Semantic Compression of TCP Traces

G. Istrate[1], A. Hansson[1], S. Thulasidasan[1], M. Marathe[2], and C. Barrett[2]

[1] CCS-5, Los Alamos National Laboratory, Los Alamos NM 87545, USA
[2] Virginia Bioinformatics Institute, Blacksburg VA 24061, USA

Abstract. We propose a new methodology, RESTORED, for model-based storage and regeneration of TCP traces. RESTORED provides significant data compression by exploiting semantics of TCP. Experiments show that RESTORED can achieve over 10,000-fold compression ratios for some really large input connections, while still being able to recover several structural and QoS measures.

1 Introduction

Traffic measurement and monitoring faces the important challenge of *data storage*; due to the large amount of information, it is simply not feasible to collect all traces. A natural approach is to keep only a fraction of the packets that traverse a given link and estimate connection characteristics from the stored packets. A number of approaches to succinctly storing network data have been proposed. For example, Cisco System's NETFLOW technology is based on the simple idea of aggregating packet information to compute flow characteristics [1], i.e., time is partitioned into slots, and a router records the total number of packets it handles in any given time slot, the total number of bytes, etc. Further, Cisco has proposed a sampled version that records information based on every Nth packet. A slightly more advanced approach is to employ sampling rate adaptation. Unfortunately, these solutions lack clear justification from a scientific standpoint: a fine time granularity is not able to provide effective data compression (since a lot of data has to be recorded), whereas too coarse time granularity leads to poor characterization of network dynamics. The latter issue is also a significant weakness of SNMP counters [2]. Other monitoring tools, such as GIGASCOPE [3], supports complex performance queries, but scalability remains a major bottleneck.

Approaches that store only a subset of traffic data face the obvious problem that the stored information might not be sufficient to capture the meaningful characteristics of TCP. For instance, while a couple of measures of quality of service (e.g. throughput) could probably be estimated by storing every Nth packet, it is likely that most measures cannot be inferred in this way. A more principled, *model-based approach* is needed.

Recent years have seen substantial advances in understanding and modeling the intricate nature of TCP traffic. Aggregate traffic can display fractal [4] and multifractal [5] characteristics at small timescales. For large enough timescales technological constraints make these correlations disappear, so that there is no long-range dependence [6]. In the presence of congestion, aggregate traffic characteristics are approximately Poisson [7]. On the other hand, individual connections have a much simpler structure; many of their aspects can be modeled by Markov chains [8]. The progress in modeling temporal aspects of TCP traces has not been matched by corresponding advances in modeling the dynamics of packet IDs. This is unfortunate, since the dynamics of packet ID can influence the overall connection dynamics: TCP is a protocol that tries to maintain packet sequence integrity, and will attempt to do so by controlling the senders' congestion window. Thus a complete understanding of TCP dynamics requires a new approach in which *packet reordering* plays a central role. As convincingly argued by Bennett, Partridge, and Shectman [9], packet reordering has many severe effects on TCP performance (see also [10]). In conclusion, receiver side models of TCP should capture both temporal and reordering aspects of TCP traces.

The main goal of this paper is to *propose a new approach called* RESTORED, REceiver-oriented STOchastic REgeneration of packet Dynamics, *for model-based storage and regeneration of TCP traces*. RESTORED uses a compression scheme based in TCP semantics, and *produces traces that are provably equivalent to the original trace with respect to a rigorously defined notion of trace equivalence*. The core functionality of RESTORED can be summarized as follows: *(i) Trace Collection:* Data is collected for each session at each destination node. *(ii) Trace Compression:* We do not require exact storage of the trace but allow lossy compression. This allows us to greatly boost the compression ratio. *(iii) Generation of Synthetic Traces from the Summary Data:* The idea is that the synthetic data retains most of the dynamic information inherent in the original packet streams. *(iv) QoS Analysis Based on the Generated Traffic:* Since the synthetic data sequences are compatible with the collected ones, we can analyze QoS measures in much the same way as we would have done on the original data.

Network Data: We use real network traces obtained by monitoring network traffic, as well as synthetic network traces generated by simulation. The real packet traces were collected during August 2001 at the border router of the Computer Science Department, University of California, Los Angeles, CA. See http://lever.cs.ucla.edu/ddos/traces for details. In particular, we have used five TCP traces, called TRACE5, TRACE7, TRACE8, TRACE9, TRACE10. We found that most of the connections in these traces are very short. For example, TRACE7 consists of 245,718 connections, but 60% of them contain only one or two packets, 80% contain at most 10, and 98% contain at most 100 packets. This is, of course, in line with the observation that a small fraction of the flows accounts for a large percentage of the total traffic [11, 12]. Estan and Varghese have argued that, for many applications, knowledge of these "heavy hitters" is sufficient [13]. Since our aim is to highlight nontrivial network dynamics, we chose to study only connections with at least 100 packets. Still, there are enough connections to allow meaningful analysis: TRACE5 contains 7582 such connections, TRACE7 contains 5662, TRACE8 contains 7936, TRACE9 contains 7612, and TRACE10 contains 3399. We also employed NS-2 to generate a synthetic TCP trace called NS-2-long, to benchmark the compression performance of RESTORED. We simulated TCP Reno with SACK and delayed ACKs; all sessions were persistent FTP data connections with a fixed payload of 1072 bytes. We simulated the classical dumbell topology with multiple sources and sinks sharing the same bottleneck link, 201 connections in total. The links connecting the sources and the sinks to the bottleneck routers had a bandwidth of 100 Mbps and a delay of 10 ms. The bottleneck link bandwidth and delay were 100 Mbps and 50 ms. The TCP send/receive buffers were set to 64 KB, and packet drop rates were controlled by limiting the output queues of bottleneck routers to at most 50 packets. All queues were drop-tail FIFO queues. The connection start times were uniformly spread on an interval between 0 and 0.5 seconds. We ran the simulation until 842 million packets had been sent (not counting ACKs), which translates to about 4.2 million packets per connection.

2 The Macroscopic Model

Our model consists of two parts: (i) A Markovian model that captures TCP dynamics at *macroscopic* time scales. (ii) A fine-grained model that completes the macroscopic view of TCP packet reordering to *microscopic* time scales. In this section discuss we discuss the first component of the model, the Markovian model for macroscopic time scales. Let us first consider the packet IDs. Since our objective is to model packet reordering rather than data fragmentation, we make the simplifying assumption that all packets have identical payload. This allows a bijective mapping from TCP sequence numbers to packet ID numbers, with the convention that the smallest sequence number is mapped to ID 1. For simplicity, we omit discussing the slow start phase; our model can readily be extended to include it. Consider the following motivating example: a receiver may observe the following packet stream (where we only display the ID numbers of the packets, and not their arrival times)

$$\underbrace{1\ \ 2}_{\mathcal{O}}\quad \underbrace{4\ \ 5\ \ 6\ \ 3}_{\mathcal{U}}\quad \underbrace{8\ \ 9\ \ 10\ \ 7}_{\mathcal{U}}\quad \underbrace{11\ \ 12\ \ 13}_{\mathcal{O}} \tag{1}$$

The order of the IDs corresponds to the order in which packets arrive at the destination, and in our case, we see that packets 3 and 7 arrive out of order. Since TCP guarantees to deliver an ordered packet stream to the application layer, it follows that there is a need for packet buffering. One can, consequently, classify the received packets into two types: those that can be immediately passed to the application layer, and those that are temporarily buffered before delivery. In our example, packets 4, 5, and 6 are temporarily buffered, and the buffer cannot be flushed until packet 3 is received. Likewise, packets 8, 9, and 10 are temporarily buffered, and the buffer is flushed at the arrival of packet 7. A packet that marks the end of a sequence of consecutively buffered packets will be called a *pivot packet*. Packets that can be immediately delivered to the application layer are trivially pivots. In our example, packets 1, 2, 3, 7, 11, 12, and 13 are thus all pivots. This definition suggests a coarsened view of TCP with two states:

State $\mathcal{O}$: The *ordered state*, in which packets can be immediately passed to the application layer
State $\mathcal{U}$: The *unordered state*, in which there is reordering and buffering.

Each occurrence of State $\mathcal{O}$ is followed by one or more occurrences of State $\mathcal{U}$. Explicitly incorporating time, one can provide a high-leven description of a TCP connection by a sequence of triples $(s_1, p_1, t_1)\,,\ (s_2, p_2, t_2)\,,\ \ldots\,,$ where $s_n \in \{\mathcal{O}, \mathcal{U}\}$ is the state descriptor, $p_n > 0$ is the number of packets received in state s_n, and $t_n > 0$ is the time spent in state s_n.

Coarsening TCP connections at the level of pivot packets *provides an operational motivation for assuming that the observed traffic characteristics are stationary*: after a pivot packet has been received and the buffer has been flushed from the point of view of the receiver TCP is "in the same state" in circumstances with the same congestion window and the same number of packets in transit. In contrast, some models of network traffic assume second-order stationarity without any plausible motivation—at least it is not entirely clear at what time scale this assumption is warranted [14]. In fact, for large enough time scales, traffic parameters may even exhibit nonstationarity [15].

The high-level view of network traffic will be our first component for modeling TCP sequences compatible with a given trace. More specifically, we propose a Markovian model for the previous sequence. This model is schematically illustrated in Fig. 1, and formally specified in Fig. 2. The Markovian model above yields an inference algorithm which, in turn, enables us to reconstruct synthetic TCP traces up to the coarsened level of pivot packets. Section 3 extends this definition to a complete model of TCP.

Let us now statistically validate the macroscopic model. One tool that is employed is the sample autocorrelation function, $\hat{\rho}_X(h)$, of a time-series $\{X_n\}$. This function is found by first computing the sample mean, $\hat{m}_X$, and the sample autocovariance function, $\hat{\gamma}_X(h)$, associated with $\{X_n\}$. Specifically, if we let $x_1, x_2, \ldots, x_N$ be observations of $\{X_n\}$, the sample mean is just the average, $\hat{m}_X \equiv N^{-1} \sum_{n=1}^{N} x_n$, the sample autocovariance function is the standard estimate of the autocovariance function, $\hat{\gamma}_X(h) \equiv N^{-1} \sum_{n=1}^{N-|h|} (x_{n+|h|} - \hat{m}_X)(x_n - \hat{m}_X)$, and finally, the sample autocorrelation function is obtained by normalization, $\hat{\rho}_X(h) \equiv \hat{\gamma}_X(h)/\hat{\gamma}_X(0)$. First, we have to validate Claim I, i.e. we have to assess that the sequence of states can be modeled using a Markov chain. To test this hypothesis, it is enough to show that the time-series $N_{\mathcal{U}}$, the number of consecutive occurrences of State $\mathcal{U}\}$, can be viewed as a sequence of independent samples drawn from a certain distribution (possibly different for different packet streams). We will test independence by performing the *sample autocorrelation test* [16]. This test states that approximately 95% of the sample autocorrelations of an i.i.d. sequence $x_1, x_2, \ldots, x_N$ should fall between the bounds $\pm 1.96/\sqrt{N}$ for large N (assuming finite variance observations). To test Claim I we thus formulate the following null hypothesis: observations of the time-series $N_{\mathcal{U}}$ are independently sampled from a unique underlying distribution. Based on the sample autocorrelation test, this hypothesis is then rejected with a confidence of 95% if more than 5% of the sample autocorrelation coefficients fall outside the bounds $\pm 1.96/\sqrt{N}$. We chose to analyze the five UCLA traces that were described in the introduction. For the robustness of the sample autocorrelation test we follow the recommendation provided by Box and Jenkins, who suggest that N should be at least about 50 [17]. A number of packet streams must then be discarded. Still, 236 streams are retained for TRACE5, 292 for TRACE7, 266 for TRACE8, 317 for TRACE9, and 63 for TRACE10. The percentage of packet streams that fail the sample autocorrelation test (and thus disprove the null hypothesis) is displayed in the topmost panel of Table 1 below.

To validate Claims II–III we need to test that (i) there is no correlation in the time-series $\{p_n, t_n\}$ associated with State $\mathcal{O}$ (or State $\mathcal{U}$), and that (ii) there is no correlation between consecutive observations of (p_n, t_n) associated with a transition from State $\mathcal{O}$ to State $\mathcal{U}$ (or from State $\mathcal{U}$ to State $\mathcal{O}$). In order to bring statistical evidence for Claim II in the definition of the Markovian model, we first consider the four time-series $P_{\mathcal{O}}$, the number of packets in State $\mathcal{O}\}$, $P_{\mathcal{U}}$, the number of packets in State $\mathcal{U}\}$, $T_{\mathcal{O}}$, the time spent in State $\mathcal{O}\}$, and $T_{\mathcal{U}}$, the time spent in State $\mathcal{U}\}$. We should test these four time series for lack of correlation. This is done by using the sample autocorrelation test, and the null hypothesis is that observations of the time series are independently sampled from four unique distributions. Once again, we analyzed the five UCLA traces, and the results are presented in the middle panel of Table 1. To complete the statistical evidence for Claim II in the definition of the Markovian model, we have to test for lack of correlation between characteristics of two different consecutive states. The test we employ is the *nonparametric Spearman test* for linear correlation between components of a bivariate time-series [18]. As null hypothesis, we assume that the two components of the four bi-variate time-series $P_{\mathcal{O}\to\mathcal{U}}$, (the number of packets in state $\mathcal{O}$, the number of packets in next state $\mathcal{U}$), $P_{\mathcal{U}\to\mathcal{O}}$, (the number of packets in state $\mathcal{U}$, the number of packets in next state $\mathcal{O}$), $T_{\mathcal{O}\to\mathcal{U}}$ (the time spent in state $\mathcal{O}$, the time spent in next state $\mathcal{U}$), $T_{\mathcal{U}\to\mathcal{O}}$, (the time spent in state $\mathcal{U}$, the time spent in next state $\mathcal{O}$), are independent. For the UCLA traces, the bottom panel of Table 1 presents the percentage of packet streams for which the test disproves the null hypothesis of independence with a confidence of 95%. From the experimental data displayed in Table 1, we conclude that the macroscopic model passes the statistical tests for an overwhelming majority of the investigated streams. Thus, at least as a first-order approximation, the macroscopic model captures TCP dynamics. Of course, this is not really surprising: the dynamics of the TCP congestion window is nonlinear, while our tools (autocorrelation, etc.) are linear. Our result only show that existing correlations (if any) are subtle enough not to be visible using linear statistics.

3 The Microscopic Model

We will now describe the details of the microscopic model. Let us begin our discussion with the packet IDs. Recall that there are two microscopic schemes; one for each state of the macroscopic model. In State $\mathcal{O}$, the ID sequencing is trivial, and apart from knowledge that we are in State $\mathcal{O}$, no additional information is required. In State $\mathcal{U}$, on the other hand, by definition, packets arrive out of order, and structural properties of the reordering events will guide the modeling. More precisely, the basis of our scheme consists of building a dictionary of frequent, well-structured reordering events arising in the observed packet sequences. Consider the example sequence in (1) and its two unordered phases, $4\ 5\ 6\ 3$ and $8\ 9\ 10\ 7$. These two events are not identical with respect to the number of inversions: three ordered packets precede a fourth packet with lower ID in the first one, while the pattern is more complicated for the second one. However, there is an important way in which the two sequences are similar: if we assume that every packet was ACKed and, furthermore, we employ simple ACKs (not SACK) then *the*

$$\begin{tikzcd}
\text{State } \mathcal{O} \arrow[r, bend left, "P_{\mathcal{O}};T_{\mathcal{O}}"] & \text{State } \mathcal{U} \arrow[l, bend left, "P_{\mathcal{U}};T_{\mathcal{U}}"] \arrow[loop right, "D"]
\end{tikzcd}$$

Fig. 1. Macroscopic model of packet dynamics.

THE MACROSCOPIC MARKOV MODEL

Claim I: The sequence of states, $s_1, s_2, \ldots$, is generated according to the following process: Each occurrence of State $\mathcal{O}$ is followed by a number of consecutive occurrences of State $\mathcal{U}$, independently sampled from an underlying distribution D on $\mathbb{N}$.

Claim II: There exist distributions $P_{\mathcal{O}}$ and $P_{\mathcal{U}}$ on $\mathbb{N}$ associated with State $\mathcal{O}$ and State $\mathcal{U}$, respectively, such that for all $n \geq 1$, the number of packets p_n in state s_n is obtained by sampling from the distribution in the set $\{P_{\mathcal{O}}, P_{\mathcal{U}}\}$ that is associated with state s_n.

Claim III: There exist distributions $T_{\mathcal{O}}$ and $T_{\mathcal{U}}$ on $\mathbb{R}_+$ associated with State $\mathcal{O}$ and State $\mathcal{U}$, respectively, such that for all $n \geq 1$, the time t_n spent in state s_n is obtained by sampling from the distribution in the set $\{T_{\mathcal{O}}, T_{\mathcal{U}}\}$ that is associated with state s_n.

Fig. 2. Specification of the macroscopic model.

sequences of ACKs sent in response to receiving those packets are similar: 3 3 3 7 for the first sequence, 7 7 7 11 for the second one. They are identical modulo a translation in the packet IDs. This motivates the following important notion:

Definition 1. *Two packet sequences A, B are* behaviorally equivalent *(written $A \equiv_{beh} B$) if the sequences ACKs sent in response to receiving the two sequences are identical.*

Behavioral equivalence is a desirable property from the standpoint of TCP modeling: indeed, TCP is a receiver-driven protocol. For behaviorally equivalent traces A and B *the receiver will act identically on A and B*. Assuming similar network conditions, this should make the senders behave in a similar way. So, by regenerating a trace that is behaviorally equivalent to the original trace we are able, indeed, to capture an important part of TCP dynamics.

We will achieve further compression in the microscopic stage by defining a many-to-one mapping from packet sequences in the unordered states to integer "sketches". By the previous discussion, a desired property of this mapping is that behaviorally equivalent sequences are mapped into the same "sketch." Defining a map with this property can be done in several ways, and (as we plan to discuss in a subsequent paper) the right map to use depends on the particular measure of reordering we want to preserve. In this paper we offer a simple solution, achieved by computing the minimum *buffer size*, denoted MBS, which is the size of the smallest buffer large enough to store all packets that arrive out of order, if we reserve space for not yet received packets. Let us clarify this with a formal definition:

Table 1. Percentage of Streams Failing the Statistical Independence Tests

	$N_{\mathcal{U}}$	$P_{\mathcal{O}}$	$P_{\mathcal{U}}$	$T_{\mathcal{O}}$	$T_{\mathcal{U}}$	$P_{\mathcal{O}\to\mathcal{U}}$	$P_{\mathcal{U}\to\mathcal{O}}$	$T_{\mathcal{O}\to\mathcal{U}}$	$T_{\mathcal{U}\to\mathcal{O}}$
TRACE5	0.51	2.71	0.08	2.26	2.00	1.74	2.00	1.72	2.00
TRACE7	1.34	1.80	1.30	4.60	2.30	2.79	3.21	3.00	3.54
TRACE8	1.42	0.30	0.20	0.78	1.10	7.90	8.20	3.15	3.60
TRACE9	2.78	0.90	1.50	1.30	3.40	7.30	7.60	2.78	3.33
TRACE10	5.35	1.70	0.00	1.70	1.50	1.50	1.50	1.00	1.38

Definition 2. *Let* LOP *be the largest ordered packet ID that has been received (at any given time), i.e., the largest packet ID such that packets in the range* 1 *to* LOP *have all been received (*0 *if packet* 1 *has not yet been received), and let* LRP *be the largest received packet ID (at any given time) (*0 *if no packets have been received yet). We then define the* minimum buffer size (MBS) *as* MBS $=$ LRP $-$ LOP*. Also, the* MBS pattern *associated with a sequence* A *of packet IDs is defined as a time-series of* MBS *values computed* after *each packet in* A *is received. In other words, the* MBS pattern *represents the time evolution of the* MBS.

Returning to our example, we arrive at an identical buffer pattern, $4\ 5\ 6\ 3 \rightarrow 2\ 3\ 4\ 0$, $8\ 9\ 10\ 7 \rightarrow 2\ 3\ 4\ 0$. Since the sequence of packet IDs in State $\mathcal{U}$ always ends with a pivot packet, the last entry in any pattern is 0, and could be omitted in the encoding scheme.

The buffer size in Definition 2 has a natural interpretation in terms of TCP semantics: When all packets have the same payload p, MBS is linearly related to the size of the advertised window, $\mathrm{AdvertisedWindow} = \mathrm{MaxRcvBuffer} - p \cdot$ MBS, where $\mathrm{AdvertisedWindow}$ and $\mathrm{MaxRcvBuffer}$ are defined in [19]. Having introduced a simple encoding scheme for the reordering events, a decoding map can be computed just as easily, because of the following result

Proposition 1. *Consider an* MBS *pattern* B *that consists of* N *positive integers,* $B = (B_1, B_2, \ldots, B_N)$*, and denote by* B_{max} *the largest integer in* B*. There exists an integer* C *and an algorithm of complexity polynomial in* $N + B_{\text{max}} + C$ *that takes* B *as input and (i) decides whether there exists a corresponding reordering pattern,* $A = (A_1, A_2, \ldots, A_N)$*, and (ii) computes such a pattern if one exists. The smallest integer in* A *is* $C + 1$*.*

Because of space constraints we omit the mathematical proof of Proposition 1 (see companion paper [20] for details). Consider now the following notion of equivalence between TCP traces: Two sequences of packets A and B are MBS *equivalent* (written $A \equiv_{buf} B$) if the sequences of values of buffer sizes MBS corresponding to the two sequences are identical. We can now restate Proposition 1 as follows: the polynomial time algorithm from Proposition 1 produces ID sequences that are MBS equivalent to the original one. Proposition 1 also guarantees the semantic similarity of the regenerated ID sequences to the original ones. The reason is the following result:

Proposition 2. *[20] Suppose that the receiver uses simple ACKs and acknowledge every packet. Then any sequences* A *and* B *that are* MBS *equivalent are also behaviorally equivalent.*

4 Regeneration of Synthetic TCP Connections

The algorithm in the previous section allows us to regenerate a sequence of packet IDs that provides a significant compression with respect to the original sequence, while being also reasonably plausible as a sequence of received IDs. Indeed, in the ordered state the sequence of packet IDs is increasing, mirroring the increase of the congestion window, while in the unordered state the sequence corresponds to a reordering pattern that occurred in the real trace. The drawback of the model so far is that it does not include regeneration of the *arrival times*. In this section we show how to transform the model into one that jointly generates packet IDs and arrival times. The higher level of the regeneration algorithm will run the Markov chain whose parameters were inferred in the learning phase. This allows regeneration of both the sequence of packet IDs (as detailed in the previous section) and of the time spent by the sequence in each state. On the other hand, we now have to "fill in the details," by assigning arrival times to the regenerated packets.

PACKET ID ALGORITHM

Learning Phase

1. In State $\mathcal{O}$, learn the distribution $P_{\mathcal{O}}$.
2. In State $\mathcal{U}$, instead of learning the distribution $P_{\mathcal{U}}$, learn the *distribution of reordering patterns*, P_{MBS}.

Regeneration Phase

1. In State $\mathcal{O}$, sample from $P_{\mathcal{O}}$ and generate a corresponding number of ordered packet IDs.
2. In State $\mathcal{U}$, sample from P_{MBS} and use the inverse mapping from Proposition 1 to generate a corresponding sequence of IDs.

PACKET ARRIVAL TIME ALGORITHM

Learning Phase

1. Learn the distributions D_{inter} and D_{intra} of inter- and intra-cluster times.
2. Learn the rate r of the best-fit exponential distribution for cluster length in State $\mathcal{U}$.

Regeneration Phase

1. Use D_{inter} and D_{intra} to generate inter-cluster and intra-cluster times, respectively.
2. Simulate the finite state machine from [21] to keep track of the value of the congestion window CWND.
3. In State $\mathcal{O}$: group packets into clusters according to the additive-increase mechanism of TCP and the values of p and CWND.
4. In State $\mathcal{U}$: group packets into clusters assuming an exponential distribution of clusters with rate r.

Fig. 3. Algorithms for learning and regeneration of packet IDs and arrival times.

It is important to realize that *by including time spent in various states in the details of the macroscopic model we are already able to capture some temporal characteristics of TCP traffic*. To substantiate this statement we first discuss a really naive approach for arrival time reconstruction. As we show in Section 4.1, even this approach is, however, good enough to recover some basic measures of QoS, such as connection *throughput*. We will next refine the approach, using the results of [21], in order to further incorporate some of the semantical aspects of TCP dynamics. It is fairly clear that even on the receiver side inter-packet times are not random, but display significant correlations. Indeed, an inspection of the data reveals the fact that packets arrive in *clusters*, that are correlated with the dynamics of the congestion window. Being able to group packets in clusters allows us to divide inter-packet arrival times into *intra-* and *inter-cluster* times. We will further make the simplifying assumption that *inter- and intra-cluster times* *are independent samples from a given distribution, one for each of the two types of times*. Thus, in the learning phase we infer distributions D_{inter} and D_{intra} of inter- and intra-cluster times, respectively. The naive approach we implemented is to *assume that cluster sizes are exponentially distributed*. In the learning phase we infer the parameter of the exponential distribution, used in the regeneration phase to decide whether the next packet is from the same or from a different cluster. This approach does not capture the dynamics of the congestion window, but the requirement that the learning/regeneration algorithms be executable on-the-fly severly limits the range of approaches we can take. Also, as demonstrated by Section 4.2, at least some QoS measures are well captured by this approach. This naive approach is conceptually simple, and easy to implement. It is likely to not be adequate for "microscopic" measures. In particular one should not expect to capture packet clustering, and measures of QoS (e.g. *jitter*) that depend on it.

One can refine the model to a certain extent to further capture aspects of TCP semantics without making very specific assumptions about network influence on TCP dynamics. The refined model will share the same macroscopic features with its simpler version (in particular it will recover throughput just as well as the simple one). In order to regenerate meaningful traces the algorithm will track the value of the congestion window of the trace regenerated so far. This has been accomplished by a heuristic from [21] that we incorporate in our approach. Applying this algorithm provides a good solution to the clustering problem in the ordered phase of RESTORED. Indeed, *in this phase it is reasonable to assume that packets sent in a cluster arrive together as a cluster and in the same order*. Given the additive increase congestion-control mechanism of TCP, this assumption is enough to group packets into clusters that correspond to receiving a whole congestion window. The connection between clustering and the dynamics of the congestion window is only valid in the absence of congestion (in the ordered phase). In the presence of reordering, packet drops and repeats, and faced with potentially different ACK mechanisms, no principled solution seems entirely natural without further assumptions on reordering. Therefore, for the unordered phase we employ the simpler approach outlined previously. The regeneration of arrival times thus takes the form displayed in the bottom panel of Fig. 3.

4.1 Compression and Regeneration Performance

In this section we first present results concerning the performance of RESTORED in compressing traces, followed by comparisons of original vs. regenerated traces with respect to four measures of QoS: throughput and three reordering metrics. Table 2 presents compression ratios for the five TCP traces recorded at UCLA, as well as for the long TCP trace generated by NS-2. Since we have chosen to store the reordering patterns of the observed packet IDs using semantic, *lossless* compression, it is interesting to first see how well RESTORED is able to compress the ID part (before also considering times, for which we have suggested a more compact description). The third column of Table 2 lists the ratio between the size of the original ID sequences and the size of the dictionary (which stores encoded patterns and their associated frequencies). It can be seen that even this simplistic framework is able to provide decent compression. For aggregate data the compression ratio should be even better due to the succinct modeling of arrival times. To boost the overall compression we prune the dictionary by simply removing extremely infrequent codewords. This way we arrive at the ratios listed in the fourth column of Table 2. In the general case, pruning of the dictionary can be done online using techniques for dealing with *iceberg queries*, well-documented in the database literature.

4.2 Recovering Throughput

We ran the implementation of our naive algorithm on the five sets of real-life connections. For each connection C in one of the traces we reconstructed one sample connection $R(C)$ and computed the throughput ratio, defined as $Q(C) = Throughput(C)/Throughput(R(C))$. For perfect throughput recovery the value of $Q(C)$ should be 1. In Table 3 we display the concentration of throughput ratios $Q(C)$ around the ideal value 1.

Despite using a method that discards a lot of information (achieving compression rates of the order of 10,000), in most cases our throughput estimates differ by no more than 10% from the true values of the throughput. We feel this is acceptable, given the stringent compression requirements of our method, and has the potential of further being improved if we incorporate some of the *nonlinear* correlations present in TCP.

4.3 Recovering Reordering Metrics

The *inversion spectrum* of a trace is simply the distribution of inversions in the observed reordering patterns. For two packet IDs p_i and p_j with associated indices i and j, we define an inversion as $p_i - p_j$ if $p_i > p_j$ and $i < j$. For example, the sequwnce of packet IDs 2 3 3 1 has one inversion of size 1 $(= 2 - 1)$ and two inversions of size 2 $(= 3 - 1)$. Of course, this is not the only way to characterize inversions; an important alternative (see e.g. [10]) is to consider the probability that two packets sent at a time difference of Δt will be received out of order. However, this definition of inversions does not take into account the fact that the sender rate varies. Our definition is also well-suited for a *receiver-oriented* view of network traffic, since it is the relative order of the received packets (and not their temporal lag) that will determine the information in the next ACK packet. Although pruning the dictionary inevitably makes the compression *lossy*, we regenerate synthetic traffic whose inversion structure shows a very high degree of fidelity. This is clear from Fig. 4, in which we have plotted the inversion spectra for NS-2-LONG and two RESTORED reconstructions.

As a second quantitative measure of disorder, we chose to compute shuffled up-sequences (SUS) [22]. This measure is defined as *the minimum number of ascending subsequences into which we can partition each listed sequence of packets.* If we compute the SUS metric for *all* sequences of packet IDs corresponding to the unordered state we find that over 95% of them are of type SUS $= 2$. In fact even packet sequences with a considerable number of inversions are relatively ordered with respect to

Table 2. Compression Ratios

	TRACE5	TRACE7	TRACE8	TRACE9	TRACE10	NS-2-LONG
Trace size	253MB	247MB	257MB	261MB	119MB	15.4GB
Pattern compression w/o pruning	528×	254×	813×	318×	1 164×	3 366×
Overall compression w/ pruning	16 443×	15 200×	16 542×	16 279×	11 709×	1 079 862×

Table 3. Original vs. Reconstructed Throughput

Throughput ratio $Q(C)$	TRACE5 %	TRACE7 %	TRACE8 %	TRACE9 %	TRACE10 %
0.95–1.05	65.8	68.1	64.2	64.0	63.7
0.90–1.10	85.8	85.5	81.6	83.4	80.3
0.85–1.15	94.4	93.1	91.7	92.3	91.9
0.80–1.20	97.9	97.1	96.2	96.5	96.5
0.75–1.25	98.8	98.1	97.9	97.8	97.9

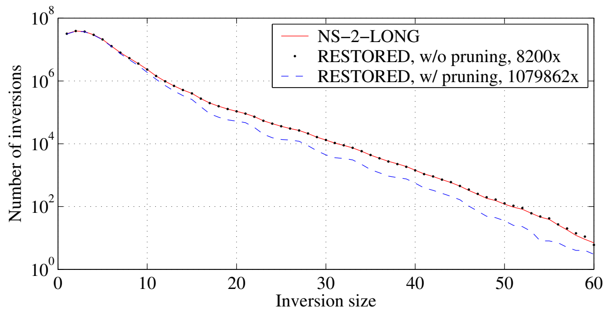

Fig. 4. Inversion spectrum of NS-2-LONG vs. RESTORED.

the SUS measure. This motivates our choice of SUS, as a metric reasonably orthogonal to the distribution of inversions captured by the inversion spectrum. We compare the distribution of SUS values of sequences produced by RESTORED (as a with those of connections in one of the real-life traces. The results are presented in Table 4, and the conclusion is that traces produced by RESTORED are very similar to the original ones (with respect to the SUS measure).

Finally two reordering metrics, introduced by Jayasumana *et al.* are *Reorder Density* and *Reorder Buffer-Occupancy Density (RBD)* [23, 24]. These measures are based on a notion of packet *displacement*. In the interest of space we point the reader to [23, 24] for precise definitions. We have computed the RD metric, aggregated over all connections in one trace, for both one of the original traces, and the corresponding sequences produced by RESTORED. Table 4 presents a comparison between these two distributions, tabulated for the most frequent values of displacement. As we can see the agreement is very good. Similar results hold at the individual connection level, as well as for the other real-life traces. Also, Table 4 presents the distribution of RBD, aggregated over all connections in Trace5, as well as the one from regenerated traces. As we can see, we are able to recover RBD distribution very well.

Acknowledgments: This work has been supported by the U.S. Department of Energy under contract W-705-ENG-36. Special thanks to Hot Rocks Café in Los Alamos.

References

1. Cisco Systems netflow, Available at http://www.cisco.com/warp/public/732/Tech/netflow.

Table 4. Recovering Reordering Metrics

	SUS				RD					RBD				
	2	3	4	5	-2	-1	0	1	2	0	1	2	3	4
TRACE5	95.07	4.54	0.33	0.04	0.20	0.78	98.20	0.23	0.12	98.59	0.46	0.29	0.20	0.13
RESTORED	95.10	4.51	0.34	0.04	0.20	0.78	98.23	0.23	0.10	98.62	0.45	0.27	0.19	0.13

2. W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, Addison-Wesley, 1999.
3. C. Cranor et al. “Gigascope: A stream database for network applications,” in *Proc.SIGMOD* 2003, 647–651.
4. W. Leland et al., “On the self-similar nature of Ethernet traffic (extended version),” *ACM/IEEE Transactions on Networking*, vol. 2 (1), pp. 1–15, 1994.
5. R. H. Riedi et al., “A multifractal wavelet model with application to network traffic,” *IEEE Transactions on Information Theory*, vol. 45, no. 3, pp. 992–1018, April 1999.
6. D. Figueiredo et al. “On TCP and self-similar traffic,” *J. Perf. Evaluation (to appear)*, 2005.
7. J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, “Internet traffic tends to Poisson and independent as the load increases,” Bell Labs, Murray Hill, NJ, Tech. Rep., 2001.
8. D. Figueiredo et al. “On the autocorrelation structure of TCP traffic,” *Computer Networks*, vol. 40, no. 3, pp. 339–361, October 2002.
9. J. Bennett et al., “Packet reordering is not pathological network behavior,” *IEEE/ACM Transactions on Networking*, vol. 7 (6), pp. 789–798, 1999.
10. J. Bellardo and S. Savage, “Measuring packet reordering,” in *Proc. ACM SIGCOMM Internet Measurement Workshop*, Nov. 2002, pp. 97–105.
11. W. Fang and L. Peterson, “Internet-AS traffic patterns and their implications,” in *Proc. IEEE GLOBECOM Conf.*, 1999, pp. 1859–1868.
12. A. Feldmann et al. “Deriving traffic demands for operational IP networks: Methodology and experience,” in *Proc. ACM SIGCOMM* , 2000, pp. 257–270.
13. C. Estan and G. Varghese, “New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice,” *ACM TOCS*, vol. 21 (3), pp. 270–313, 2003.
14. K. Park and W. Willinger, Eds., *Self-similar network traffic and performance evaluation.* Wiley, 2000.
15. J. Cao et al. “On the nonstationarity of Internet traffic,” in *Proc. ACM SIGMETRICS*, 2001.
16. P. Brockwell and R. Davis, *Introduction to Time Series and Forecasting*, Springer, 2002.
17. G. Box and G. Jenkins, *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976, p. 33.
18. R. Hogg and A. Craig, *Introduction to Mathematical Statistics*, Macmillan, 1995.
19. L. Peterson and B. Davie, *Computer Networks. A Systems Approach*, M. Kauffman, 2000.
20. A. Hansson, G. Istrate, and S. Kasiviswanathan, “Combinatorics of TCP reordering,” submitted to a special issue of *Journal of Combinatorial Optimization*, July 2005.
21. S. Jaiswal et al. “Inferring TCP connection characteristics through passive measurements,” in *Proc. IEEE INFOCOM.*, 2004.
22. V. Estivill-Castro and D. Wood, “A survey of adaptive sorting algorithms,” *ACM Computing Surveys*, vol. 24, no. 4, pp. 441–476, December 1992.
23. A. Jayasumana et al. “Reorder density and reorder buffer-occupancy density—metrics for packet reordering measurements”, IETF IP Performance Metrics WG, Available at http://www.ietf.org/internet-drafts/draft-jayasumana-reorder-density-04.txt.
24. A. Jayasumana et al., “RD: A formal, comprehensive metric for packet reordering,” in *Proc. IFIP Networking*, 2005.
25. A. Veres and M. Boda, “The chaotic nature of TCP congestion control,” in *Proc. IEEE INFOCOM* 2000, pp 1715–1723.