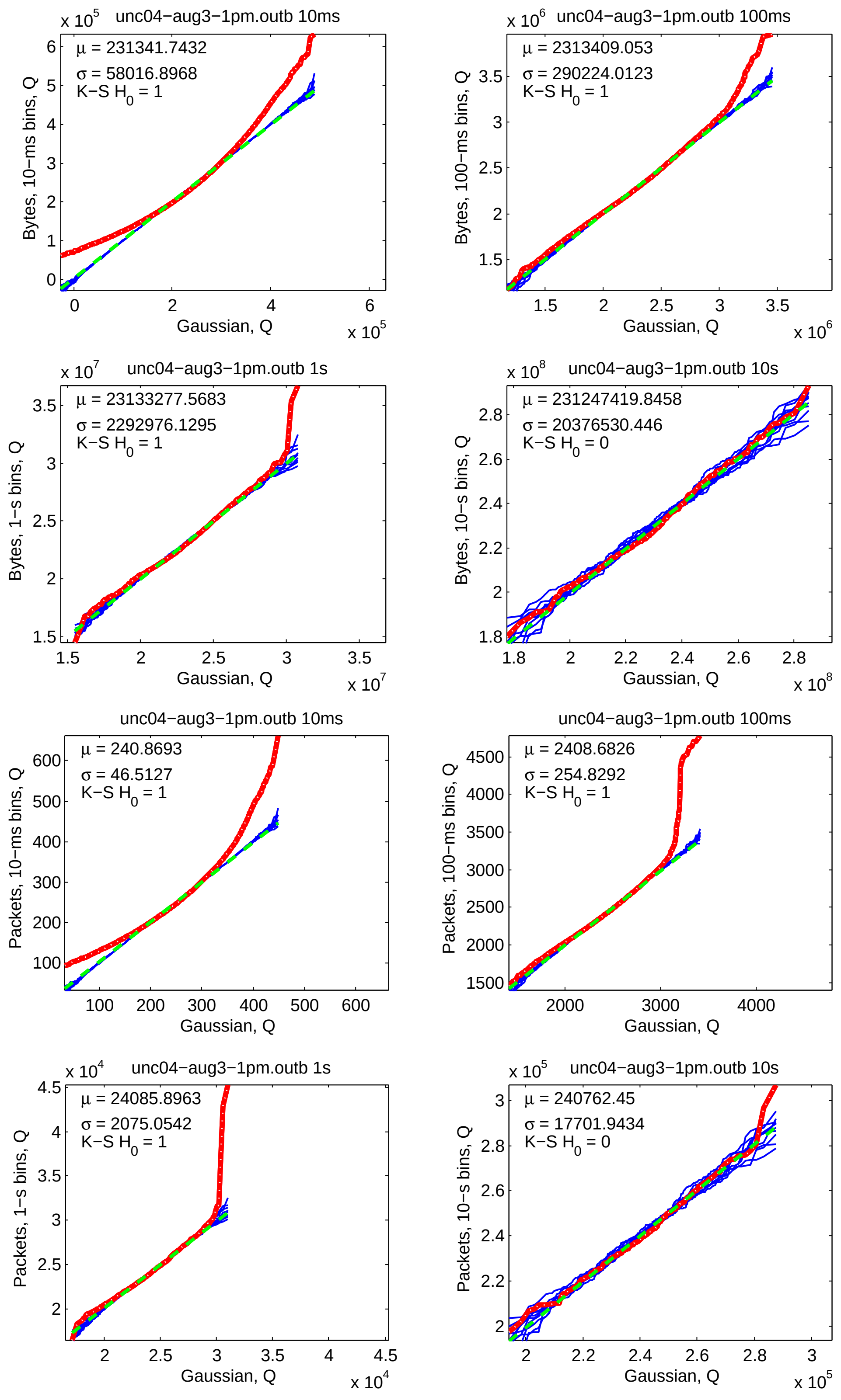


Figure 4.42: Quantile-quantile plots with simulation envelops for the marginal distribution of UNC 1 PM outbound. The top four plots show byte throughput, while the four bottom plots show packet throughput.

difference is slightly larger for packet throughput in the Leipzig-II trace.

- The tails of the marginal distributions from the real traces are substantially heavier. The largest values for the Leipzig-II traces are 75% larger, while those for the UNC 1 PM trace are 20-25% larger.

These deviations are present in every one of our traces, showing that throughput marginal distributions deviate from the normal distribution systematically.

The deviation from normality of the empirical marginal distributions is statistically significant. First, every marginal distribution from the traces fails the Kolmogorov-Smirnov test of normality [NIS06]. Second, every Quantile-Quantile (Q-Q) plot [NIS06] shows a clear departure from normality. This is true not only for the 10-millisecond time-scale, but also for the 100-millisecond, the 1-second time series, and even for the 10-second time series in some cases. We illustrate this type of analysis again using the throughput marginals of Leipzig-II inbound and UNC 1 PM outbound. The plots in Figures 4.41 and 4.42 show Q-Q plots for different time-scales, where the quantiles of the data and the theoretical normal distribution are compared using a thick line with white dots. If the data were normally distributed, the Q-Q line would closely follow the the dashed 45 degree line. This is clearly not the case, but the Q-Q plot does not provide any sense of statistical significance. To address this deficiency, the plots also show simulation envelopes, depicted using thin, dark-gray lines, following the methodology in Hernández-Campos *et al.* [HCMSS04]. They are easiest to see in the 10-second time series, the ones with the least over-plotting. Each line in the envelope corresponds to a distribution constructed by sampling the theoretical normal distribution as many times as values were present in the empirical marginal distribution. The envelope therefore captures the natural variability of the normal distribution for the given sample size. If the Q-Q line comparing the empirical marginal and the theoretical normal distribution is outside this envelope, the deviation from normality is considered statistically significant. This is clearly the case for every marginal distribution in the plots, except the ones at the 10-second scale of Leipzig-II inbound.

The plots in Figures 4.41 and 4.42 also show the results of the Kolmogorov-Smirnov test (K-S), a formal test of normality. The third line in the inside legend, below the sample mean μ and the standard deviation ρ, shows the result of the formal test. The null hypothesis (non-normality) can only be rejected for the marginals of the Leipzig-II throughput at the 10-second time-scale. The plots show a $H_0 = 0$ when the null hypothesis can be rejected, and a $H_0 = 1$ when it cannot be rejected. Given these results, assuming normality to study the marginals of our traces (and those of their synthetic versions) is of dubious value. We will restrict ourselves to comparative plots of CDFs and CCDFs for comparing throughput marginals.

Note that we are not arguing that our finding of pervasive deviations from normality invalidates earlier studies based on the assumption of normality in throughput marginals. From our analysis, the bodies of the marginals are close enough to the normal distribution that assuming normality can provide a useful simplification. As long as significant deviations from normality in the tails have little or no effect on the reasoning, assuming normality makes analytical studies more treatable and even more intuitive.

Our finding of non-normality in our traces is consistent with the observation by Sarvotham *et al.* [SRB01]. These authors demonstrated that deviations of the throughput marginal from normality can be explained by the presence of an *alpha* component in Internet traffic. Alpha traffic is composed of connection with high throughputs that transfer large amounts of data. In contrast, connections with moderate or low throughputs and connections with moderate or small amounts of data to transfer are considered *beta* traffic, whose throughput marginal is normally distributed. Intuitively, a traffic generation method should be able to reproduce both the alpha and the beta components of Internet traffic. Sarvotham *et al.* also proposed to consider traffic bursty when its throughput marginal deviates from normality. This is an alternative (and complementary) view of traffic burstiness, which is more commonly associated with long-range dependence in the arrival process, as we will discuss next.

4.2.3 Throughput Self-Similarity and Long-Range Dependence

A remarkable characteristic of Internet traffic is its high variability in throughput across a wide range of time scales, and how that variability changes as scale increases. If we plot the number of packets or bytes that arrive at a network link, say every 1 or 10 milliseconds, we observe a highly variable process where the number of arrivals is constantly changing. If we plot these arrivals at a coarser scale, say every 100 milliseconds or 1 second, this high variability does not decrease significantly. In contrast, Poisson arrivals exhibit a rapid decrease in variability as we increase the scale of the time series. For this reason, it is often said that Internet traffic has a “very bursty” arrival process, far more variable than that of call arrivals in a phone network. Starting with the work of Leland *et al.* [LTWW93], traffic burstiness has usually been characterized using the theoretical framework of statistically self-similar processes. This framework provides some powerful methods to study traffic burstiness and quantify its strength.

The motivation behind the study of traffic burstiness is the observation that an increase in the burstiness of traffic results in a more demanding network workload. For example, Erramilli *et al.* [ENW96] demonstrated that router queues exhibit dramatically heavier distributions of queue lengths as the burstiness of the input packet arrival process increases. Numerous measurement studies, *e.g.*, [WTSW97, ZRMD03, PHCMS05, PHCL$^+$], have examined Internet traffic and consistently observed highly bursty arrivals that appear self-similar for scales between a few milliseconds and tens of seconds. It is therefore expected that representative synthetic traffic reproduces this high burstiness. In this dissertation, we employ well-known methods to assess the self-similarity of real and synthetic traffic, and verify that our traffic generation methods can reproduce the level of burstiness in Internet traffic.

The term self-similarity comes from the study of fractal objects. Fractals are geometrical constructs that appear similar a different scales. The most famous example of fractal is the Mandelbrot set, whose cardioid shape repeats itself as we zoom into the set. In this fashion, a second-order self-similar time series shows a similar pattern of variation at different time-scales. For this reason, self-similarity is also known as scale-invariance. Some authors talk about "traffic scaling" or simply "scaling" to refer to the observed self-similarity in network traffic.

Quantitatively, the change in the arrival variance for a self-similar time series of bin counts X^t is proportional to t^{2H-2}, where $t \geq 1$ represents scale as the aggregation of arrival counts, and H is known as the Hurst parameter. For example, the variance in bin counts in a Poisson process is proportional to $\frac{1}{H} = t^{2(\frac{1}{2})-2}$. That is, a Poisson arrival process has $H = 0.5$. A stationary, long-range dependent process has $0.5 < H < 1$. The closer the value of the Hurst parameter is to 1, the slower the variance decays as scale (t) increases, and the traffic is said to be increasingly more *bursty* (than Poisson arrivals). The slow decay of the arrival variance in self-similar traffic, as scale increases, is in sharp contrast to the mathematical framework provided by Poisson modeling, in which the variance of the arrivals process decays as the square root of the scale (see [LTWW93, PF95]). This quantitative characterization of self-similarity provides us with the right framework to compare real and synthetic traffic, assessing the validity of the traffic generation process in terms of the burstiness of the packet/byte arrival process.

Self-similarity also manifests itself as Long-Range Dependence[13] (LRD) in the time series of arrivals. This means that there are non-negligible correlations between the arrival counts in bins that are far apart. A common way of studying these correlations is to compute the autocorrelation $\rho(k)$ of a time

[13]Long-range dependence is sometimes referred to as *long memory.*

series, where k is the autocorrelation lag. The autocorrelation at lag k,

$$\rho(k) = \frac{\sum_{i=1}^{n-k}(X_i^t - \overline{X}^t)(X_{i+k}^t - \overline{X}^t)}{\sum_{i=1}^{n}(X_i^t - \overline{X}^t)^2},$$

is the correlation between a time series and a shifted version of itself, where the i-th value in the original time series becomes the $i+k$-th value in the shifted time series. The autocorrelation function $\rho(k)$ of a long-range dependent time series decays in proportion to $k^{-\beta}$ as the lag k tends to infinity, where $0 < \beta < 1$. The Hurst parameter is related to β via $H = 1 - \beta/2$, so the closer the value of the Hurst parameter is to 1, the more slowly the autocorrelation function decays. In contrast, Poisson processes are short-range dependent, *i.e.*, their autocorrelation decays exponentially as the lag increases.

The concepts and definitions of self-similarity and LRD assume that the time series of arrivals is *second-order stationary* (also called *weakly stationary*). Loosely speaking, this means that the variance of the time series (and more generally, its covariance structure) does not change over time, and that its mean is constant (so the time series can always be transformed into a zero-mean stochastic process by simply subtracting the mean). The intuitive interpretation of this concept is that the time series should not experience any major change in variance, which would be associated with a fundamental change in the nature of the studied process. For example, a link usually used by 1,000 hosts that suddenly becomes used by 10,000 hosts (*e.g.*, due to a "flash crowd") would show a massive throughput increase, and much higher variance, which would make it non-stationary. These types of major changes are outside the scope of LRD analysis. They represent a coarse-scale feature of the time series which should be studied using other methods (*e.g.*, trend analysis using SiZer [CM99]).

Traffic is certainly not second-order stationary at the scales at which time-of-day effects are important. For example, a 24-hour trace is usually non-stationary due to the sharp decrease in network utilization at night. The number of sources at night is far smaller, which decreases variance, violating the second-order stationarity assumption. Trying to estimate the Hurst parameter of a 24-hour trace that exhibits a time-of-day effect results in a meaningless number. In our traffic generation work, we will estimate Hurst parameters (and other measures of self-similarity) for traces that are second-order stationary. Our traces have moderate durations, between 1 and 4 hours, which greatly diminishes the impact of time-of-day variations. We also carefully examined the packet and byte arrival time series of our traces and found no evidence of sharp changes that could be associated with second-order non-stationary.

Estimation of Hurst parameters is not a trivial exercise. Besides ensuring that no significant second-order non-stationarity is present in the data, common estimation methods are very sensitive to outliers and trends in the data, as pointed out by Park *et al.* [PHCL$^+$]. These difficulties motivate some preprocessing of the studied time series (*e.g.*, detrending) or to employ robust methods. In this dissertation, we will make use of wavelet analysis to study the scaling properties of real and synthetic traffic. We will follow the analysis method of Abry and Veitch [AV98] and make use of their Matlab implementation of the method. In general, we will compute what is called the *wavelet spectrum* of the time series of packet and byte counts in 10 millisecond intervals. This is also referred to as the *logscale diagram* in some works[14]. The *wavelet spectrum* provides a visualization of the scale-dependent variability in the data (see Figure 4.43 for an example). Briefly, a logscale diagram plots the logarithm of the (estimated) variance of the Daubechies wavelet coefficients, the *energy*, as a function of the logarithm of the scale $j = log_2(t)$, where t is the time scale and j is known as the *octave*. The Daubechies wavelet coefficients come from a decomposition of the time series in terms of the Daubechies wavelet basis, which is a collection of shifted and dilated versions of a mother Daubechies wavelet (a function) [Wal99]. Intuitively, this decomposition is similar to the Fourier transform, which decomposes a time series in terms of sinusoidal functions. The wavelet transform also performs a decomposition but it uses a compact support, so it can represent localized features (sinusoidal functions have infinite support). Besides this property, the benefit of the wavelet transform is its robustness to trends in the data, which can easily confuse other types of analysis, such as the variance-time plot [LTWW93]. Wavelet analysis is robust to moderate non-stationarities.

For processes that are long-range dependent, the wavelet spectrum exhibits an approximately linear relationship with a positive slope between energy and octave. For Internet traffic, the region where this linear scale relationship begins is generally on the order of a few hundred milliseconds (4th to 6th octave for 10-milliseconds time series). An estimate of the Hurst parameter H along with a confidence interval on the estimate can be obtained from the slope of the wavelet spectrum, $H = \frac{slope+1}{2}$. See the book edited by Park and Willinger [PW00] for a more complete overview of long-range dependence in network traffic, and papers by Veitch *et al.* [AV98, HVA02] and Feldmann *et al.* [FGHW99, Fel00] for more detail on traffic analysis using wavelets.

Figure 4.43 shows the wavelet spectra of the time series of packet throughputs for fully-captured sequential connections in the Leipzig-I trace. As explained in Section 4.2.1, this type of connection is responsible for the overall burstiness of the traffic in our trace[15]. For comparison, Figure 4.43 also

[14]The rationale for choosing the term "logscale diagram" can be confusing, since it is applicable to any kind of plot in which one or more axes show a logarithmic transformation of the data. The term "wavelet spectrum" is more specific and seems more appropriate and has become the standard in the literature.

[15]The only exception is the Abilene-I trace in the direction from Cleveland to Indianapolis, where routing asymmetries are responsible for most of the burstiness.

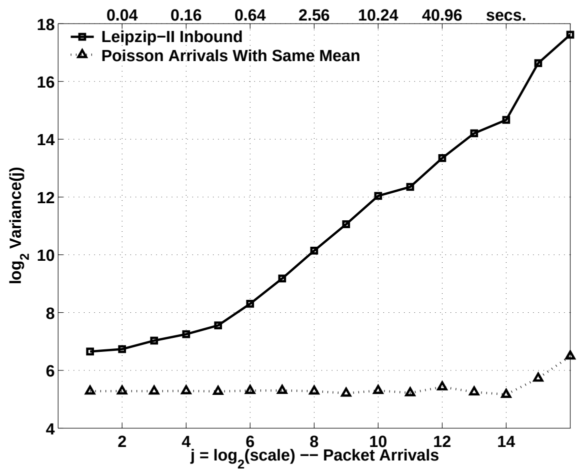


Figure 4.43: Wavelet spectra of the packet throughput time series for Leipzig-II inbound and its Poisson arrival fit.

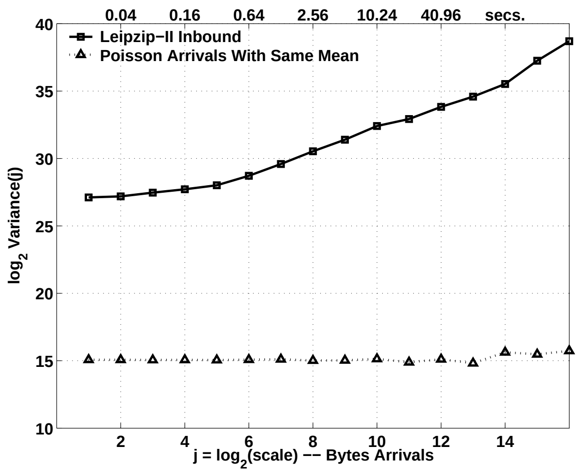
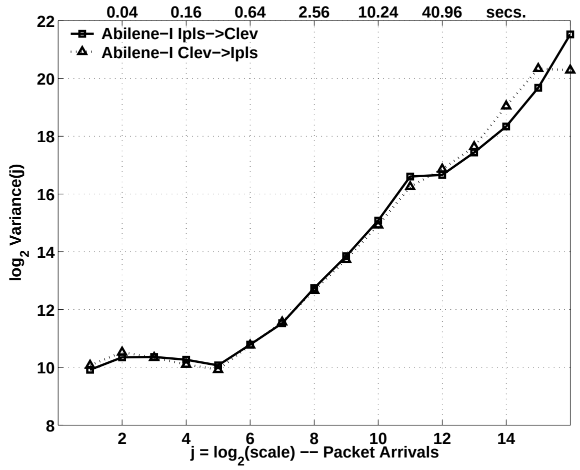


Figure 4.44: Wavelet spectra of the byte throughput time series for Leipzig-II inbound and its Poisson arrival fit.

plots a simulated time series of Poisson arrivals with the same mean (38.94 packets per 10-millisecond bin[16]). Note that only the middle 150 minutes of the Leipzig time series were used, eliminating the non-stationarity created by the boundaries of the trace. The plot shows the variance of the wavelet coefficient (or energy) as a function of the octave. The first octave comes from the dyadic aggregation of 10-milliseconds bins, so it represents the energy at the 20-millisecond scale. The second octave comes from the dyadic aggregation of the bins aggregated in the previous octave, so it represents the energy at the 40-millisecond scale. The same dyadic aggregation is used for every successive scale, so octave 12 represents the energy at the 10 milliseconds times 2^{12} scale, *i.e.*, at the 40.96-second bins. To make the plot more readable, we added labels on top of the plot with the scale given in seconds. Due to the nature of the wavelet basis, the exponential decay of the autocorrelation in a short-range dependent process results in a wavelet spectrum with a slope of zero. On the contrary, the decay in a long-range dependent process is slower than exponential, and results in a wavelet spectrum with a positive slope. The wavelet spectrum of Leipzig-II has a positive slope that indicates long-range dependence, while the synthetic Poisson time series does not show such a trend (it is short-range dependent). Note also that the height of the curves is rather different. This is because the overall variance of the Poisson arrivals is smaller. The standard deviation of the aggregate packet throughput time series was 12.96 while that of the synthetic Poisson arrivals was 6.23. The estimated Hurst parameters were 0.940 (with confidence interval [0.931,0.949]) for the Leipzig-II trace and 0.496 (with confidence interval [0.487, 0.505]) for the synthetic Poisson arrivals.

[16]Note that the simulated time series had exactly this mean, but the number of packets in each bin was always an integer number.



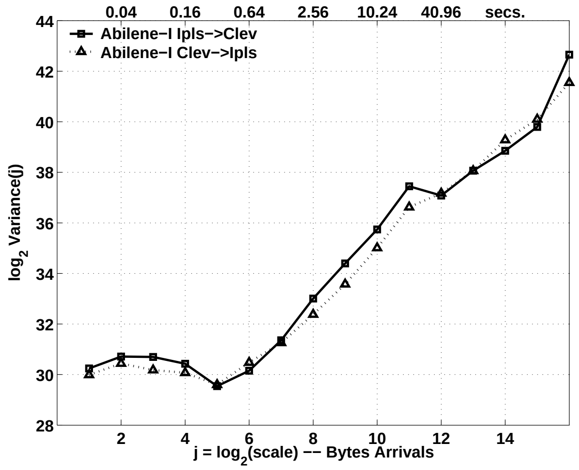


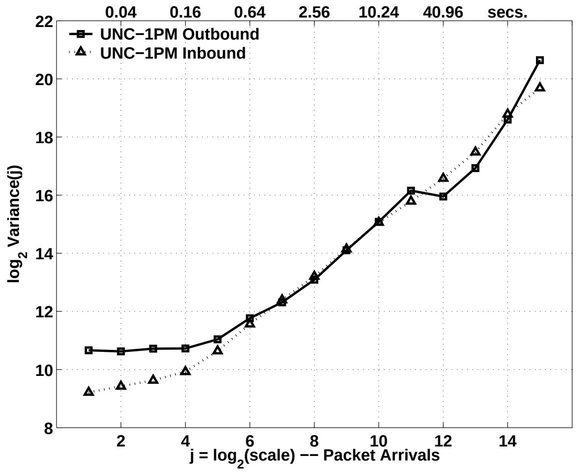
Figure 4.45: Wavelet spectra of the packet throughput time series for Abilene-I.

Figure 4.46: Wavelet spectra of the byte throughput time series for Abilene-I.

The same qualitative results hold for byte arrivals, as illustrated in Figure 4.44. Here the mean number of bytes per 10-millisecond bin for Leipzig was 34,400, and the standard deviation of the trace was 14,000, while the standard deviation of the synthetic Poisson arrivals was only 188. The estimated Hurst parameters were 0.941 (with confidence interval [0.932,0.950]) for Leipzig-II and 0.496 (with confidence interval [0.487, 0.505]) for the synthetic Poisson arrivals.

Figure 4.45 shows the wavelet spectrum of the packet throughput time series for Abilene-I (in the two directions: Indianapolis to Cleveland and Cleveland to Indianapolis). While the overall impression is similar to that of the previous figures, we find a change in slope after the 11th octave. Note that both directions exhibit similar long-range dependence. The estimated Hurst parameters were quite high: 1.016 (confidence interval [1.005, 1.027]) for the Indianapolis to Cleveland trace, and 1.009 (confidence interval [0.998, 1.019]) for the opposite direction. Byte throughput for the same trace shown in Figure 4.46 is qualitatively similar. The estimated Hurst parameters were 1.169 (confidence interval [1.158, 1.180]) and 1.046 (confidence interval [1.035, 1.057]). Both are significantly above 1, so some non-stationarity is present in the trace.

Another example of this type of analysis is given in Figures 4.47 and 4.48. For UNC 1 PM, these diagrams show a large separation between the two directions, that translates into significantly different Hurst parameters. The entire set of Hurst parameters for the traces considered in this dissertation is shown in Tables 4.1 and 4.2.



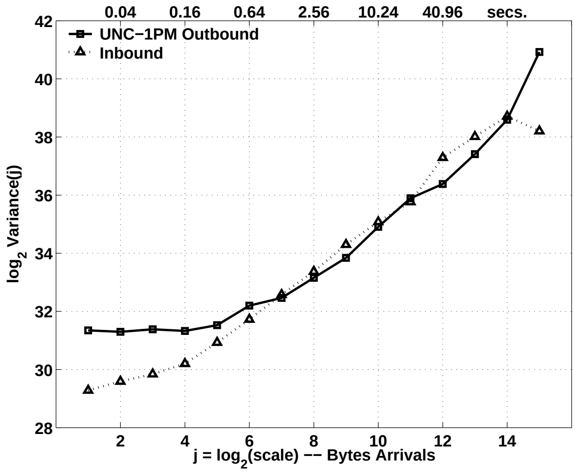


Figure 4.47: Wavelet spectra of the packet throughput time series for UNC 1 PM.

Figure 4.48: Wavelet spectra of the byte throughput time series for UNC 1 PM.

Trace	Estimated Parameters
Leipzig-I Inbound	H=0.940356 C.I.=[0.931459, 0.949254]
Leipzig-I Outbound	H=0.968425 C.I.=[0.959527, 0.977322]
Abilene-I Ipls/Clev	H=1.016014 C.I.=[1.005242, 1.026786]
Abilene-I Clev/Ipls	H=1.008771 C.I.=[0.998000, 1.019543]
UNC 1 PM Outbound	H=0.890024 C.I.=[0.872508, 0.907541]
UNC 1 PM Inbound	H=0.926588 C.I.=[0.909072, 0.944105]
UNC 1 AM Outbound	H=0.906053 C.I.=[0.888537, 0.923569]
UNC 1 AM Inbound	H=0.932574 C.I.=[0.915058, 0.950091]
UNC 7:30 PM Outbound	H=1.001424 C.I.=[0.983908, 1.018940]
UNC 7:30 PM Inbound	H=0.981452 C.I.=[0.963935, 0.998968]

Table 4.1: Estimated Hurst parameters and their confidence intervals for the packet throughput time series of five traces.

Trace	Estimated Parameters
Leipzig-I Inbound	H=0.941176 C.I.=[0.932278, 0.950073]
Leipzig-I Outbound	H=1.019947 C.I.=[1.011049, 1.028844]
Abilene-I Ipls/Clev	H=1.169007 C.I.=[1.158236, 1.179779]
Abilene-I Clev/Ipls	H=1.045921 C.I.=[1.035149, 1.056692]
UNC 1 PM Outbound	H=0.820944 C.I.=[0.803428, 0.838460]
UNC 1 PM Inbound	H=0.925690 C.I.=[0.908174, 0.943206]
UNC 1 AM Outbound	H=0.906226 C.I.=[0.888710, 0.923742]
UNC 1 AM Inbound	H=0.957370 C.I.=[0.939854, 0.974887]
UNC 7:30 PM Outbound	H=0.963306 C.I.=[0.945789, 0.980822]
UNC 7:30 PM Inbound	H=0.970991 C.I.=[0.953474, 0.988507]

Table 4.2: Estimated Hurst parameters and their confidence intervals for the byte throughput time series of five traces.

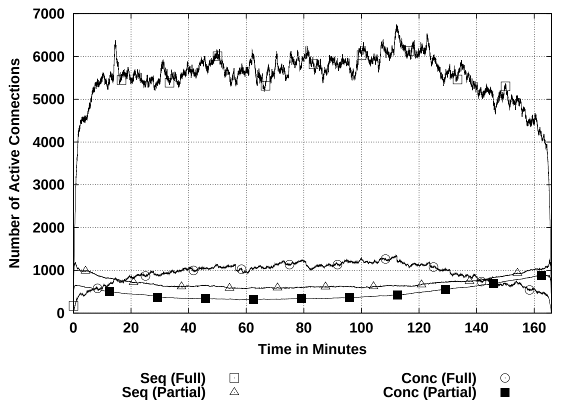


Figure 4.49: Breakdown of the active connections time series for Leipzig-II.

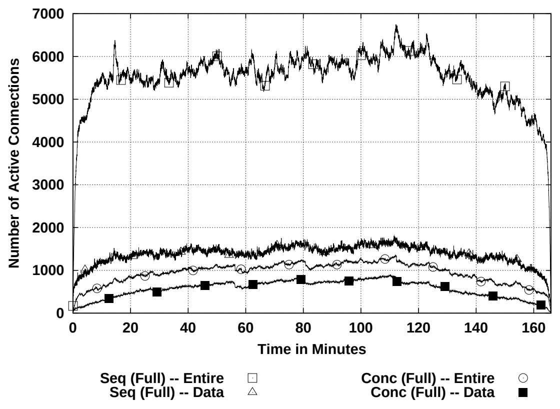


Figure 4.50: Impact of the definition of active connection on Leipzig-II.

4.2.4 Time Series of Active Connections

Another important metric for describing the workload of a network is the number of connections that are simultaneously active. The feasibility of deploying mechanisms that must maintain some amount of state for each connection is highly dependent on this metric. For example, stateful firewalls can selectively admit packets belonging to connections started from a protected network, and not those packets from connections that originated somewhere else on the Internet. This kind of filtering requires to maintain state for every connection observed in the recent past. Similarly, network monitoring equipment often reports on the number of connections and their aggregate characteristics, and tries to identify *heavy-hitters* that consume large amounts of bandwidth. This also requires per-connection state. A good example of this type of monitoring is Cisco's NetFlow [Cor06]. The performance of other mechanisms, such as route caching, may also be affected by the number of active connections. Evaluating these types of mechanisms and their resource consumption requirements can only be accomplished using synthetic traffic that is realistic in terms of the number of connections that are simultaneously active.

One important difficulty when analyzing the time series of active connections is the way connection start and end times are defined. The most obvious way to define connection start and end times is to consider the first and the last segment of a connection as the boundaries of the connection. Figure 4.49 shows the time series of active connections in 1-second intervals using this technique for the Leipzig-II trace. As in the throughput time series in Figures 4.21 and 4.22, the number of active connections from fully-captured sequential connections is much larger than the number of active connections for the other types of connections.

As the focus of our work is on the effect of source-level behavior, we can also use an alternative

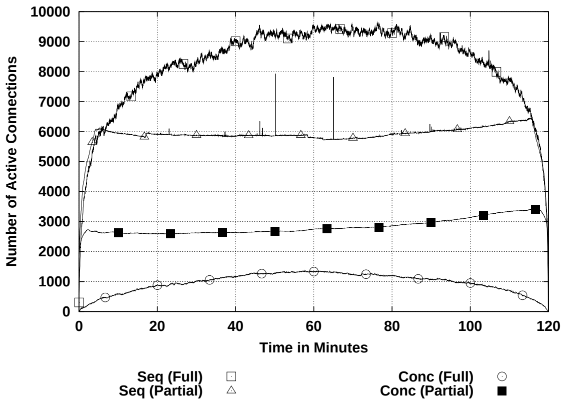


Figure 4.51: Breakdown of the active connections time series for Abilene-I.

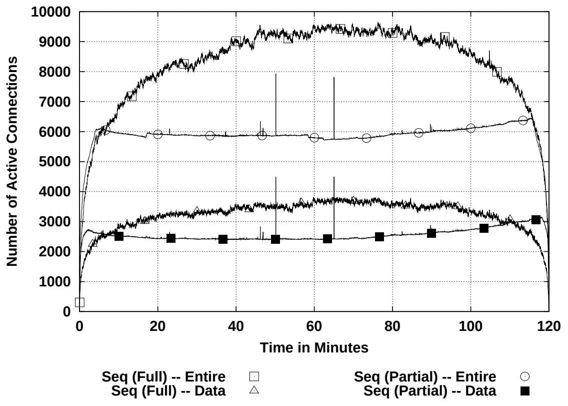


Figure 4.52: Impact of the definition of active connection on Abilene-I.

definition in which a connection is considered active as soon as it sends the first data segment, and inactive as soon as it sends the last data segment. Interestingly, these two definitions result in quite different time series. Figure 4.50 compares the time series for fully-captured sequential and concurrent connections (the time series for partially-captured connections changed very little). The average number of active connections is much smaller when only the data exchange portion of TCP connections is considered. The main cause of this difference is the presence of significant quiet times between the last ADU and connection termination. Figure 3.28 in the previous chapter showed the distribution of this quiet time. In some cases, we also observe quiet time between connection establishment and the first ADU. The duration of connection establishment and connection termination is generally very short (around two round-trip times), but we have observed numerous cases in which losses and TCP implementation problems[17] lengthened them substantially. We believe the second definition, considering only duration between data segments, is more useful for studying the realism of synthetic traffic, since connection establishment and termination create very little network load when compared to the actual exchanges of data. Furthermore, congestion control plays little role when no data is being exchanged. We will use this second definition of active connection in the rest of this work.

The breakdown of the active connection time series for Abilene-I is shown in Figure 4.51. Partially-captured sequential connections are far more significant for this trace, reaching 2/3 of the average number of fully captured sequential connections. We also note that the time series exhibits surprisingly small variability except for a few very small spikes in the middle. Finally, and in contrast to the breakdown of byte throughput for Abilene-I, the number of active connections from partially-captured concurrent connections is less than half of the number of active connections from partially-captured sequential

[17]For example, some implementations send several reset segment after a lossy connection termination, and these segments are often separated by long period of inactivity.

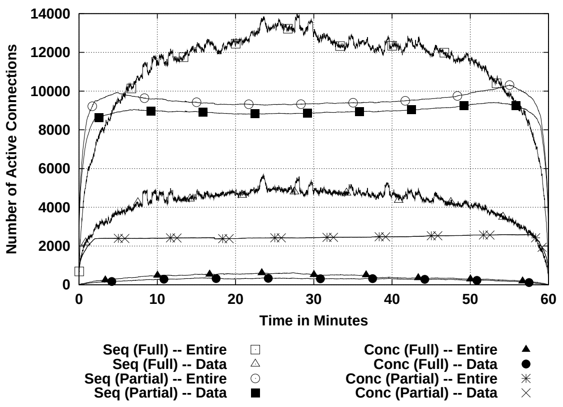


Figure 4.53: Breakdown of active connections time series for UNC 1 PM using both definitions of active connection.

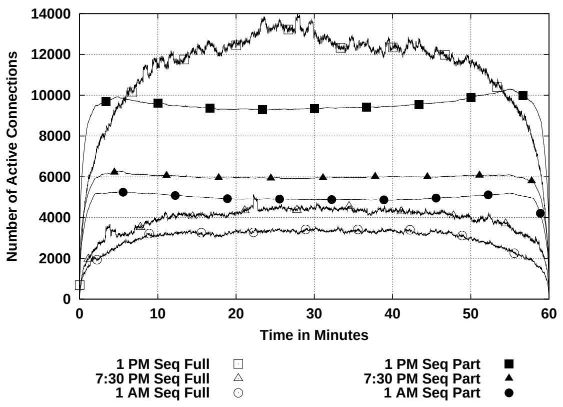


Figure 4.54: Impact of the time-of-day on the active connections time series for the three UNC traces.

connections.

Figure 4.52 illustrates the impact of the definition of active connection. The time series from both fully- and partially-captured sequential connections decrease considerably when only the data exchange part of the connection is considered. Note also that the spikes in the time series of partially-captured connections are not affected by the change in the definition. This is interesting when we observe that the magnitude of the variability of the other time series did decrease significantly.

The largest number of active connections was found in UNC 1 PM as shown in Figure 4.53. This is surprising given that Abilene-I carries more bytes and packets, and should be explained by the differences in the mix of applications that drives the traffic in these two links. The plots shows that fully- and partially-captured sequential connections are affected in a very different way by the definitions of active connections. While the number of active connections for fully-captured sequential connections decreases very significantly, the number for partially-captured ones is almost the same. This can be explained by long connections that were active throughout the entire duration of the trace.

Finally, Figure 4.54 studies the impact of the time of the day on the time series of active connections for sequential connections. The number of fully-captured sequential connections is more sensitive to the definition of active connection than the number of partially-captured sequential connections.

4.3 Summary

The first part of this chapter presented our approach for introducing realistic *network-level parameters* in our traffic generation methodology. In particular, we considered how to measure three basic network parameters that have a major impact on the throughput of a TCP connection: round-trip time, receiver window size, and loss rate. As in our analysis of source-level behavior, we focused on the efficient analysis of segment headers for extracting these network parameters, and evaluated the accuracy of our chosen measurement methods using testbed experiments.

Our discussion on measuring round-trip time considered the classic SYN estimator, and proposed a novel technique based on computing one-side transit times (OSTTs). Our technique has two main advantages. First, it is applicable to connections observed both on the edges and on the core of the network. In either case, it provides us with a way to measure the distance, in terms of network delay, between the monitoring point and the end hosts taking part in each connection. Second, OSTT-based estimation provides a number of samples proportional to the number of data segments on a TCP connection, unlike the single sample that can be obtained using the SYN estimator. This provides a better way to understand the inherent variability in round-trip times. It also served us to study the impact of delayed acknowledgments on path round-trip time estimation from segment headers. We clearly showed that delayed acknowledgments substantially inflate estimates of round-trip time that rely on non-robust statistics like averages and maxima. For this reason, we favor the use of minima or medians to estimate path round-trip time, which were proved to be highly accurate in our testbed experiments.

We also studied the empirical distributions of round-trip times in our collection of five traces. We can highlight several observations. The edge traces from UNC and Leipzig showed between 20% and 35% of connections with very short round-trip times below 20 milliseconds. In contrast, the backbone trace from Abilene showed less than 1% of connections with these small round-trip times. Our analysis of the total number of bytes carried in connections with a given round-trip time revealed that Leipzig-II had a far larger fraction of bytes (10%) carried in connections with round-trip times above 500 milliseconds. The distributions of round-trip times did not only differ substantially on their range, but also on their shapes, even among those collected on the same site. For example, the UNC 1 PM trace showed only 15% of connections with round-trip times above 100 milliseconds, while this percentage became 25% and 38% for UNC 7:30 PM and 1 AM respectively.

The second parameter we considered is the maximum size of the receiver window, which, in combination with the round-trip time, puts a hard limit on the maximum throughput of a TCP connection. This parameter is straight-forward to measure, since each TCP segment contains a field with the size of the receiver window at the time of its sending. Taking the maximum of the observed receiver windows provides an accurate way of measuring the largest receiver window supported by an endpoint, even for connections that grow their limit some time after the connection is opened. We used this technique to study the distribution of maximum receiver window sizes in our traces, and found a large fraction of connections with a small maximum. Between 45% and 65% of the connections had maximum receiver window sizes below 20 KB, which is well below the 64 KB limit.

The last network parameter that we studied was the segment loss rate. Loss has a substantial impact on TCP connections. First, losses force the endpoints to retransmit segments to maintain a reliable communication. Second, TCP endpoints use losses as the signal of congestion, and react to them by lowering their sending rate. For these two reasons, even a small number of losses can have a dramatic effect on a TCP connection. Measuring loss rates purely from segment headers must necessarily be based on the same mechanisms used by TCP endpoint to detect losses: retransmissions and duplicate acknowledgments. We proposed a technique to measure the loss rate of data segments using these signals, where differentiating between losses before the monitoring point, detected using duplicate acknowledgments, and losses after the monitoring point, detected using retransmissions. Our evaluation using testbed experiments showed that our technique is reasonably accurate. The experiments also illustrate the impact of lost acknowledgments, which increase data segment loss rates, and variability introduced by simulating losses using *dummynet* 's dropping mechanism. We also studied the loss rates in our traces, and found that between 92.5% and 96.2% of the TCP connections experienced no losses. However, connections with one or more losses accounted for 46% (Leipzig-II) to 78% (UNC 1 AM) of the total bytes in traces, and connections with loss rates above 1% (*i.e.*, moderately high) accounted for 8% (Abilene-I) to 34% (UNC 1 AM) of the total bytes.

The second part of this chapter described our approach for comparing real and synthetic traffic using several *network-level metrics*. The goal of such a comparison is to evaluate how closely synthetic traffic generated on a closed-loop manner can reproduce the aggregate characteristics of real traffic. This type of comparison concerns itself with the *extrinsic* characteristics of the generated traffic, which were not a direct input to the traffic generators. On the contrary, evaluating how well source-level properties and network-level parameters are preserved by our traffic generation method and its implementation focuses on *intrinsic* characteristic of the generated traffic, which are the input to the traffic generation system. We first discussed how to study the time series of packet and byte throughputs, using plots of time series at a coarse scale, tens of seconds. This broad view was specially useful to identify major trends and features in the traffic. We used this approach to study the composition of our traces, finding that sequential connections are mostly responsible for the features of the time series, being the aggregate throughput for concurrent connections generally smooth. We further differentiate between traffic from connections for which we observed every packet between TCP connection establishment and termination, uncovering substantial boundary effects in the UNC traces and to some extent in the Abilene-I trace. We also showed that the fraction of the total throughput from unidirectional connections is generally negligible. The only exception is Abilene-I, where routing asymmetries explain the finding that 1/4 of total Cleveland-to-Indianapolis bytes were carried in connections whose packets appear in only one direction of the trace.

The second way in which we proposed to examine throughput was to construct the marginal distributions of the time series at a fine-scale (10 milliseconds). While marginals ignore dependency structure, their interpretation in networking terms is intuitive. Plots of the body of the marginal distribution provide an overview of the range of fine-scale throughputs in a trace, while plots of the tail of the marginal distribution make the highest (fine-scale) throughputs stand out. The analysis of our traces showed that Poisson arrivals cannot be used to model neither packet or byte throughputs. The bodies of the marginal distributions from our traces are between 2 and 3 times more variable that the ones from Poisson arrivals with the same mean. We also showed that the marginal distributions from our traces have statistically significant departures from normality, which are most prominent on the tails. This was demonstrated using two methods, Q-Q plots with simulation envelopes and the Kolmogorov-Smirnov test of normality. Both methods were applied to scales of aggregation between 10 milliseconds and 10 seconds. While the distributions became closer to normality as scale increased, only a few of them were statistically consistent with the normal distribution at the 10 second scale. For this reason, our analysis of marginal distribution will rely on CDFs of the bodies and CCDFs of the tails, rather than making assumptions about the underlying statistical distribution.

Our third type of analysis of throughput focused on the long-range dependence of traffic. We employ the wavelet analysis for this purpose, which has been shown to be robust and accurate in the literature. This method provides both an overview of the way in which variability changes with scale using wavelet spectra plots, and a state-of-the-art estimator of Hurst parameter with confidence intervals. Our discussion illustrated how clearly wavelet spectra and Hurst parameter estimates differentiate between the short-range dependence in Poisson arrivals and the long-range dependence in our traces. Our traces show remarkably high Hurst parameter estimates, well above 0.9 for both packet and byte throughput.

Finally, the chapter introduced the plot of the time series of active connections. This type of analysis is essential to validate the realism of traffic generation for certain experiments where per-connection state is important. Our analysis considered two definitions of active connections: a connection was considered active between the arrivals of its first and last segments, or between the arrivals of its first and last segments that carried application data, *i.e.*, not control segments. We demonstrated that these two definitions have a dramatic impact on the number of active connections. We will favor the latter definition (data active connections) for our evaluation in Chapter 6, since the focus of our modeling is the source-level behavior in terms of useful data exchanges. Our discussion of active connections also considered the effect of trace boundaries, revealing a large fraction of active connections from partially-captured connections.