CHAPTER 1

Introduction

As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.

— Albert Einstein (1879–1955)

Humankind cannot stand very much reality.

— T. S. Elliot (1888–1965)

Research in networking has to deal with the extreme complexity of many layers of technology interacting with each other in frequently unexpected ways. As a consequence, there is a broad consensus among researchers that purely theoretical analysis is not enough to demonstrate the effectiveness of network technologies. More often than not, careful experimentation in simulators and network testbeds under controlled conditions is needed to validate new ideas. Every researcher therefore faces, at some point or another, the need to design realistic networking experiments, and synthetic network traffic is a foremost element of these experiments. Synthetic network traffic represents not only the workload of a computer network, but also the direct or indirect target of any optimization. For instance, congestion control research focuses on preserving as much as possible the ability of a network to transfer data in the face of overload. Therefore, evaluating a new congestion control mechanism in a transport protocol such as the Transport Control Protocol (TCP) [Pos81] usually requires constructing experiments in which a number of network hosts exchange data using this protocol in an environment with one or more saturated links. The value of the new mechanism is then expressed as a function of the performance of these data exchanges. For example, the new mechanism may be optimized for achieving a higher overall throughput or a more fair allocation of bandwidth.

A fundamental insight, which provides the main motivation for this dissertation, is that the characteristics of synthetic traffic have a dramatic impact on the outcome of networking experiments. For example, a new mechanism that improves the throughput of bulk, long-lasting file transfers in a congested

environment may not improve and may even degrade the response time of the small data exchanges in web traffic. This was precisely the case of Random Early Detection (RED), an Active Queue Management (AQM) mechanism. The original analysis by Floyd and Jacobson [FJ93a] clearly demonstrated the benefits of RED over the basic First-In First-Out (FIFO) queuing mechanism for bulk transfers. In this study, RED queues were exposed to a small number (2–4) of large file transfers. However, a later experimental study by Christiansen *et al.* [CJOS00] showed that this first AQM mechanism degraded the performance of web traffic in highly congested environments. In contrast to the original evaluation, web traffic mostly consists of a very large number of small data transfers, which create a very different workload. The emergence of the web clearly changed the nature of Internet traffic, and made it necessary to revisit existing results obtained under different workloads. The systematic evaluation of network mechanisms must therefore include experiments covering the wide range of traffic characteristics observed on Internet links. It is critical to provide the research community with methods and tools for generating synthetic traffic as representative as possible of this range of characteristics.

The concept of *source-level modeling* introduced by Paxson and Floyd [PF95] constitutes a major influence on this dissertation. These authors advocated for building models of the behavior of Internet applications (*i.e.*, the sources of Internet traffic), and generating traffic in networking experiments by driving network stacks with these application models. The main benefit of this approach is that traffic is generated in a *closed-loop* manner, which fully preserves the fundamental feedback loop between network endpoints and network characteristics. For example, a model of web traffic can be used to generate traffic using TCP/IP network stacks, and the generated traffic will properly react to different levels of congestion in networking experiments. In contrast, *open-loop* traffic generation is associated to models of the packet arrivals on network links, and these models are insensitive to changes in network conditions, and tied to the original conditions under which they were developed. This makes them inappropriate for experimental studies that change these conditions.

The main motivation of our work is to address one important difficulty with source-level modeling. In the past, source-level modeling has been associated with characterizing the behavior of individual applications. While this approach can result in high-quality models, it is a difficult process that requires a large amount of effort. As a consequence, only a small number of models is available, and they are often outdated. This is in sharp contrast to the traffic observed in most Internet links, which is driven by rich *traffic mixes* composed of a large number of applications. Source-level modeling of individual applications does not scale to modern traffic mixes, making it very problematic for networking researchers to conduct representative experiments with closed-loop traffic.

This dissertation presents a new methodology for generating network traffic in testbed experiments and software simulations. We make three main contributions. First, we develop a new source-level model of network traffic, the *a-b-t model*, for describing in a generic and intuitive manner the behavior of the applications driving TCP connections. Given a packet header trace collected at an arbitrary Internet link, we use this model to describe each TCP connection in the trace in terms of data exchanges and quiet times, without any knowledge of the actual semantics of the application. Our algorithms make it possible to efficiently derive empirical characterizations of network traffic, reducing modeling times from months to hours. The same analysis can be used to incorporate network-level parameters, such as round-trip times, to the description of each connection, providing a solid foundation for traffic generation. Second, we propose a traffic generation method, *source-level trace replay*, where traffic is generated by replaying the observed behavior of the applications as sources of traffic. This is therefore a method for generating entire traffic mixes in a *closed-loop* manner. One crucial benefit of our method is that it can be evaluated by directly comparing an original trace and its source-level replay. This makes it possible to systematically study the *realism* of synthetic traffic, in the terms of how well our description of the connections in the original traffic mix reflects the nature of the original traffic. In addition, this kind of comparison provides a means to understand the impact that the different characteristics of a traffic mix have on specific traces and on Internet traffic in general. Third, we propose and study two approaches for introducing variability in the generation process and scaling (up or down) the level of traffic load in the experiments. These operations greatly increase the flexibility of our approach, enabling a wide range of experimental investigations conducted using our traffic generation method.

1.1 Abstract Source-Level Modeling

This dissertation presents a methodology for generating synthetic network traffic that addresses some of the main shortcomings of existing techniques. Figure 1.1 illustrates the levels of detail at which Internet traffic can be studied, providing a good starting point for framing our discussion. We focus on the traffic on a single Internet link, such as the one between the University of North Carolina at Chapel Hill (UNC) and the Internet. We can study the traffic in this link at different levels of detail. The top-most time-line represents traffic observed in the link between UNC and the Internet as a sequence of packet arrivals. This level of detail is known as the aggregate packet arrival level. Here packets from many different connections were interleaved creating a complex arrival process in the network link. In general, TCP traffic accounts for the vast majority of the packets on Internet links (usually between 90%

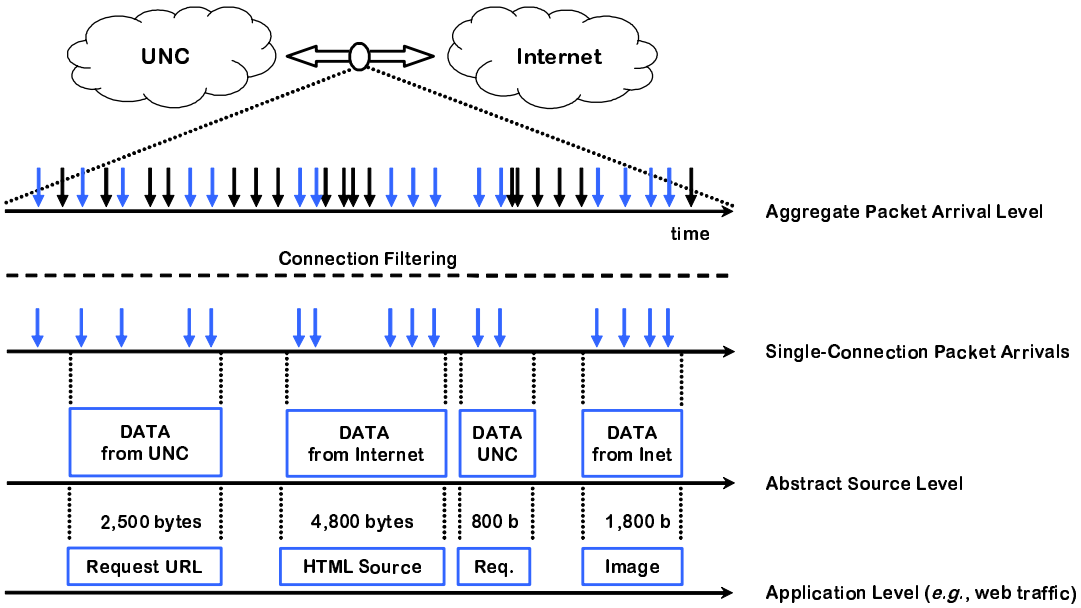

Figure 1.1: Network traffic seen from different levels.

and 95%), which justifies our focus on TCP in this work. The second time-line depicts the packet arrivals that belonged to a single TCP connection. These packets were used to send data back and forth between two network endpoints, one located at UNC, and the other one somewhere on the Internet. The sources of these data are applications running on the endpoints, which rely on the packet switching service provided by the Internet to communicate. Prominent examples of these applications are the World Wide Web, email, file sharing, *etc.* Hundreds of different applications are commonly found on Internet links. The traffic observed at an Internet link is therefore the result of multiplexing the communication of a large number of endpoints driven by a wide range of applications. This dissertation considers the problem of generating traffic in networking experiments that preserves both the aggregate-level and the connection-level properties of traffic observed in a real network link. Note that we restrict ourselves to this most basic form of the problem where only a single link is considered both for observing traffic and for reproducing it in networking experiments. Our findings can certainly be applied to a broader context, *e.g.*, multiple links along a path following the "parking lot topology" [PF95], links in an ISP, *etc.*, but we choose to keep to this problem in its most essential form throughout this dissertation.

As mentioned before, every connection on the Internet is driven by an application exchanging data between two endpoints. It is therefore possible to examine traffic at a higher-level, where the communication is described in terms of *application data units* (ADUs) rather than network packets. This

application level is illustrated in the bottom time-line of Figure 1.1, which reveals that the source of the packets in the second time-line was the exchange of data between a web browser and a web server using a TCP connection. The time-line shows a first ADU of 2,500 bytes, which carried a request for an HTML page. The way the data is organized within this ADU and its meaning is given by the specification of the HyperText Transfer Protocol (HTTP) [FGM+97], which standardizes the exchange of data between web browsers and web servers. The time-line shows a second ADU, sent by the web server to the web browser in response to the first ADU. It carried the actual HTML source code of the page requested by the browser. Its size was 4,800 bytes, which included not only the HTML source code but also an appropriate HTTP header. The time-line shows another pair of ADUs that also corresponded to an HTTP request and an HTTP response, which this time carried an image file. Each ADU is associated to one or more packets in the second time-line. The amount of data in these ADUs and its meaning was decided by the application, while the actual number of packets, their sizes, the need for retransmissions, *etc.*, were decided by lower layers (transport and below).

The application level provides the starting point for the traffic modeling and generation methodology developed in this dissertation. Our approach to traffic generation relies on the notion of *source-level modeling*, advocated by Paxson and Floyd [FP01]. Rather than directly generating packets according to some trace or some packet arrival model, source-level modeling involves simulating the behavior of the applications running on the endpoints and allowing lower layers to control the actual exchange of packets. For example, generating traffic with a source-level model of web traffic means to simulate web browsers and web servers according to statistical models of web page sizes, the durations of user think times and other source-level parameters [Mah97, BC98, SHCJO01].

Modeling traffic at the source level produces descriptions of traffic that are mostly independent of the underlying protocols and network conditions, so they can be used to drive traffic generation in experiments that modify these same protocols and conditions. For this reason, source-level models are also known as *network-independent model*. For example, the size of an HTML page carried in a TCP connection does not change with the degree of congestion (it always has the same number of characters). Therefore, its size is a network-independent property. Lower-level descriptions of traffic, such as characterizations of packet arrivals, are *network dependent*. For example, the rate at which the packets of a TCP connection arrive decreases as the degree of congestion increases, since TCP uses a congestion control algorithm that decreases the sending rate as the loss rate increases. Also, packet losses force TCP endpoints to perform retransmissions. This means that the transmission of the same amount of data at the source-level (*e.g.*, an HTML page) at different times may require different

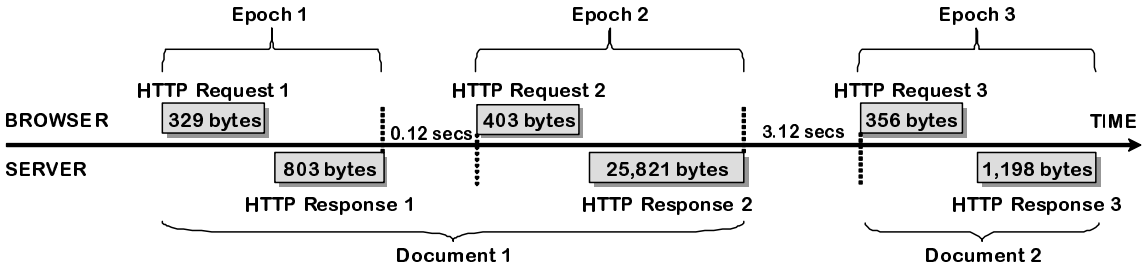

Figure 1.2: An a-b-t diagram illustrating a persistent HTTP connection.

numbers of packets to be transferred, depending on the number of lost packets. A source-level model describes the sizes of ADUs, but not the times at which a connection should lower its sending rate or retransmit a packet. For this reason, the same model can be used to generate traffic under different network conditions, such as low and high levels of congestion. Endpoints generating traffic using these models are able to adapt to each specific set of network conditions in the experiments. This preserves the fundamental feedback loop that exists between endpoints and network conditions. For this reason, this type of traffic generation is said to be *closed-loop*. On the contrary, traffic generated according to lower level models is necessarily *open-loop*. For example, *tcpreplay* [tcpb] can be used to reenact the sending of every packet recorded in a trace, which results in open-loop traffic that is insensitive to the underlying network conditions. This traffic is inappropriate for experiments where network conditions are important, such as the evaluation of congestion control mechanisms.

In the past, source-level modeling has been considered a synonym of application modeling, so researchers have developed a number of application-specific models including models for web traffic, file transferring and other individual applications. This approach is good if one is interested in the traffic generated by a single application (or by a handful of applications). However, if one is interested in realistic traffic mixes, application-specific traffic modeling has some important shortcomings. The first problem is that application specific modeling does not scale well to the large number of applications that form contemporary traffic mixes. For example, the weekly traffic report from Internet2 [Con04] collects separate statistics for more than 80 different applications that make up Internet2 traffic. Using existing technology, it is simply too time-consuming to develop and populate individual models for each application. Moreover, even if we had the resources to examine the behavior of all applications, many applications use proprietary protocols, so painstaking reverse engineering is needed to understand and model their behavior. In addition, Internet traffic evolves quickly, since new applications and improved versions of the existing ones appear very frequently.

This dissertation proposes a more general solution to the source-level modeling and the traffic generation problems. We develop an *abstract model* of network data exchange wherein each connection is described independently of the semantics of the application initiating the connection. This idea is illustrated in the third time-line of Figure 1.1. Here the communication is described in generic terms, simply as a sequence of ADU exchanges between the two endpoints of the TCP connection, without attaching any meaning to the ADUs. Other generic characteristics of traffic include the direction in which the ADUs are sent, from the connection initiator or from the connection acceptor, and the duration of quiet times between ADUs, which are due to user behavior and processing times. These characteristics can generally be used to describe the behavior of any specific application. For example, the ADUs of web traffic are HTTP requests and responses, while the inter-ADU times are user think times and server processing times. The crucial observation is that the sizes of ADUs and the times between them can be measured from the packet traces of two connections *without* knowledge of the behavior of the application driving the connection. This makes it possible to construct a source-level description of the entire set of connections observed in a measured link, instead of only the connections driven by one or a few well-known applications. Any trace of packets traversing a network link can be transformed into an abstract source-level trace, without examining the payload of the packets and without instrumenting the endpoints.

Our approach to source-level modeling results in an abstract representation of a TCP connection using a notation that we call an *a-b-t connection vector*. We also refer to this idea as the *a-b-t model*, in the sense that it provides a mental model for understanding network traffic at the source level, rather than in the sense of a mathematical or statistical model[1]. The term a-b-t is descriptive of the basic building blocks of this model: *a-type* ADUs (a's), which are sent from the connection initiator to the connection acceptor, *b-type* ADUs (b's), which flow in the opposite direction, and quiet times (t's), during which no data segments are exchanged. We will make use of these terms to describe the source-level behavior of TCP connections throughout this dissertation.

Our a-b-t model has a sequential version and a concurrent version. The sequential version applies to connections where the endpoints follow a strict order in their exchange of ADUs. In this version, a TCP connection is described by a vector of epochs $(e_1, e_2, \ldots, e_n)$. Each epoch has the form $e_j = (a_j, ta_j, b_j, tb_j)$, where a_j is the size of an ADU sent from the connection initiator to the connection acceptor, b_j is the size of an ADU sent in the opposite direction, and ta_j and tb_j are inter-ADU quite

[1]Our a-b-t model provides however a good foundation for developing mathematical and statistical models of traffic at the source-level. This dissertation consistently follows a non-parametric approach to traffic modeling. The only exception is the Poisson Resampling method presented in Chapter 7, for which we also offer a more powerful non-parametric alternative, block resampling.

times (during which the endpoints are idle). We call this representation of source-level behavior a *sequential connection vector*. For example, the connection illustrated in Figure 1.2 is represented as

$$((329, 0, 403, 0.12), (403, 0, 25821, 3.12), (356, 0, 1198, 15.3))$$

using the sequential a-b-t model. This connection has three epochs, each carrying one HTTP request/response pair. The first epoch has an ADU a_1 of size 329 bytes, which was sent from the connection initiator (a web browser) to the connection acceptor (a web server), and an ADU b_1 of size 803 bytes, which was sent in the opposite direction. We also observe some quiet times between the ADUs, such tb_2, which had a duration of 3.12 seconds. While Figure 1.2 includes labels for HTTP requests, responses and documents, our a-b-t notation is completely generic.

We consider this TCP connection sequential because only one endpoint sent data to the other one at any point in the lifetime of the connection. It is important to iterate that an ADU is *not* a TCP segment (*i.e.*, TCP packet), but an application message that is independent of its actual network representation as a link-level packet. As such, an ADU can be of arbitrary size, like the smaller $a_1 = 329$ bytes and the larger $b_2 = 25,821$ bytes in the previous example. The transferring of a_1 would usually involve a single TCP segment, but it is also possible that this segment gets duplicated, or lost and then retransmitted. In this case, the TCP endpoint sending a_1 would result in the generation of two or more segments carrying this ADU. Our notation would still describe this part of the TCP connection as a single 329-byte ADU, and not as the sequence of TCP segments used to transfer the data. Similarly, transferring $b_2 = 25,821$ bytes requires a minimum of 18 TCP segments in a path without loss and with a regular Maximum Segment Size (MSS) of 1,460 bytes (the one derived from Ethernet's Maximum Transmission Unit (MTU) of 1,500 bytes, after subtracting 20 bytes for the IP header and 20 bytes for the TCP header). It may require many more segments in a lossy environment, or in a path with a lower MTU. However, these details are irrelevant at the abstract source level, where b_2 captures the need of one of the endpoints to send 25,821 bytes of data, and this need is independent of the way in which the data is transferred by the network. Our modeling is therefore network-independent, which makes it suitable for generating closed-loop traffic.

While most TCP connections are driven by applications that follow a sequential pattern of ADU exchanges, we can also find cases in which the two endpoints send data to each other at the same time. This is illustrated in Figure 1.3 using a BitTorrent [Coh03] connection, where we can see ADUs whose transmission overlaps in time (*i.e.*, the ADUs are exchanged concurrently). This pattern is certainly less

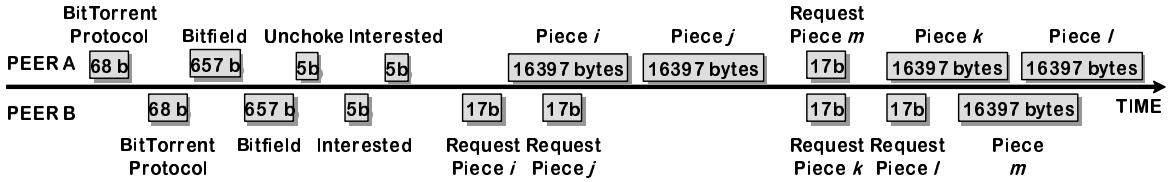

Figure 1.3: A diagram illustrating the interaction between two BitTorrent peers.

common that the sequential one, but it is supported in important protocols like HTTP/1.1 (pipelining), NNTP (streaming mode) and BitTorrent. Our analysis shows that while the fraction of connections with concurrent data exchanges is usually small, (17.4%), such concurrent connections often carry a significant fraction (15%-35%) of the total bytes seen in a trace, and hence modeling these connections is critical if one wants to generate realistic traffic mixes.

To represent concurrent ADU exchanges, the actions of each endpoint are considered to occur independently of each other. Thus each endpoint is a separate source generating ADUs that appear as a sequence of epochs following a unidirectional flow pattern. Formally, this means that we represent each connection as a pair (α, β) of connection vectors of the form

$$\alpha = ((a_1, ta_1), (a_2, ta_2), \ldots, (a_{n_a}, ta_{n_a}))$$

and

$$\beta = ((b_1, tb_1), (b_2, tb_2), \ldots, (b_{n_b}, tb_{n_b})),$$

where a_i and b_i are sizes of ADUs sent from the initiator and from the acceptor of the TCP connection respectively, and ta_i and tb_i are quiet times between the ADUs. We call this representation of source-level behavior a *concurrent connection vector*. Unlike the sequential version of the a-b-t model, this representation does not capture any causality between the two directions of a TCP connection. As a consequence, traffic generated according to this version of the model usually exhibits a substantial number of concurrent data exchanges.

The a-b-t model provides a simple yet expressive way of describing source-level behavior in a generic manner that is not tied to the details of any application. In addition, this non-parametric model was designed to incorporate quantities (ADU sizes, ADU directionality, and inter-ADU quiet time duration) that can be extracted from packet header traces in a efficient, accurate manner. We can easily imagine more complex and expressive models of TCP connections for which no efficient data acquisition algorithm

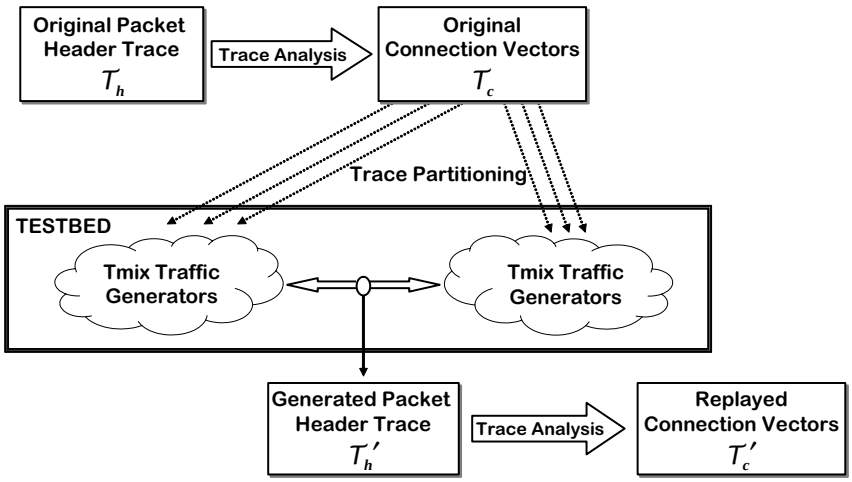

Figure 1.4: Overview of Source-level Trace Replay.

exists, or models that deal with characteristics of source-level behavior that cannot be extracted purely from packet headers. In the case of the a-b-t model, we have developed a data acquisition algorithm that relies on TCP sequence numbers for measuring ADU sizes, and on the packet arrival timestamps obtained during trace collection to determine inter-ADU quite times. Our algorithm constructs a data structure in which TCP segments are ordered according to their *logical data order*, *i.e.*, the order in which data must be delivered to the application layer of the receiving endpoint. In reconstructing this logical order for each connection, we have developed methods for dealing with network pathologies such as arbitrary segment reordering, duplication and retransmission. Furthermore, when the data segments in a TCP connection cannot be ordered according to the logical data order, we can classify the connection as concurrent with certainty. Our data structure supports both sequential (*i.e.*, bidirectional) and concurrent (*i.e.*, unidirectional) ordering, making it possible to extract ADU sizes and quiet times with a single pass over the segments of a TCP connection found in a trace. The analysis can be performed in $O(sW)$ time, where s is the number of data segments in the connection and W is the maximum size of the TCP window (which bounds the maximum amount of reordering).

1.2 Source-Level Trace Replay

Our abstract source-level modeling of TCP connection provides a solid foundation for generation traffic mixes in simulators and network testbeds. We propose to generate traffic using *source-level trace replay*, as illustrated in Figure 1.4. Given a packet header trace $\mathcal{T}_h$ collected from some Internet link, we

first use our data acquisition algorithm to analyze the trace and describe its content as a collection of connection vectors $\mathcal{T}_c = \{(T_i, C_i)\}$, where T_i is the relative start time of the i-th TCP connection, and C_i is the sequential or concurrent connection vector corresponding to this connection. The basic approach for generating traffic according to $\mathcal{T}_c$ is to replay every connection vector C_i. Each connection vector C_i is replayed by starting a TCP connection precisely at C_i's relative start time T_i, and transmitting the measured sequence of ADUs (a_j and b_j) separated in time by the inter-ADU measured quiet times (ta_i and tb_i). In this dissertation, we evaluate a specific implementation of this approach for FreeBSD network testbeds, where traffic is generated using a tool we developed called *tmix* .

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 implies the need to simulate the behavior of applications, using regular network stacks to actually translate source-level behavior into network traffic. In particular, our experiments use an implementation which relies on the standard socket interface to reproduce the data exchanges in each connection vector. Generating traffic in this manner is closed-loop in the sense that it preserves the feedback mechanism in TCP, which adapts its behavior to changes in network conditions, such as loss and receiver saturation. In contrast, packet-level trace replay, the direct reproduction of $\mathcal{T}_h$, is an open-loop traffic generation method in the sense that TCP control algorithms are not used during the generation, and hence the traffic does not adapt to network conditions.

The evaluation of our methodology consists of comparing the original trace $\mathcal{T}_h$ and the synthetic trace $\mathcal{T}_h'$ obtained from the source-level trace replay. Validating our traffic generation method consists of transforming $\mathcal{T}_h'$ into a set of connection vectors $\mathcal{T}_c'$, using the same method used to 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 $\mathcal{T}_h$. There are however some differences that are explained by the nature of the model and our measurement methods.

The direct comparison of $\mathcal{T}_h$ and $\mathcal{T}_h'$ also provides a way to study the accuracy of our approach in terms of how well traffic is described by the a-b-t model. This is however a subtle exercise. The actual replay of $\mathcal{T}_c$, which creates $\mathcal{T}_h'$, necessarily requires the selection of a a set of network-level parameters, such as round-trip times and TCP receiver window sizes, for each TCP connection in the source-level trace replay. The exact set of generated TCP segments and their arrival times is a direct function of these parameters. As a consequence, if we conduct a source-level trace replay using arbitrary network-level parameters, we obtain a $\mathcal{T}_h'$ with little resemblance to the original $\mathcal{T}_h$. The replayed a-b-t connection

vectors may be a perfect description of the source behavior driving the original connections, but the generated packet-level trace $\mathcal{T}_h'$ would *still* be very different from the original $\mathcal{T}_h$. To address this difficulty, our replay incorporates network-level parameters individually derived from each connection in $\mathcal{T}_h$. We have also incorporated methods for measuring three important network-level parameters (round-trip time, TCP receiver window size and loss rate) into our analysis and generation procedure. While this set of parameters is by no means complete, it does include the main parameters that affect the average throughput of a TCP connection found in a trace. This enables us to generate traffic in a closed-loop manner that approximates measured traces very closely.

Incorporating network-level properties is important, but it is critical to understand the main shortcoming of this approach. The goal of our work is not to make the generated traffic $\mathcal{T}_h'$ identical to the original traffic $\mathcal{T}_h$, which could be accomplished with a simple packet-level replay. As mentioned before, packet-level replays generate traffic that does not adapt to changes in network conditions, resulting in open-loop traffic. Our goal is to develop a closed-loop traffic generation method based on a detailed characterization of source behavior. Traffic generated in a closed-loop manner can adapt to different network conditions, which are intrinsic when evaluating different network mechanisms. Our comparison of $\mathcal{T}_h$ and $\mathcal{T}_h'$ is only a means to understand the quality of traffic generation method, where quality is considered to be higher as the original trace is more closely approximated. If enough parameters of the original traffic are accurately measured and incorporated into the traffic generation experiment, we expect to observe a great similarity between $\mathcal{T}_h$ and $\mathcal{T}_h'$. On the contrary, if we are missing some important parameters, we expect to observe substantial differences between traces.

By construction, traffic generated using source-level trace replay can *never* be identical to the original traffic. The statistical properties of original packet header traces are the result of multiplexing a large number of connections onto a single link, and these connections traverse a large number of different paths with a variety of network conditions. It is simply not possible to fully characterize this environment and reproduce it in a laboratory testbed or in a simulation. This is both because of the limitations of passive inference from packet headers, and because of the stochastic nature of network traffic. Source-level trace replay can never incorporate every factor that shaped $\mathcal{T}_h$, and therefore differences between $\mathcal{T}_h$ and $\mathcal{T}_h'$ are unavoidable. Still, finding a *close* match between an original trace and its replay, even if they are not identical, constitutes strong evidence of the accuracy of the a-b-t model and the data acquisition and generation methods we have developed. It also demonstrates the feasibility of generating realistic network traffic in a closed-loop manner that resembles a rich traffic mix.

1.3 Trace Resampling and Load Scaling

As long as the network setup of a simulation or testbed experiment remains unchanged, the source-level trace replay of a connection vector trace $\mathcal{T}_c = \{(T_i, C_i)\}$ always results in traffic that is similar to the original trace. Every replay contains the same number of TCP connections behaving according to the same connection vector specification and starting at the same times. Only tiny variations are introduced on the end-systems by changes in clock synchronization, operating system scheduling and interrupt handling, and at switches and routers by the stochastic nature of packet multiplexing. Source-level trace replay has therefore two desirable properties:

- The quality of the synthetic traffic can be evaluated by *directly comparing* synthetic and original traffic. This makes it possible to study the accuracy of the analysis methods and the generation system with complete freedom, using any metric that can be derived from real traffic. In contrast, more abstract methods based on parametric models of traffic are inherently stochastic and therefore more difficult to evaluate. For such methods, it is less obvious whether the observed difference between the traffic generated using the parametric model and the original traffic from which the model derives should be admitted.

- The generation of the synthetic traffic is fully *reproducible*. A researcher can expose a collection of network protocols and mechanisms to exactly the same closed-loop traffic, which provides the right foundation for fair comparative studies. In contrast, stochastic variation in the traffic generated using parametric models is often difficult to control. For example, experiments with models that rely on heavy-tailed distributions converge very slowly to comparable conditions, as discussed by Crovella and Lipsky [CL97].

While these properties are important, the practice of experimental networking often requires to introduce controlled variability in the generated traffic for exploring a wider range of scenarios. This motivates the development of methods that manipulate $\mathcal{T}_c$ in order to generate different traffic that still *resembles* the original one. Furthermore, developing a statistically sound way of manipulating $\mathcal{T}_c$ is essential for generating traffic with different levels of offered load. This manipulation to match a target offered load is a very common need in experimental networking research. This is because the performance of a network mechanism or protocol is often affected by the amount of traffic to which it is exposed. Therefore, rigorous experimental studies frequently require to generate a complete range of target loads.

In this dissertation, we propose two flexible methods for introducing variability in traffic generation experiments. In both cases, the set of connection vectors in $\mathcal{T}_c$ is randomly resampled, resulting in a new set $\mathcal{T}_c'$ that preserves the aggregate source-level characteristics of the original traffic. In our first method, *Poisson Resampling*, we construct a new connection vector trace $\mathcal{T}_c'$ by randomly resampling connections from $\mathcal{T}_c$, and assigning them exponentially distributed inter-arrival times. As a result, connections in $\mathcal{T}_c'$ arrive according to a Poisson process. In the second method, *Block Resampling*, we resample blocks (groups) of connections rather than individual connections. This method results in a more realistic connection arrival process, which matches the substantial burstiness observed in real traces. In more technical terms, Block Resampling preserves the moderate long-range dependence found in real connection arrival processes, while Poisson Resampling results in a short-range dependent connection arrivals process. This difference is demonstrated in our experimental evaluation of the two methods. In addition, the evaluation shows that the duration of the resampling block creates a trade-off between shorter blocks (which increase the number of distinct resamplings) and long-range dependence (which disappears for short blocks). Our analysis demonstrates that block durations between 1 and 5 minutes offer the best compromise.

Researchers often need to conduct a set of experiments with a range of different traffic loads. When using a traditional source-level model, *e.g.*, a model of web traffic, researchers have to first conduct a preliminary experimental study to determine how the parameters of the model, *e.g.*, the number of user equivalents, affect the generated load [CJOS00, LAJS03, KcLH$^+$02]. This is usually known as the *calibration* of traffic generator. Our resampling methods eliminate this common need for calibrating traffic generators, since the resampling process can be controlled to match a specific target load (*i.e.*, generated load is known a priori). In the case of Poisson Resampling, this is accomplished by changing the mean arrival rate of connections. In the case of Block Resampling, offered load is manipulated using block thinning (*i.e.*, subsampling) and block thickening (*i.e.*, combining blocks). Our work reveals that load scaling cannot be based simply on controlling the number of connections. Such an approach frequently results in offered loads that are far from the target, because the number of connections in a resample is not strongly correlated with the offered load represented by these connections. We address this difficulty by developing *byte-driven* versions of Poisson Resampling and Block Resampling, which scale load using a running count of the total data in the resampled trace $\mathcal{T}_c'$. Unlike the number of connections, the total amount of data in $\mathcal{T}_c'$ is strongly correlated to traffic load offered by $\mathcal{T}_c'$. Our experiments confirm that byte-driven resampling is highly accurate, eliminating the common need for calibrating traffic generators.

1.4 Thesis Statement

This dissertation considers the following thesis:

1. An abstract source-level model can describe in detail the entire set of TCP application behaviors observed in real networks.

2. Descriptions of abstract source-level behavior can be empirically derived from packet header traces in an efficient, accurate manner.

3. Traffic generation based on this abstract source-level modeling results in synthetic traffic that is realistic and suitable for experimental networking research.

4. The abstract source-level model of a trace can be manipulated to introduce statistically valid variability in the generated traffic and also to accurately match a target offered load while preserving application characteristics.

1.5 Contributions

We highlight the following contributions from this dissertation:

- We develop the concept of abstract source-level modeling and the a-b-t notation for describing the source-level behavior of entire traffic mixes. We identify a fundamental dichotomy in source-level behavior between connections that exchange data sequentially and connections that exchange data concurrently. Our a-b-t notation includes a sequential version and a concurrent version that makes it possible to appropriately describe these two types of behaviors.

- We formulate a formal test of concurrency that can be applied to the packet headers of any TCP connection, and that does not suffer from false positives. This enables us to accurately classify connections as sequential or concurrent. We show that only a small fraction of TCP connections (less than 4% in our traces) exchange data concurrently, but that these TCP connections account for a substantial fraction (up to 32%) of the total traffic.

- We present an efficient algorithm for transforming a packet header trace into a collection of sequential and concurrent a-b-t connection vectors. Given a TCP connection for which we observe s segments and that has a maximum receiver window size of W, the asymptotic cost of our algorithm

is $O(sW)$. We demonstrate that this algorithm is accurate using traffic generated from synthetic applications (*i.e.*, with known characteristics).

- We develop source-level trace replay, a closed-loop traffic generation method that uses a-b-t connection vectors as a non-parametric model of network traffic. One key benefit of this approach is the possibility of directly comparing original and generated traffic, which we use to evaluate the "realism" of our traffic generation approach. This comparison requires us to incorporate some network-level parameters (round-trip times, maximum receiver window sizes, and possibly loss rates) into the traffic generation. These parameters can be measured from packet header traces. We pay special attention to passive round-trip time estimation in our data acquisition, developing the concept of One-Side Transit Time and studying the impact of delayed acknowledgments on passive round-trip time estimation.

- We implement our traffic generation method in a network testbed, developing a new distributed traffic generation tool, *tmix* . We use this implementation to study the results of a large collection of trace replay experiments, evaluating the need for detailed source-level modeling and the impact of losses on measured network traffic. Our results demonstrate that detailed source-level modeling is often required for accurately approximating real traffic, which demonstrates that source-level behavior is a major factor shaping Internet traffic. The most substantial differences are observed for the number of active connections and the number of packet arrivals per unit of time. Byte arrivals per unit of time and long-range dependence do not improve so consistently with the use of detailed source-level modeling. We also show that losses had only a secondary effect in our traces, but they are not negligible when comparing original and generated traffic.

- We present two trace resampling algorithms which can be used to derive new traces from an existing one, preserving its statistical characteristics at the source-level. Our comparison of the two methods reveals that the observed long-range dependence in connection arrivals has no apparent impact on the long-range dependence of packet and byte arrivals.

- We demonstrate the need for byte-driven rather than connection-driven resampling in order to accurately scale offered loads, and develop byte-driven versions of our two resampling methods. This approach eliminates the need for the experimental calibration of traffic generators (which study the relationship between the parameters of the generator and the offered traffic load).

- Our entire methodology makes it possible to conduct networking experiments with closed-loop synthetic traffic derived from real traces in an automated manner. This eliminates the need for

painstaking parametric modeling.

1.6 Overview

Chapter 2 presents a review of the state-of-the-art in synthetic traffic generation. We first expand our discussion of packet-level traffic generation and data acquisition, and then examine source-level traffic generation more in depth. We review the literature on application-specific modeling, discussing models of web traffic and other applications, and also consider several approaches for generating traffic driven by more than one application. We also discuss existing methods for controlling the traffic load created in networking experiments. The chapter finally considers some research efforts addressing implementation issues.

Chapter 3 discusses abstract source-level modeling, presenting several examples of real applications and how their behavior can be described using our a-b-t notation. We also present our measurement algorithm for transforming a packet header trace into a collection of sequential and concurrent a-b-t connection vectors. The chapter also includes a validation of the measurement method using synthetic applications, and a measurement study that examines the statistical properties of the a-b-t connection vectors extracted from five real traces.

Chapter 4 focuses on network-level measurement. We first describe our methods for measuring round-trip times, window sizes and loss rates, and an evaluation of their accuracy. While this set of parameters is by no means complete, it does include the main parameters that affect the average throughput of a TCP connection found in a trace. The second part of Chapter 4 describes the network-level metrics that we consider in the evaluation of our traffic generation method: packet and byte throughput time series, their marginal distributions, wavelet spectra, Hurst parameter estimates and time series of active connections.

Chapter 5 describes source-level trace replay and our implementation in a network testbed. We present a validation of this implementation using the source-level trace replays of five traces. For each trace, we study the a-b-t connection vectors extracted from the original traces and those found in replays with and without packet losses at the network links. The results demonstrate the accuracy of our approach, and also uncover some difficulties, which are in some cases inherent to the a-b-t model and its passive method of data acquisition.

Chapter 6 examines the results of several source-level trace replay experiments. Our analysis compares original traces and their source-level trace replays using the rich set of metrics introduced in Chapter 4, revealing a remarkably close approximation. This study also includes a comparison of traffic generated with the a-b-t model and with a simplified version that "disables" source-level modeling, which is shown to perform well for some metrics and poorly for others. As in the previous chapter, we also consider experiments with and without artificial losses, showing that loss did not have a dominant impact on the characteristics of the original traffic. In general, our results provide a strong justification of our source-level modeling approach, demonstrating that the closed-loop replay of a-b-t connection vectors closely resembles real traffic.

Chapter 7 presents our two resampling methods, Poisson Resampling and Block Resampling. These methods enable the researcher to introduce controlled variability in source-level trace replay experiments, without sacrificing reproducibility. In addition, we consider the problem of load scaling, *i.e.*, how to control the resampling process to obtain a new trace with a target offered load. Our work demonstrates that this task can be accomplished by keeping track of the total number of data bytes in the resampled trace, but not by keeping track of the number of connections. Our scaling methods eliminate the common need for running a preliminary study to calibrate the traffic generator.

Chapter 8 presents our conclusions and discusses future work.