CHAPTER 5

Generating Traffic

Today's scientists have substituted mathematics for experiments, and they wander off through equation after equation, and eventually build a structure which has no relation to reality.

— Nikola Tesla (1857–1943)

Reality is merely an illusion, albeit a very persistent one.

— Albert Einstein (1879–1955)

This chapter discusses the use of the data acquisition and modeling methods presented in the two previous chapters to generate traffic in network experiments. In addition, it discusses the overall methodology we have developed for validating our traffic generation approach. We will distinguish between validating the method itself, and studying how closely the generated traffic approximates real traffic for properties not directly incorporated in the method. In this chapter, we consider the validation of the method itself, which means to verify that the source-level properties and network-level parameters of the traffic are preserved by the traffic generation method. The study of other properties is left for the next chapter.

5.1 Replaying Traces at the Source-Level

Our approach to traffic generation is illustrated in Figure 5.1. Given a packet header trace $\mathcal{T}_h$ collected from some Internet link, we first use the methods described and evaluated in Chapters 3 and 4 to analyze this trace and describe its content. This description is a collection of connection vectors $\mathcal{T}_c$. Each vector describes the source-level behavior of one of the TCP connections in $\mathcal{T}_h$ using either the sequential or the concurrent a-b-t model. In addition, each vector includes the relative start time of each connection, and its measured round-trip time, TCP receiver window sizes and loss rate. The basic

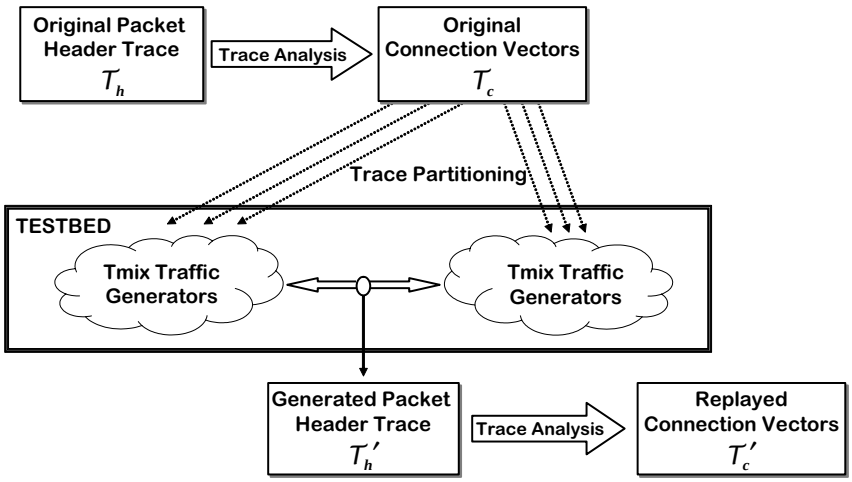

Figure 5.1: Overview of Source-level Trace Replay.

approach for generating traffic according to $\mathcal{T}_c$ is to replay each connection vector. For each connection vector, the replay consists of starting a TCP connection, carefully preserving its relative start time, and reproducing ADUs and inter-ADU quiet times. We call this traffic generation method *source-level trace replay*, and we have implemented it in a network testbed. Source-level trace replay in our environment implies the need to first partition $\mathcal{T}_c$ into disjoint subsets and then assign each subset to a pair of traffic generators. Partitioning is important in our environment, since the high throughput and large number of simultaneously alive connections in our real traces prevents us from using a single pair of traffic generators. We provide further details on our partitioning method in 5.1.1.

The goal of the direct source-level trace replay of $\mathcal{T}_c$ is to reproduce the source-level characteristics of the traffic in the original link, generating the traffic in a *closed-loop* fashion. Closed-loop traffic generation requires to simulate the behavior of applications, using regular network stacks to actually translate source-level behavior (the input of the generation) into network traffic (the output of the generation). In our implementation, described in Section 5.1.2, this is accomplished by relying on the standard socket interface to reproduce the communication in each connection vector. This is a closed-loop manner of generating traffic in the sense that it preserves the feedback mechanisms in the TCP layer, which adapt their behavior to changes in network conditions, such as in congestion. In contrast, packet-level trace replay, which means to directly reproduce $\mathcal{T}_h$, is an open-loop traffic generation method where TCP and lower layers are not used, and the traffic does not adapt to network conditions.

A new packet header trace $\mathcal{T}_h'$ can be obtained from the source-level trace replay of $\mathcal{T}_c$. Our validation of the traffic generation method is then based on analyzing this trace using the same methods used to

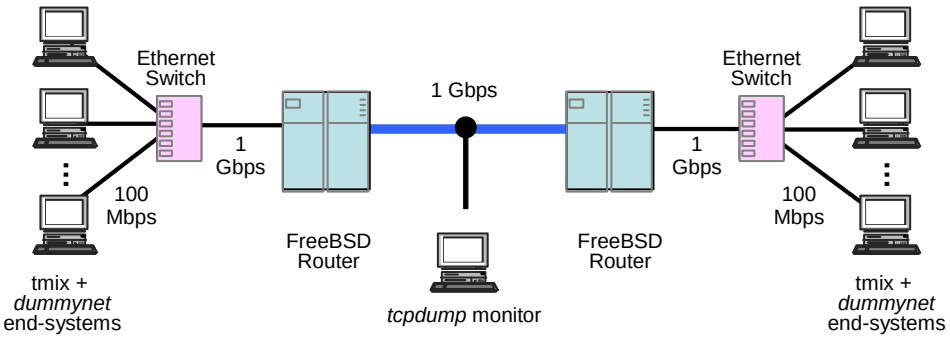

Figure 5.2: Diagram of the network testbed where the experiments of this dissertation were conducted.

transform $\mathcal{T}_h$ into $\mathcal{T}_c$. We then compare the resulting set of connection vectors $\mathcal{T}_c'$ with the original $\mathcal{T}_c$. In principle, they should be identical, since $\mathcal{T}_c$ represents the invariant source-level characteristics of $\mathcal{T}_h$. Section 5.2 studies the results from the source-level trace replay of three traces, assessing how closely $\mathcal{T}_c'$ approximates $\mathcal{T}_c$. $\mathcal{T}_h'$ is necessarily different from $\mathcal{T}_h$. Besides the stochastic nature of network traffic, this is because $\mathcal{T}_h'$ is generated according to $\mathcal{T}_c$, which is a simplified description of the source-level behavior and network parameters in the original trace $\mathcal{T}_h$. It is however important to understand the difference between $\mathcal{T}_h$ and $\mathcal{T}_h'$ in order to understand to what extent $\mathcal{T}_c$ describes the original traffic. Chapter 6 is an in-depth study of this question.

5.1.1 Trace Partitioning

The focus of our traffic generation work is the generation of wide-area traffic in a closed-loop manner. This type of generation process requires to drive a large number of connections by simulating the behavior of the applications on the endpoints. For example, the experiments presented in the latter part of this chapter involve several millions of TCP connections, behaving in the manner specified by as many connection vectors. At any given point in time during the generation, tens of thousands of connections are active. Given CPU, memory and bus speed limitations, a single pair of traffic generators cannot handle such loads, so we generate traffic in our experiment in a distributed fashion. Experiments are conducted in the environment illustrated in Figure 5.2. The goal of the experiment is to generate traffic on the link between the two routers. Traffic is generated by 42 traffic generators, 21 on each side of the network. This type of topology is usually known as the "dumbbell" topology.

Each pair of traffic generators (one on each side) is responsible for replaying the source-level behavior of a (disjoint) subset of the connection vectors in $\mathcal{T}_c$. In our experience, assigning connection vectors to

subsets in a round-robin fashion works well. While the resulting subsets are far from being completely balanced, this simple partitioning technique results in subsets that can be easily handled by a pair of traffic generators. We carefully collected statistics on CPU and memory utilization from our source-level trace replay experiments, and found that no pair of traffic generators was ever overloaded. For the results in this dissertation, CPU utilizations were never above 60%, and usually well below that. The use of network connections involves allocating and deallocating pieces of memory known as "mbufs" for buffering purposes. No request for this type of memory was ever denied for the experiments reported in this dissertation. While larger traces than the ones we use in this dissertation could certainly overload our specific environment, our approach is fully scalable, in the sense that $\mathcal{T}_c$ can be partition into an arbitrary number of subsets. This means that the number of traffic generators can increase as much as necessary to handle the replay of any trace without running into resource constraints. This is obviously true as long as no individual connection requires more resources than those provided by an entire traffic generator end host.

5.1.2 Conducting Experiments

We have developed a traffic generation tool, *tmix* , which accurately replays the source-level behavior of an input set of connection vectors using real TCP sockets in a FreeBSD environment. In addition, we make use of a modified version of *dummynet* [Riz97] to apply arbitrary packet delays and packet drop rates to the segments in each connection[1] Our version of *dummynet* , that we will call *usernet* in the rest of this text, implements a user-level interface that can be used by *tmix* instances to assign per-connection delays and loss rates read from the input set of connection vectors. Finally, a single program, *treplay*, is used to control the setup of the experimental environment, configure and start *tmix* instances (assigning them a subset of $\mathcal{T}_c$ and a traffic generation peer), and collect the results.

Tmix is a user-level program that receives a collection of connection vectors as input, and generates traffic according to their source-level behavior. Figure 5.3 illustrates the relationship between *tmix* and the network layers in the traffic generation end host in which a *tmix* instance runs. *Tmix* instances rely on the standard socket interface to create a connection, send and receive ADUs, and to close the connection. The socket interface is an Application Programming Interface (API) that enables user-level programs, such as *tmix* , to communicate with other end host using a programming abstraction similar to a file. Calls to the socket interface are translated by the kernel into requests to use the process-to-

[1]We thank the members of the FreeBSD project in general, and in particular the creator of *dummynet* , Luigi Rizzo, for their outstanding work. Our empirical work would not have been possible without their generous efforts.

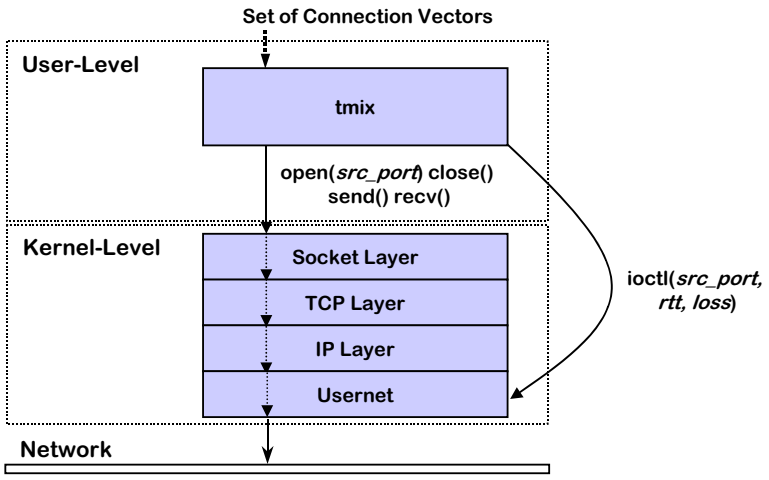

Figure 5.3: End-host architecture of the traffic generation system.

process communication service provided by the transport layer (TCP). The transport layer itself uses the host-to-host communication service provided by the network layer (IP), and the network layer uses the link layer (Ethernet in our case) to handle the network interface and create physical packets.

Usernet

Our experiments also require a special simulation service, *usernet* , which is a modified version of *dummynet* , that provides a highly scalable way of imposing per-connection round-trip times and loss rates. These per-connection round-trip times and loss rates are directly controlled from the user level by *tmix* instances. This requires a direct communication between the *tmix* instance and the *usernet* layer that is not directly supported by the network stack. In order to overcome this difficulty, we use a covert communication channel: the source port number of each replayed connection. By having *tmix* assigning specific source port numbers to each connection, we can then use `ioctl` calls to modify a table at the *usernet* layer that maps source port numbers to round-trip times and loss rates. When a segment is received by *usernet* (from the higher layer), *usernet* can appropriately use the source port number to decide which network parameters should be applied. Source port numbers are unique for each active connection in the same end host, and they are always present in TCP segments[2]. The user-level program, *i.e.*, the *tmix* instance, has therefore to keep track of the (dynamic) source port number that is used for each new TCP connection it opens. Using this technique, *usernet* can determine the delay and loss rate that should be applied to each segment simply by reading an entry in a table indexed by source port

[2]Fragmentation takes place below the *usernet* layer. Figure 5.3 can be confusing in this regard, since fragmentation does take place at the IP layer. *Usernet* is actually embedded in the IP layer.

number, so the lookup time is $O(1)$. The number of source port numbers is small (2^{16}), so this table does not require too much kernel memory (524 KB). No special infrastructure was required to accurately replay the receiver window sizes measured for each connection. This is because these parameters can be directly modified by *tmix* instances using a FreeBSD system call. This approach has worked very well in our experiments.

An alternative solution using traditional *dummynet* would be to use the programmable API of `ipfw`, which makes it possible to add new *dummynet* rules from a user-level program. The idea would be to add a new rule for each connection, again using the source port number to map delay/loss to individual connections. However, this will mean an $O(n)$ lookup cost for each segment, where n is the number of rules, since the current implementation of `ipfw` searches through the rules in a sequential fashion. Given the large number of connections that each end host handles during the experiments, this per-segment lookup is unacceptable.

Another way of introducing per-connection round-trip times was used by Le *et al.* [LAJS03]. This study used random sampling from a uniform distribution whose parameters were be set at the start of the experiment. As seen in Section 4.1.1, the uniform distribution is not a good approximation of real round-trip times. A later refinement enabling sampling from an empirical distribution was rather inflexible, since it required to modify the *dummynet* source code and recompile it for each experiment. The use of *usernet* , which is fully controllable from the user level, is far more convenient.

Replaying an a-b-t Connection Vector

Two instances of *tmix* can replay an arbitrary subset of $\mathcal{T}_c$ by establishing one TCP connection for each connection vector in the trace, with one instance of the program playing the role of the connection initiator and the other instance playing the role of the connection acceptor. To begin, the connection initiator opens the connection and performs one or more socket writes in order to send exactly the number of bytes specified in the first ADU a_1. The other endpoint accepts the connection and reads as many bytes as specified in the ADU a_1. For efficiency, the size of these read and write operations was chosen to be a multiple of the MSS in our Ethernet testbed (1,460 bytes). We made no attempt to actually measure and reproduce the size of the I/O operations in the original connections. The impact of this simplification is likely to be small, given the results in Section 3.4.

One important issue is how to synchronize the two endpoints (*i.e.*, instances of *tmix*) of the connection to replay exactly the same connector vector. This is accomplished by having the first ADU

unit in each generated connection include a 32-bit *connection vector id* in the ADU's first four bytes. Connection vector ids are assigned to each connection vector prior to the traffic generation, and they are unique. Since this id is part of the content of the first data unit, the acceptor can unambiguously identify the connection vector that is to be replayed in this new connection. If a_1 is less than 4 bytes in length, the connection initiator will open the connection using a special port number designated for connections for which the id is provided by the connection acceptor. This approach guarantees that the two *tmix* instances always remain properly synchronized (*i.e.*, they agree on the C_i they replay within each TCP connection) even if connection establishment segments are lost or reordered. It also makes it possible to generate traffic without introducing any control traffic into the experiment, *i.e.*, traffic comes only from the replay of connection vectors, and from any need to manage the behavior of the *tmix* instances.

One important design consideration in the implementation of our traffic generation approach is the assumption of independence among flows. While this is not completely realistic, the level of aggregation at which we generate traffic makes it a reasonable approach (see Hohn *et al.* [HVA02] for a related discussion). This assumption makes traffic generation fully scalable, since $\mathcal{T}_c$ can be partitioned into an arbitrary number of subsets. As long as there are enough traffic generation hosts, we can replay traffic from arbitrarily large traces.

5.1.3 Data Collection

We obtain two types of data from each experiment. First, we collect a new packet header trace $\mathcal{T}_h'$, which can be directly compared with the original packet header trace $\mathcal{T}_h$ and analyzed with our methods to extract a new set of connection vectors $\mathcal{T}_c'$. This new set can be directly compared to $\mathcal{T}_c$. Second, *tmix* instances create a number of logs. Some *tmix* logs can be used to verify that the traffic generation host did not run out of resources during traffic generation, and they successfully replayed their subset of $\mathcal{T}_c$. Other *tmix* logs report on the performance of the TCP connections in the experiments. This includes connection and epoch response times and the list of uncompleted connections with a description of their progress by the end of the experiment.

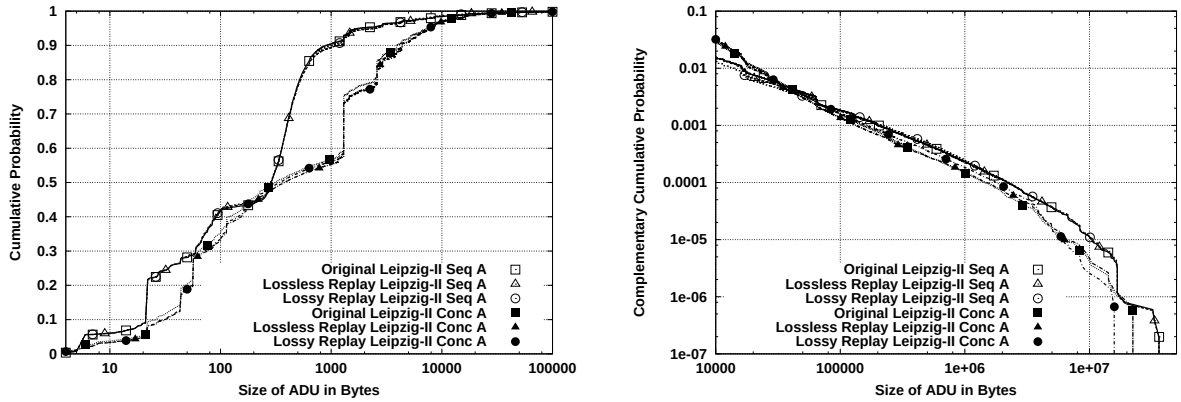

Figure 5.4: Bodies and tails of the A distributions for Leipzig-II and its source-level trace replays.

5.2 Validation of Source-level Trace Replay

In this section, we consider the source-level trace replay of the three packet header traces: Leipzig-II, UNC 1 PM, and Abilene-I. The first goal is to study how well the replay experiments preserve the source-level input, which is the collection of connection vectors $\mathcal{T}_c$ extracted from the original trace $\mathcal{T}_h$. In principle, the characterization of source-level behavior using the a-b-t model represents characteristics of each connection that are invariant to network conditions, so the analysis of the generated trace $\mathcal{T}_h$ should result in a collection of connections vectors $\mathcal{T}_c'$ that is identical to $\mathcal{T}_c$. In practice, there are some practical limitations that make the two sets of connection vectors different. We will discuss the possible causes in this section, and present a statistical comparison of $\mathcal{T}_c$ and $\mathcal{T}_c'$.

The second goal of this section is to study the impact of introducing packet losses in the generated process. For this purpose, we conducted two source-level trace replays of each original trace. The *lossless replay* reproduced the a-b-t connection vector of each original connection, and gave each connection its measured round-trip time and TCP receiver window sizes. The *lossy replay* additionally applied its measured loss rate to each replayed connection. Differences between the lossless and lossy replays tell us about the robustness of both our source-level characterization and traffic generation tools in the presence of losses. These losses are completely absent from our experiments unless they are artificially introduced using *usernet* , as in the lossy replay.

5.2.1 Leipzig-II

The plots in Figure 5.4 compare the distributions of a-type ADU sizes, A, for the original set of connection vectors in Leipzig-II, and for the sets of connection vectors extracted from its lossless and

lossy replays. In each plot, the three distributions marked with white symbols correspond to sequential connection vectors, and the ones marked with black symbols to concurrent connection vectors. The left plot shows the bodies of the distributions, using CDFs in log-linear axes. The right plot shows the tails of the distributions, using CCDFs in log-log axes. In general, there is an excellent agreement between the original distributions and those from the source-level replays.

The bodies of the distributions from sequential connections lie on top of each other, even if per-connection loss rates are used during the experiments. As discussed in 3.4, our ADU measurement algorithm can sometimes be inaccurate when one of the last segments of a TCP window is lost before the monitor. In this case, the loss is recovered after a timeout, which can create a quiet time between the consecutive segment that is long enough to unnecessarily split an ADU. This means that a sample a_i from one of the a-type data units in $\mathcal{T}_c$ becomes two samples a'_i and a'_{i+1} in $\mathcal{T}'_c$, such that $a'_i + a'_{i+1} = a_i$. The validation of the data acquisition methods in Section 3.4 demonstrated that *ADU splitting due to TCP timeouts* is possible, although its impact was small even when large data units and aggressive loss rates were used. The comparison of the Leipzig-II lossless and lossy replays, which represent much more realistic traffic, shows that ADU splitting due to TCP timeouts has very little impact in practice, at least for the relatively light distribution of loss rates in Leipzig-II. We can hardly observe any difference between the bodies of the A distributions when losses are added to the replay. The two bodies from the replay are also very similar to the body of the original distribution. The same is true for the tails, which do not show any significant difference. This analysis demonstrates that *tmix* can accurately reproduce the sizes of a-type data units in sequential connections, even when ADUs are large and when experiments are lossy.

There is also a very good match between the A distributions for concurrent connection vectors. In some regions, we notice somewhat thicker lines that come from small offsets of the curves. The tails of the A distribution for concurrent connections are also very similar, although the one from the lossy replay is slightly heavier for values below 5 MB, and slightly lighter for values above that. This could be explained by the inaccuracy discussed above, or by trace boundaries. In the latter case, losses reduce throughput, making the replay of lossy connections are slower than the replay of lossless ones. This means that some a-type ADUs may not have time to complete their transmission before the end of the experiment.

Figure 5.5 compares the distribution of b-type ADU sizes, B, for the connections vectors extracted from the original Leipzig-II trace and their lossless and lossy source-level replays. For sequential con-

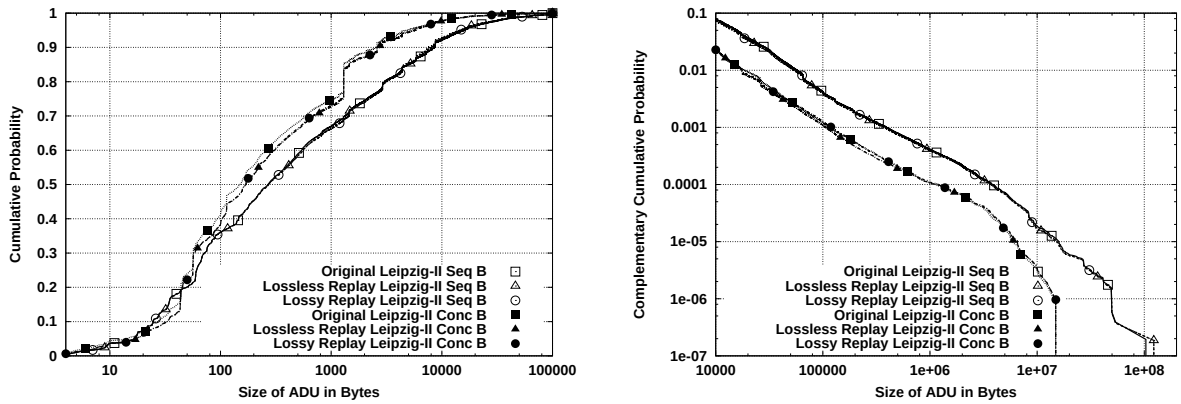

Figure 5.5: Bodies and tails of the B distributions for Leipzig-II and its source-level trace replays.

nection vectors, both the bodies and the tails are identical. For concurrent connection vectors, the distributions show slightly different bodies, but identical tails. The differences cannot be explained by the ADU splitting due to TCP timeouts. If so, we would see a difference between the distributions from the lossless replay and the ones from the lossy replay, but this is not the case. The source of the difference is an inherent problem with the replay of concurrent connections, the *misclassification of the replayed concurrent connections*. While *tmix* always replays a concurrent connection vector in the right way (*i.e.*, decoupling the two directions), the actual set of segments observed at the monitor may simply not have any pair of data segments that satisfy the concurrency test given in Section 3.3.3. In other words, the segments of a replayed concurrent connection may exhibit a fortuitous sequential ordering. As a consequence, the data analysis algorithm classifies as sequential some connections from the replay that were concurrent in the original trace. The sizes of the b-type ADUs in these misclassified connections are then absent from the B distribution for replayed concurrent connections. The small difference in the plot between the original and replayed distributions demonstrates that the number of misclassifications is relatively small, so the majority of the concurrent connections still exhibit concurrent behavior in the replays.

It is important to note that the probability of a misclassification decreases as the sizes of the ADUs increase, since the larger number of data segments makes finding a concurrent pair more likely. Therefore, misclassifications become less significant for the tails of the distributions, since the connections whose samples are in the tail have necessarily at least one large ADU (the one we see in the tail), and are less likely to be misclassified. There is no appreciable difference between the tails of the B distributions from concurrent connections, in agreement with our observation regarding the lower likelihood of misclassification for connections with large ADUs. Misclassified connections are described using the sequential a-b-t model, so they result in additional samples for the distributions that characterize se-

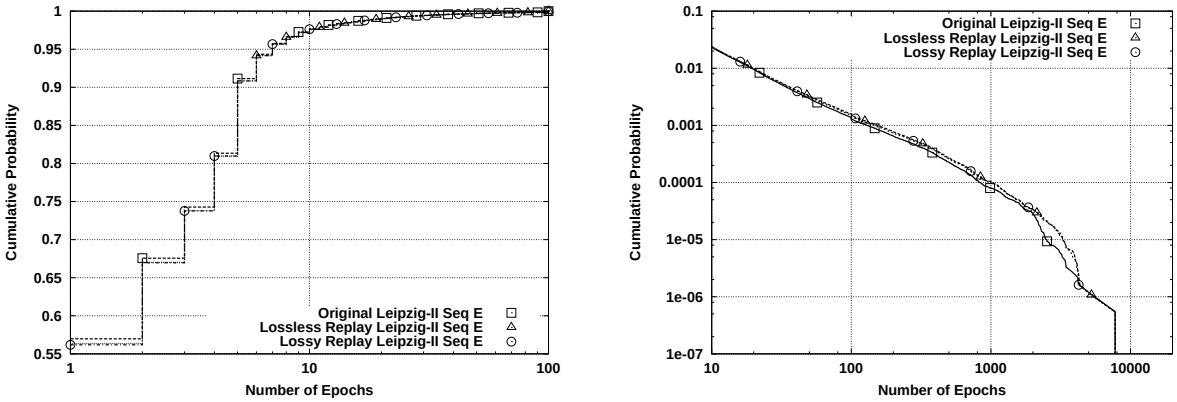

Figure 5.6: Bodies and tails of the E distributions for Leipzig-II and its source-level trace replays.

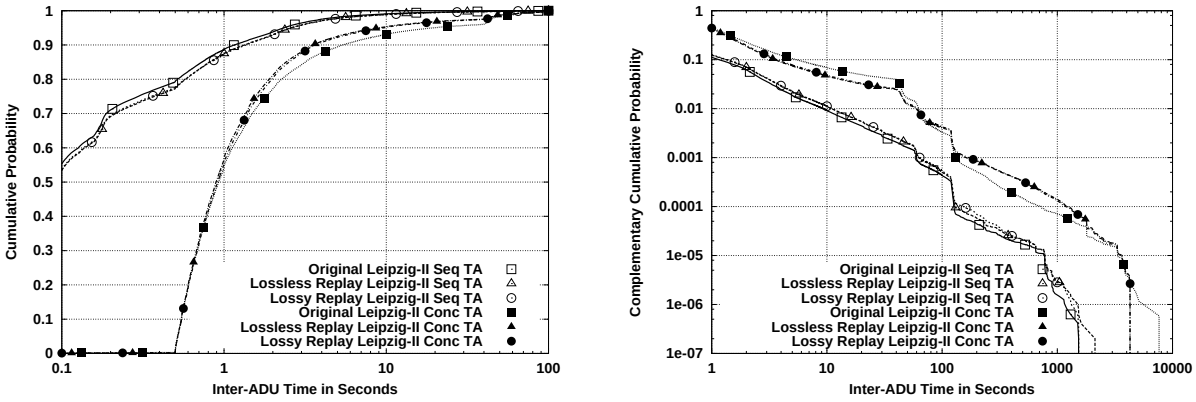

Figure 5.7: Bodies and tails of the TA distributions for Leipzig-II and its source-level trace replays.

quential connection vectors. These extra samples have a much smaller effect on the CDFs, since the number of samples from sequential connections is far larger anyway.

Figure 5.6 considers the distribution of the number of epochs E extracted from the original and from the generated packet header traces. The distributions from the replays are very similar to the original one. The small difference comes again from the small number of misclassified concurrent connections that were considered sequential. Misclassified connections add extra samples to E which slightly distort the distributions from the replays. There is a somewhat bigger difference in the far tail, for connection vectors between 1250 and 1500 epochs. This difference could be explained by misclassification and by trace boundaries (connections replayed more slowly than in the original that do not replay all of their epochs). We observe no difference between lossless and lossy replays in this part of the tail.

The next pair of plots, shown in Figure 5.7, examines the distribution TA of the quiet times on the acceptor side of TCP connections, *i.e.*, between a_i and b_i. The plot of the bodies shows a very good match between the original distribution and the ones measured from the replays of sequential

connections. The slightly heavier distributions from the replays is due to a small simplification we made regarding the replay of quiet times. *Tmix* will replay the exact quiet times specified in each connection vector. However, as discussed in Section 3.3.1, when these quiet times are extracted from a packet header trace, the measured quiet time is the sum of two components. The first component comes from the quiet time q at the end host, and the second component comes from the delay d between the monitor and the endpoint. When *tmix* replays a quiet time, it remains quiet for the exact duration of the sum of these components, $q + d$. Given that the replay in the testbed uses *usernet* to reproduce the measured round-trip time of each connection, there is also a delay between *tmix* end hosts and monitor, so the analysis of the generated packet header trace results in quiet times of the form $q + 2d$. It would have been possible to eliminate this inaccuracy by subtracting d from the originally measured quiet times. The value of d is equal to half of the one-side transit time, although delayed acknowledgments and queuing can affect individual samples. We did not try to incorporate a correction for this *quiet time overestimation problem* in our experiments. Besides measurement difficulties, the extra delay becomes less significant in larger quiet times, for which d is far smaller than q. Larger quiet times are far more significant, since they are the ones that can increase the duration of TCP connections substantially.

There is also a good agreement in the tails of the TA distributions, although the distributions from the replays are slightly heavier than the original distributions. This is not explained by the previous overestimation of quiet times due the location of the monitor, because the magnitude of the quiet times in the tail is far larger than the magnitude of d. The source of this small mismatch is the misclassification of some concurrent connections. This is true for both the differences between the tails from sequential connection vectors and between the tails from concurrent connection vectors. It may seem counter-intuitive that the misclassifications makes both types of tails heavier, instead of making one type of tail heavier and the other one lighter. The explanation is that misclassifications move samples from concurrent connections to sequential connections. These moved samples satisfy at the same time the following two properties:

- They have a lighter tail than the tail of the samples left in connections correctly classified as concurrent in the analysis of the generated traffic. The removal of these samples therefore makes the shown distributions from concurrent connections in the replays heavier than the one in the original trace.
- They have a heavier tail than the tail of the samples that they joined in connections correctly classified as sequential in the analysis of the generated traffic. The addition of these samples

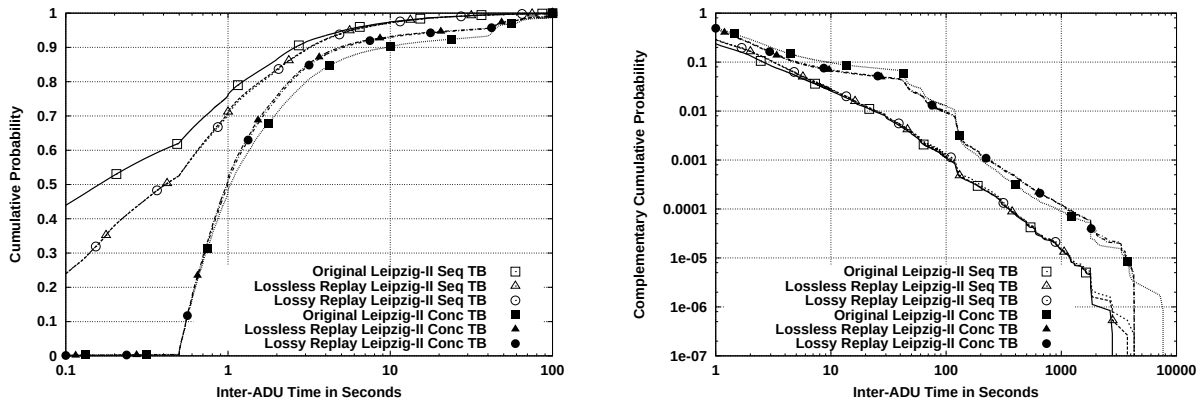

Figure 5.8: Bodies and tails of the TB distributions for Leipzig-II and its source-level trace replays.

therefore makes the shown distributions from the sequential connections in the replays heavier than the one in the original trace.

The distribution TB of quiet times on the initiator side of TCP connections, *i.e.*, between b_i and a_{i+1}, is compared for original and replayed traces in Figure 5.8. The bodies of the distributions show the same kind of mismatch that we discussed for the TA distributions. For values below a few seconds the TB distribution from the replay of sequential connections appears heavier that the original distribution. This is due to the overestimation of quiet times, which becomes less significant as the quiet time becomes larger. We can also observe that the difference in the shortest quiet time is larger for TB than for TA. The reason is not completely clear, but it is probably related to the absence of samples in TB from the large subset of connection vectors with only one epoch. The TB distribution from the replay of concurrent connections appears lighter than the original for values above one second. This is due to concurrent connection misclassification. The much larger number of samples in the distributions for sequential connections makes the impact of the misclassification very small.

Besides the replay of the source-level characteristics of the connections in Leipzig-II, our experiments also involved replaying the network-level parameters measured for each connection in $\mathcal{T}_c$. The left plot in Figure 5.9 compares the distributions of round-trip times extracted from the original and the generated packet header traces. The reproduction was very accurate for sequential connection vectors, and the three distributions exactly lie on top of each other. On the contrary, the distributions for the replayed concurrent connections show a strange jump in probability at 100 milliseconds. The reason for this anomaly, which changed the shape of the rest of the distribution, is unclear.

The right plot of Figure 5.9 compares the distributions of receiver window sizes. Note that the

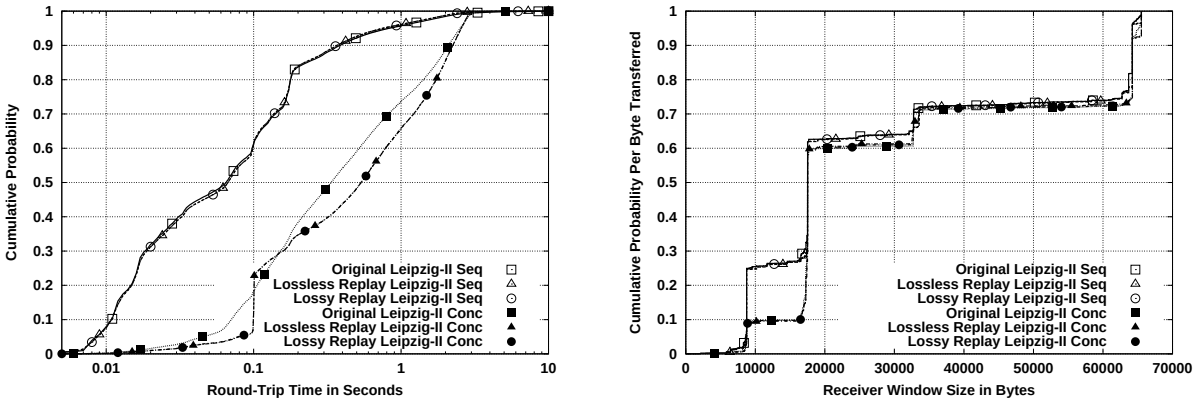

Figure 5.9: Bodies of the round-trip time and receiver window size distributions for Leipzig-II and its source-level trace replays.

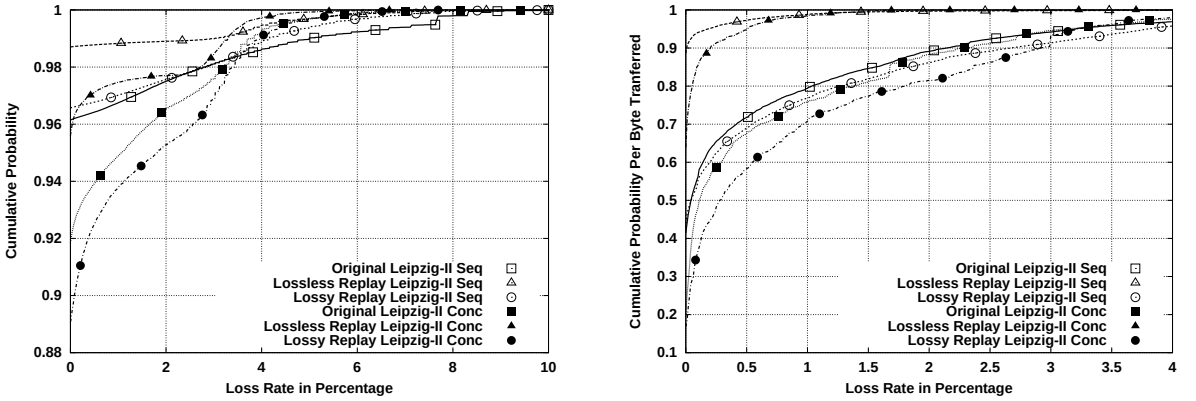

Figure 5.10: Bodies the loss rate distributions for Leipzig-II and its source-level trace replays, with probabilities computed per connection (left) and per byte (right).

probability was computed over the total number of data bytes transferred, to give a better sense of the amount of data associated with each receiver window size. There is an excellent match between the distribution obtained from the Leipzig-II trace and those from its two source-level replays.

The final comparison examines the distributions of loss rates. The left plot of Figure 5.10 shows the distribution of the measured loss rates for the original Leipzig-II trace and its replays. There is a reasonable match between the original and the lossy replays, especially for sequential connections. This is a good result given that *usernet* creates losses by generating random numbers in an independent manner. The small difference is probably explained by a sample size problem in short connections with non-zero loss rates, as discussed in Section 4.1.3, and by concurrent connection misclassification.

Note that we measured some non-zero loss rates in the lossless experiment, in which no artificial losses were introduced. This suggests some problem with the experimental environment, perhaps some network interfaces that were duplicating segments. Such duplicates confuse the loss rate measurement

algorithm, which considers each retransmission a loss event[3]. If duplication is behind our observations, the impact on the experiments would be minimal. True loss slows down TCP, but duplication does not.

The right plot of Figure 5.10 shows the distributions of loss rates per byte, rather than per connection as in the left plot. The CDFs show the probability that each byte had of being carried in a connection with at most the given loss rate. For example, the CDFs for the original sequential connections shows that 80% of the bytes were carried in connections with a loss rate of 1% or less. The CDFs in the right plots are easier to read than those in the left plot, since they are far smoother. They are also more significant, since they pay more attention to the connections that carry more bytes, which are those than have a larger impact on the load of the network. There is a good match between loss rate distributions for the original and the lossy replay. Both the distribution from the replayed sequential connections and the one from replayed concurrent connections are slightly heavier than those from the original traces.

In general, we always observe heavier loss rates in the replays than in the original data. The explanation is the dropping of pure acknowledgment packets, which was discussed in Section 4.1.3. The analysis of the original trace considers only the loss rate of data segments, and not the combined loss rate of data and acknowledgment segments. However, the artificial dropping mechanisms in *usernet* that is used to create per-flow losses is applied to all of the packets in the connections. This means that both data segments and acknowledgment segments are dropped according to the original loss rates of data segments. The dropping of acknowledgment segments can increase the loss rate of data segments in the replay, because missing acknowledgments can trigger unnecessary retransmissions. Every retransmission is considered a loss event, and therefore we have an increase of loss rate in the replays, which makes the measured distributions of (data segment) loss rates heavier for the replays than for the original. It is certainly possible to modify *usernet* to apply the dropping rate to data segments only, but our experiments did not incorporate this refinement. It is somewhat unrealistic to use a biased dropping mechanism, so it would be better to refine the data acquisition algorithm to consider both data and pure acknowledgment losses. Measuring pure acknowledgment loss rates is far more difficult that measuring data segment loss rates. Endpoints may acknowledge every data segment, or every other data segment, and they do so using cumulative acknowledgment numbers, rather than individual sequence numbers as it is done for data segments. It is therefore more difficult to determine when an acknowledgment does not arrive as expected.

[3]This approach could certainly be refined using the IP ID field to distinguish duplications from retransmissions.

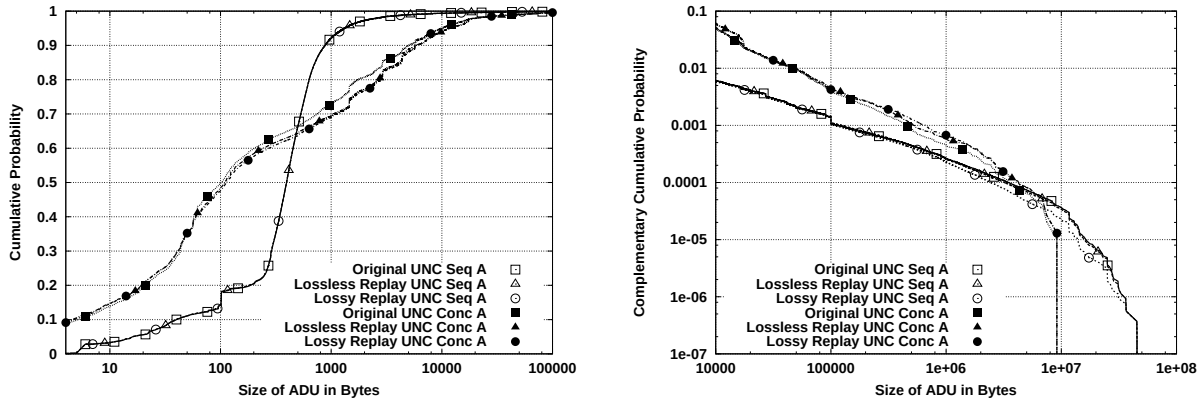

Figure 5.11: Bodies and tails of the A distributions for UNC 1 PM and its source-level trace replays.

5.2.2 UNC 1 PM

The second trace considered in our validation of the source-level trace replay approach is the UNC 1 PM trace. This trace is shorter than Leipzig-II (1 hour *vs.* 2 hours and 45 minutes) but it has much higher throughput, which results in a substantially larger number of samples in the distributions that we will examine in this section. Figure 5.11 compares the A distributions extracted from the UNC 1 PM and its lossless and lossy replays. The bodies of the A distributions from sequential connections reveal no difference between original and generated traces. The tail of the A distribution from the lossy replay is slightly lighter than the one from the original trace and the one from the lossless replay. This difference can be attributed to trace boundaries. Losses make the replay of some connections slower, which can easily result in some connections that do not have time to finish during the replay experiment. This effect is more important for the largest data units, those in the tail of the distribution, since they are the ones that require a substantial amount of time to complete their transmission even without losses.

Concurrent connections show a slightly worse match. This is due to the misclassification problem described in the previous section. As pointed out before, misclassifications are more likely to occur in concurrent connections with small ADUs. These connections have a small number of packets, making the observation of concurrent pairs less likely. As a result, the bodies of the distributions from the replays are slightly heavier, since some fraction of the small ADUs disappeared from the A distribution for concurrent connections. On the contrary, misclassifications had no visible impact on the A distribution for sequential connections. This is because the number of a-type ADUs in sequential connection vectors is much larger than the number of samples from misclassified connections. The tails of the A distributions for concurrent connections show a good agreement.

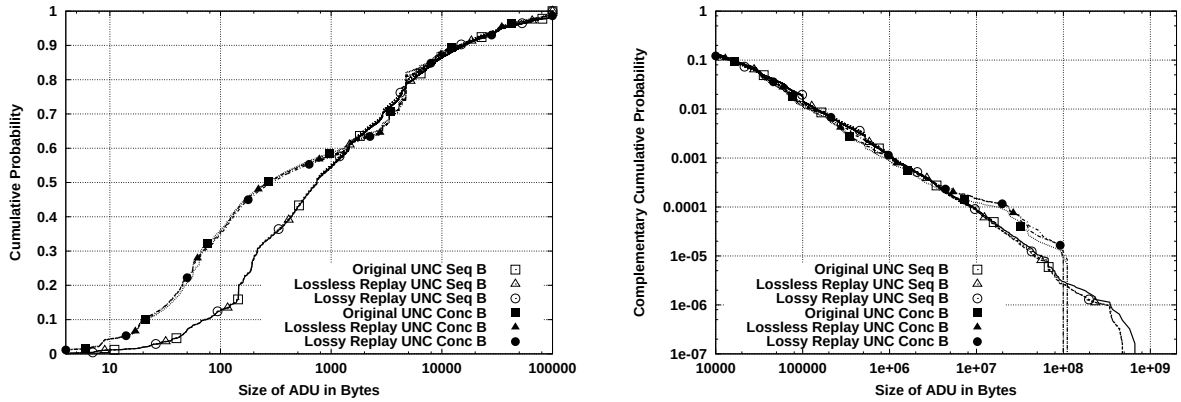

Figure 5.12: Bodies and tails of the B distributions for UNC 1 PM and its source-level trace replays.

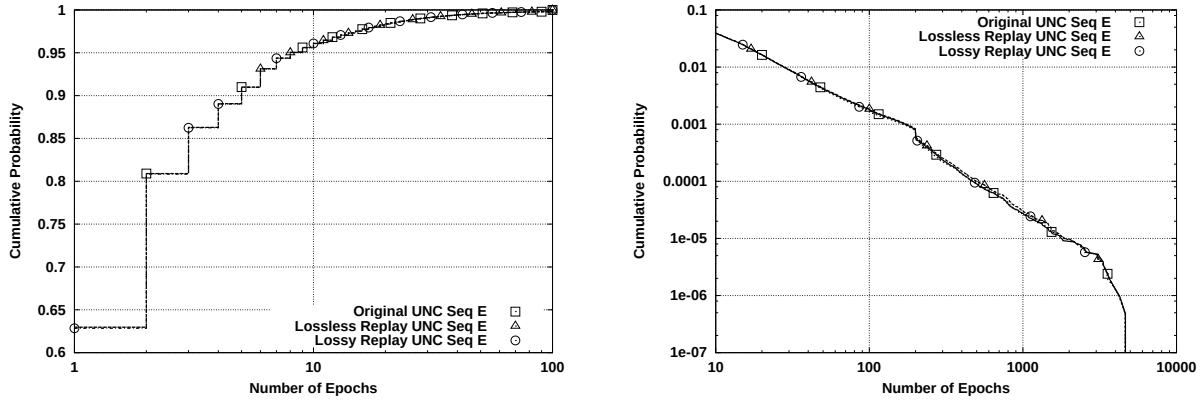

Figure 5.13: Bodies and tails of the E distributions for UNC 1 PM and its source-level trace replays.

The B distributions from the original UNC 1 PM traces and its replays are even closer, as Figure 5.12 shows. We can barely see any differences in bodies of the distributions from concurrent connections and no difference for those from sequential connections. The tails are also very similar, and the slight differences can be explained using the same arguments put forward in the discussion of the A distributions (*i.e.*, trace boundaries and misclassifications).

Figure 5.13 shows an excellent match between the number of epochs in sequential connection vectors measured from the UNC 1 PM traces, and those measured from the replays. The bodies of the distributions are identical, and the tails show only a very minor difference. We therefore observe a better agreement between original and replay for UNC 1 PM than for Leipzig-II (see Figure 5.6).

The plots in Figure 5.14 study the TA distributions. The bodies for sequential connections show an excellent match between the inter-ADU quiet time measured from the original UNC 1 PM trace, and those measured from the generated traces. The bodies for concurrent connections are also very similar.

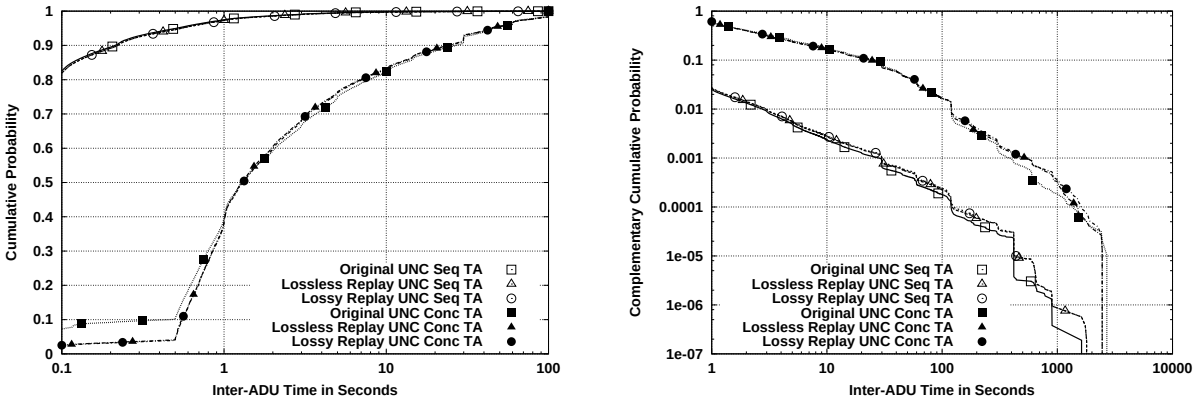

Figure 5.14: Bodies and tails of the TA distributions for UNC 1 PM and its source-level trace replays.

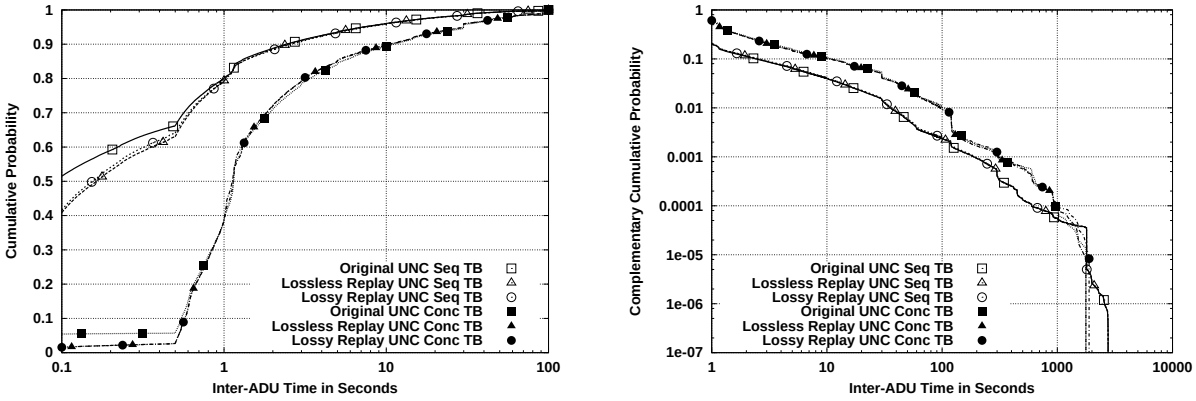

Figure 5.15: Bodies and tails of the TB distributions for UNC 1 PM and its source-level trace replays.

The small difference for the smallest values requires further investigation. We should not see these samples here because our only method for detecting inter-ADU quiet times in concurrent connections is to identify periods of inactivity above 500 milliseconds. We do not observe such a difference for Leipzig-II and Abilene-I. The tails of the distributions are very similar for sequential and concurrent connections. As it was also the case in the data from Leipzig-II shown in Figure 5.7, we observe slightly heavier tails from the replays, which can be explained by misclassifications.

Figure 5.15 shows the bodies and the tails of the TB distributions. Data from sequential connections shows an excellent match for values above 1 second, and even the far tail is very closely approximated. For values below 1 second, we observe that the replays have heavier distributions. This is explained by the quiet time overestimation problem discussed in the analysis of the Leipzig-II results. Concurrent connections also show an excellent match between original and generated traces. The artifact in the smallest inter-ADU quiet times that was observed for the TA distributions from concurrent connections

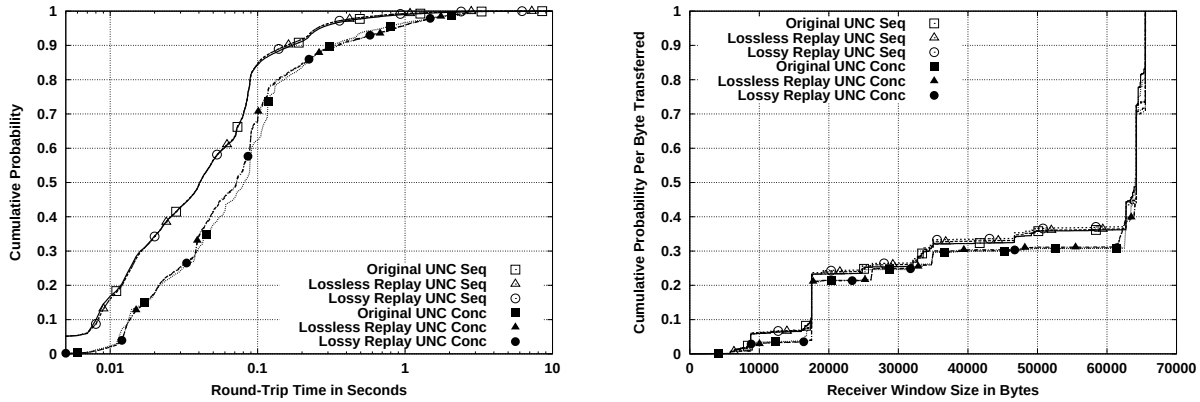

Figure 5.16: Bodies of the round-trip time and receiver window size distributions for UNC 1 PM and its source-level trace replays.

is also present in the TB distributions from concurrent connections.

The next four plots study how closely the replays of UNC 1 PM approximated the network-level parameters observed in the original plot. The left plot of Figure 5.16 shows the distributions of round-trip times. For sequential connections, there was no difference between the round-trip times obtained from the original trace and those obtained from its replays. For concurrent connections, there is only a very small difference, which we can attribute to concurrent connection misclassifications. The large masses of probability for 100 milliseconds observed in the Leipzig-II replays are not present in the UNC 1 PM replays.

Regarding the distribution of TCP receiver window sizes, the plot on the right in Figure 5.16 shows a good match between the original data and the one obtained from the analysis of the generated packet header traces. The tiny difference can again be explained by concurrent connection misclassifications, but it is clear that the replayed traffic accurately captured the use of TCP receiver window sizes.

Figure 5.17 studies the distributions of loss rates rates obtained from original and replayed traffic. As indicated in the analysis of the replays of Leipzig-II, matching loss rate is difficult given the use of independent packet dropping in *usernet* . Consequently, we can consider the approximation of the loss rates shown in the left plot of the figure reasonable, especially in the case of sequential connections, for which many more samples were available. In contrast to these per-connection loss rates, the right plot of the figure shows a substantially closer approximation when loss rate per bytes are considered. Note also that difference between distributions of loss rates for sequential and concurrent connections is far smaller in the case of probabilities per byte.

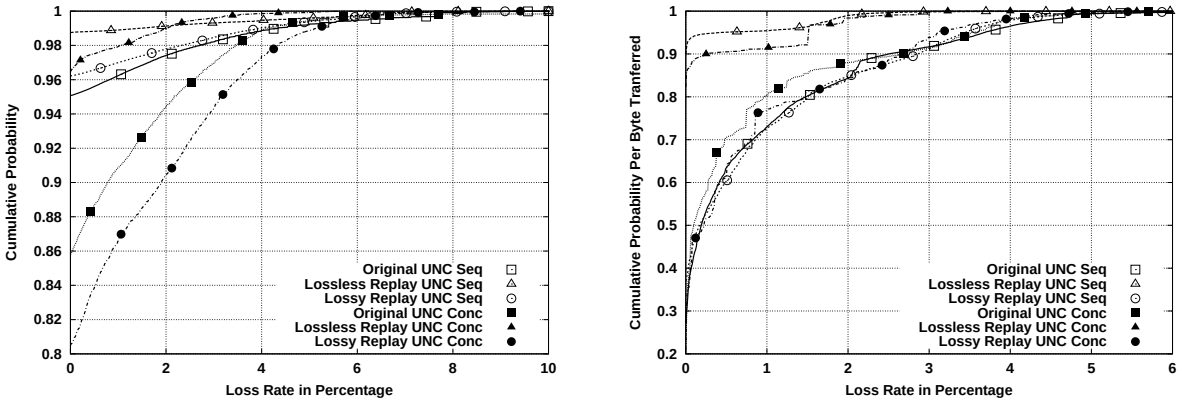

Figure 5.17: Bodies of the loss rate distributions for UNC 1 PM and its source-level trace replays, with probabilities computed per connection (left) and per byte (right).

5.2.3 Abilene-I

We conclude the validation of our source-level trace replay method by comparing the original Abilene-I trace and its lossless and lossy replays. This is the trace with the highest average throughput. Figure 5.18 shows that the A distributions measured from the replayed traces are very similar to those measured from the original trace. Given the completely different A distributions for sequential and concurrent connections, we would expect that any substantial number of misclassified concurrent connections would result in distributions from the replays that significantly diverge from the original distributions. The excellent approximation in this figure, and for the B distributions shown in Figure 5.18, suggest that the number of misclassifications was very small. We also observe a very good match for the tails where the only difference is found for the largest values. In some cases, the replay is slower than the original trace, so some of the largest ADUs may not have had enough time to complete. Adding losses to the replay experiment did not introduce any noticeable difference in the measured distributions, which confirms the robustness of the data acquisition and generation methods to the challenge of lossy environments.

The two plots in Figure 5.19 show that the original distribution of b-type ADU sizes is almost identical to the ones obtained from the lossless and lossy replays. This is true both for the bodies studied in the plot on the left, and for the tails studied in the plot on the right. It is quite difficult to find any region where the distributions differ. It is also clear that the addition of losses to the replay did not modify the sizes of the ADUs in the experiment.

Figure 5.20 shows that the bodies and the tails of the distributions of the numbers of epochs are closely approximated in the source-level replays. There is only a very slight difference in the far tail of the distributions. This could be attributed to a few connections that were replayed more slowly than

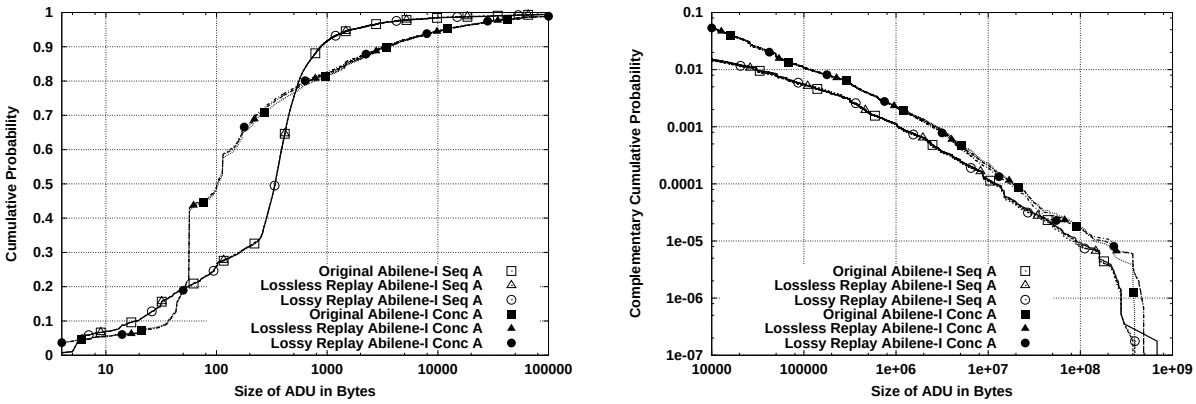

Figure 5.18: Bodies and tails of the A distributions for Abilene-I and its source-level trace replays.

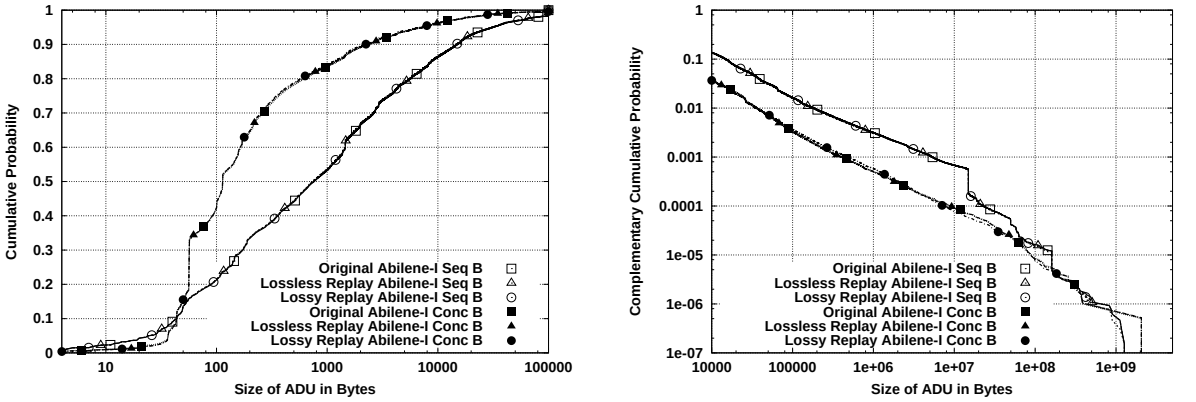

Figure 5.19: Bodies and tails of the B distributions for Abilene-I and its source-level trace replays.

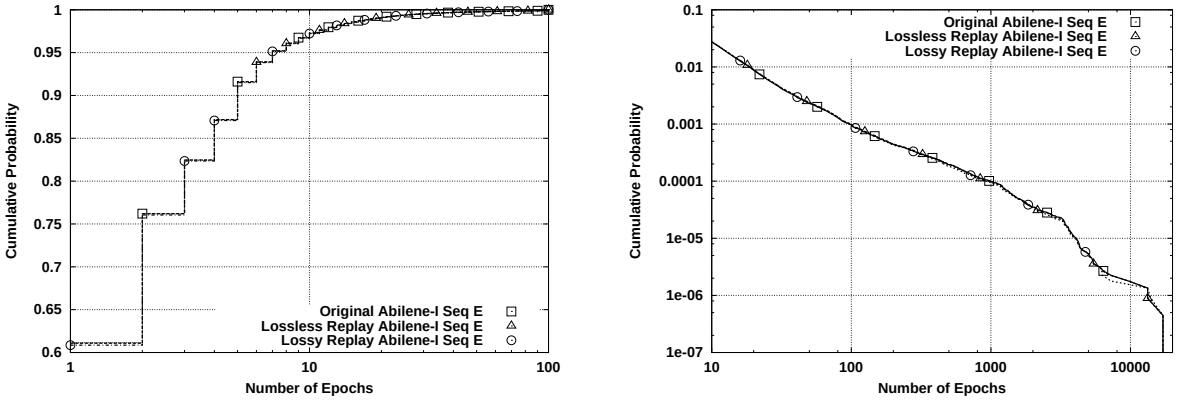

Figure 5.20: Bodies and tails of the E distributions for Abilene-I and its source-level trace replays.

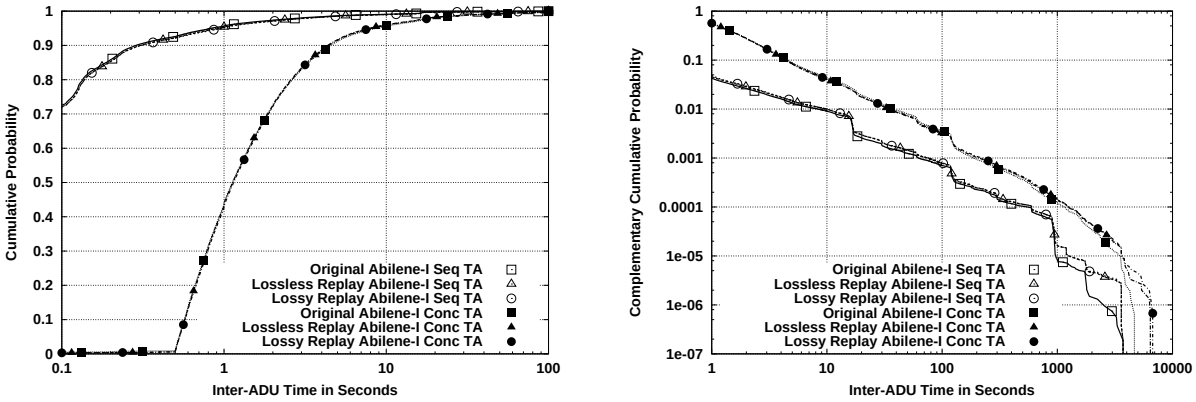

Figure 5.21: Bodies and tails of the TA distributions for Abilene-I and its source-level trace replays.

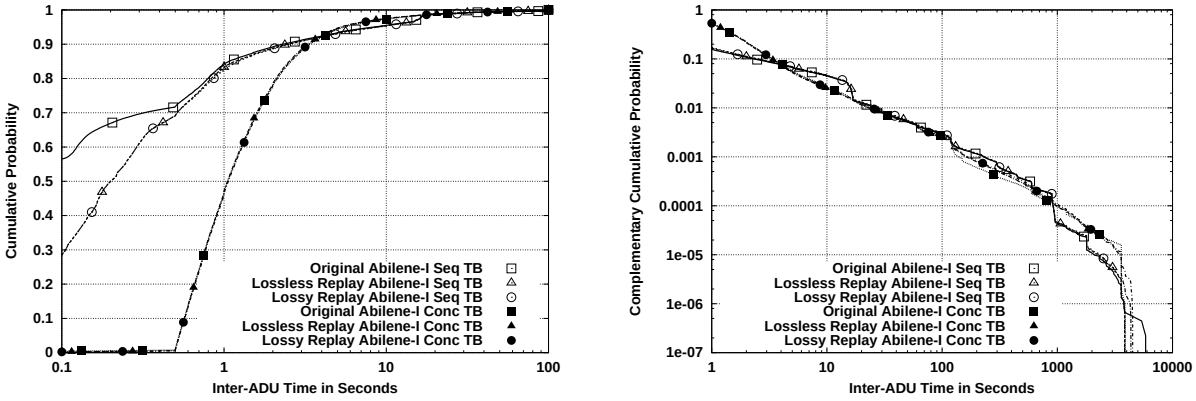

Figure 5.22: Bodies and tails of the TB distributions for Abilene-I and its source-level trace replays.

in the original trace, so they did not have time to complete all of their epochs. Another possibility is that a small number of concurrent connections with a large number of epochs were misclassified. The probabilities in the tail are so small, that even a few samples can create a visible difference.

The quality of the replay of quiet times between ADUs is studied in the next two figures. Figure 5.21 shows that the TA distributions are accurately approximated in the replays. This is true both for sequential and concurrent connections. We only observed a small difference in the far tail, where the replays show slightly heavier values for quiet times above 1000 seconds. As in the case of the E distributions, both experiment boundaries and concurrent connection misclassification can explain the difference.

Figure 5.22 examines the distribution TB of quiet times on the initiator side. As shown on the left, there is an excellent match between the bodies of the distributions from the original trace and those from the replays. The only difference is found in the distributions from sequential connections for quiet times

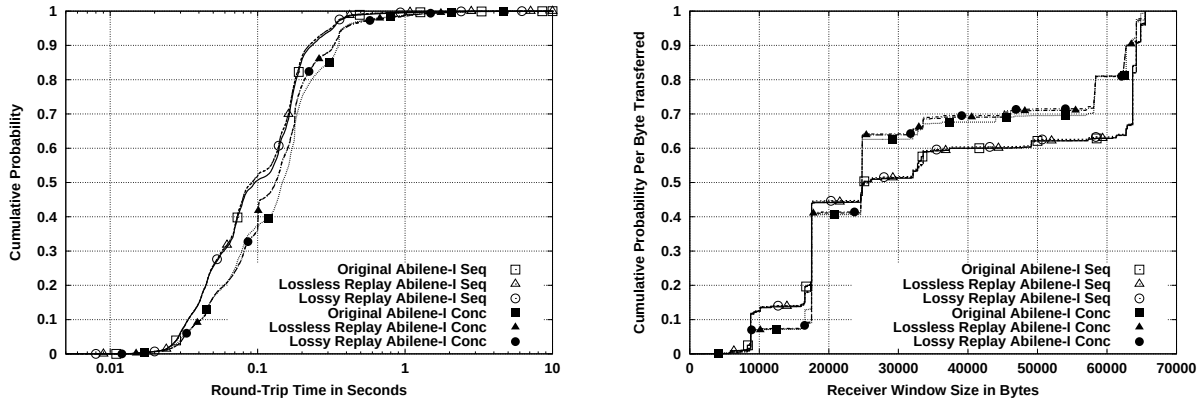

Figure 5.23: Bodies of the round-trip time and receiver window size distributions for Abilene-I and its source-level trace replays.

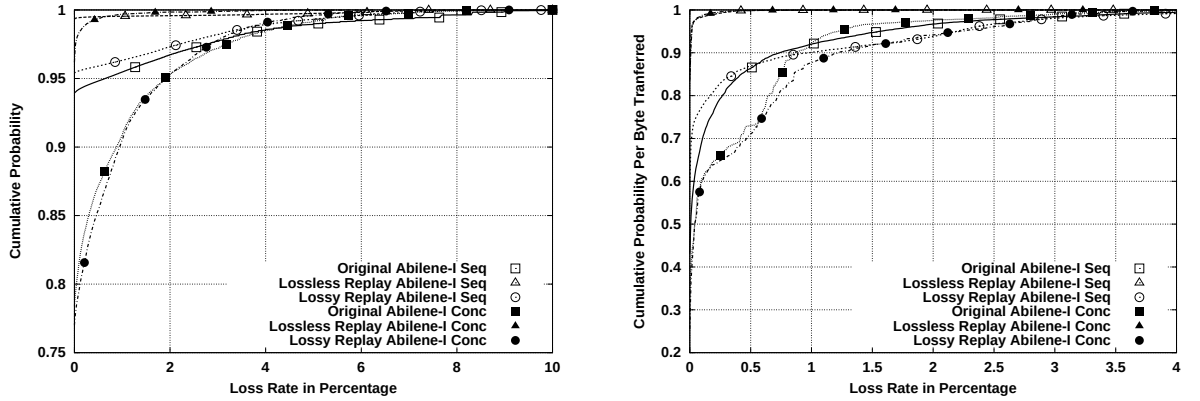

Figure 5.24: Bodies of the loss rate distributions for Abilene-I and its source-level trace replays, with probabilities computed per connection (left) and per byte (right).

below 1 second. The quiet times measured from the replays became increasingly heavier than those from the original trace as their magnitude decreased. This finding is consistent with inaccuracies due to the overestimation of quiet times, since end-host location has a larger impact on the measured quiet time as the magnitude of the application-level quiet time decreases. The tails of the distributions reveal an excellent approximation. It is also important to note that the distributions for concurrent connections do not show the unexpected values below 500 milliseconds that were observed for UNC 1 PM.

The analysis of the round-trip times in Figure 5.23 reveals an excellent match between the original and the replay distributions of round-trip times. The replay of concurrent connections exhibits the same artifact at 100 milliseconds encountered in the replays of the Leipzig-II trace, but the magnitude is far smaller. The distributions of receiver window sizes show very close approximations, with only a small divergence for concurrent connections, which can be easily explained by a small number of misclassifications.

The distribution of loss rates in the lossy replay is very close to the original distribution, as shown in Figure 5.24. The CDFs on the left plot show cumulative probabilities computed per connection, and they reveal a remarkably good match between the original and the lossy replay, both for concurrent and sequential connections. This is significantly better than in the cases of Leipzig-II and UNC 1 PM, which were studied in Figures 5.10 and 5.17. The better match is mostly explained by two characteristics of the original data. First, Abilene-I has the largest fraction of lossy connections, which more than doubles the one in Leipzig-II. This means a wider y-axis that reduces the distance between the distributions in the plot. Second, the heavier distribution of connection sizes in the Abilene-I trace means a larger number of packets, which makes the use of independent drops approximate the intended loss rates more accurately. The right plot shows a good match when the distributions of the per-byte loss rates are considered.

5.3 Summary

This chapter presented our traffic generation method, source-level trace replay. The first step in source-level trace replay is to transform a packet header trace into a set of connection vectors, which describe its source-level behavior using the sequential or the concurrent version of the a-b-t model. Connection vectors also include three network-level parameters, round-trip time, TCP receiver window size and loss rate. The actual traffic generation consists of replaying the characteristics of each connection vector in an accurate manner. We demonstrated the possibility of this approach using an implementation in a network testbed, which includes a distributed traffic generator, *tmix* , that can replay source-level behavior, and coordinate with a packet manipulation layer, *usernet* , to impose specific round-trip times and loss rates to each connection. The approach, and its implementation, was then validated by comparing the statistical characteristics of three traces and those of their replays. This comparison focused on how well the replay preserved the original parameters, *i.e.*, the source-level description and the network-level characteristics.

The validation results showed a good match between original traces and their replays, which confirms the highly accurate reproduction of source-level properties that can be achieved with our approach. The differences, which are shown to be small or nonexistent in every case, are due to the following causes:

- There is no guarantee that the replay of a concurrent connection exhibits measurable concurrency, *i.e.*, that a pair of concurrent data segments can be observed in the generated trace. This results

in connections that are replayed as concurrent but classified as sequential in $\mathcal{T}_c'$, therefore adding spurious samples to the characterization of sequential connections, and removing samples from the characterization of concurrent connections. In general, this affects the comparison of concurrent connections more substantially, since the number of samples from concurrent connections is usually far smaller. This problem is inherent to the form of the concurrent a-b-t model used in this dissertation.

- Our measurement of quiet times tended to overestimate their durations, since it did not compensate for the delay between the end host and the monitor. This difference is only significant for the smallest quiet times, whose magnitude is similar to that of network delays. A possible refinement of our measurement method that would eliminate the overestimation of quiet times and make the replay of quiet times even more accurate, is to subtract the corresponding one-side transit time from each measured quiet time.

- *Usernet* uses independent dropping to simulate losses, and this is not completely accurate. Connections often have too few packets to converge to the intended loss rate per connection. If loss rates per byte are considered, the replay is shown to be very close to the original distribution. Achieving a close approximation of the original loss rate would involve some form of dependent dropping.

- Measured drop rates consider only data segments, but the loss rate simulation also drops pure acknowledgments with the same probability. This makes the distributions of loss rates in the lossy replays slightly above the intended values. Addressing this inaccuracy requires developing a measurement algorithm that can determine the loss rate of pure acknowledgments, which seems rather difficult, or modifying *usernet* to drop only data segments, which is a somewhat artificial solution.

The analysis of the validation results also served us to verify the robustness of our data acquisition and generation method to the introduction of losses with regard to the source-level characteristics. We found very little difference, if any, between the results from the lossless and lossy replays, which confirms the accuracy of the analysis even in the face of packet losses and reordering. TCP timeouts, which can sometimes confuse the heuristic used to split ADUs in the same direction, do not appear to have any significant effect.