What if …?
Scaling Network Traffic for Realistic Experimentation

By

Shaddi Hasan

Honors Thesis

Computer Science Department

University of North Carolina at Chapel Hill

May 8, 2010

Approved:

_____

Abstract
Networking researchers perform controlled experiments to test new protocols and techniques they develop. A critical variable for the network traffic commonly used as input for such experiments, known as a trace, is its offered load – how much data is transmitted per unit time. Modifying a trace's offered load is a desirable ability for performing controlled experimentation, but performing such modification while preserving important unrelated characteristics of the trace is difficult. I examine the effects of one technique for achieving this goal, block resampling, and propose modifications for adjusting the offered load of a trace while preserving its other fundamental characteristics. I show that the existing block resampling technique, despite achieving its original design goals, can produce unpredictable result traces and introduces biases into result traces that effect important protocol performance measures such as queue lengths and number of active connections. I then provide modifications to the basic block resampling algorithm that more faithfully preserve these characteristics in the way they were expressed in the original input trace.

I. Introduction
Emulation of network traffic in controlled experiments underpins much important research in the networking literature and is essential for repeatable, scientific study of network protocol performance. Despite this, the network community lacks a comprehensive understanding of how different emulation methodologies impact important protocol performance metrics. As Douglas Comer, a networking researcher at Purdue and VP of Research at Cisco, recently noted, "building a large packet-switching network is easy; understanding the behavior of traffic in a large packet-switching network is nearly impossible". A result of this complexity is that poorly grounded assumptions made during experimental design can and do have drastic effects on experimental results.

The goal of the networking research, of course, is to understand network protocol performance so as to improve the performance of networks used outside the laboratory environment, such as the Internet. Thus, to ensure that experimental results are relevant for widely-used systems, the

conditions of an experiment should reflect as closely as possible conditions seen on these real-world networks.

This work represents an effort to better understand one such aspect of experimental design, the method by which one introduces variability into an experiment by modification of offered load. The ability to modify the offered load of a network trace is a highly desirable weapon in the networking researcher's arsenal. This would allow researchers to examine how the technique they are evaluating responds when the volume of traffic on a network increases or decreases. Currently, the question of "what happens if volume of traffic increases by 30%?" can only be answered approximately with simulation techniques: such a tool would permit that question, and others like it, to be answered with confidence that the modified trace still exhibited the realistic source-level behavior characteristic of the original trace. Unfortunately, the ability to scale Internet traffic in a controlled manner implies an understanding of the underlying structure of that traffic, the complexities of which is eloquently outlined in [FP01] and encompasses several open research questions.

I demonstrate that one technique for scaling traffic, referred to here as randomized block-resampling and introduced in [HC06], introduces several important differences into the resultant scaled traces it creates, dramatically affecting second order performance metrics beyond throughput alone. While there are circumstances where such variability is desirable, but there is also a substantial class of experiments for which such unpredictable variation is unacceptable. Because this variation's impact on protocol performance is poorly understood, I have sought to modify the basic block-resampling technique to produces sets of connection vectors that differ in offered load from the original trace from which they derive, yet in other respects demonstrate similar performance characteristics. For the sake of this work, I focus on both the throughput, as well as three important performance metrics: active connections, queue lengths, and connection durations. Here, throughput is the first-order metric is directly affected by scaling offered load, with active connections, queue lengths, and connection durations being second order effects.

The remainder of this paper is organized as follows. I will first provide a background on the theoretical model used for traffic generation and a description of the block resampling technique. Next, I compare a number of scaled traces produced using block resampling to the original trace from which they derive. I use this to motivate some modifications to the randomized block resampling technique in order to more faithfully preserve second order performance characteristics, and provide an evaluation of those modifications. Finally, I will discuss future directions to this work, with an emphasis on extensions to the block resampling technique.

II. Background

This work is the continuation of a body of work built around the "a-b-t model" of connection structure. The a-b-t model defines a simple yet powerful abstraction for describing a network trace at the source level, in keeping with [FP01]. While a full description of the a-b-t model is beyond the scope of this work, a basic introduction is provided below. For a more in-depth treatment of the subject, please see [WAHC+06].

In the a-b-t model, a network trace is defined to consist of multiple connection vectors. Each connection vector represents a single TCP connection as a sequence of data exchanges between two hosts, A and B. Data is exchanged in a series of request-response-thinktime sequences known as epochs. "Thinktime" is a period following a data exchange in which not data is transmitted. A single connection vector is made up of multiple epochs.

Using the a-t-b-t model of connection structure and the vocabulary it provides, we can state the "Offered Load Scaling" problem as follows:

Offered Load Scaling Problem: *Scale the offered load of a network trace while maintaining the source-level characteristics of the original trace.*

> **Input:** An anonymized packet header trace with an offered load of **L** Mbps, represented by a set of connection vectors **V=v1, v2, …, vn**.
>
> **Output:** A set of connection vectors **V' = v1', v2', …, vn'** representing a packet header trace with an offered load of **L'**.

One algorithm to solve this problem was presented in [HC06]. This original block-resampling algorithm, referred to here as "Randomized Block Resampling" and described in further detail below, was intended to not only scale the offered load of a set of connection vectors, but also to introduce variability into the connection vectors that it produced.

The block-resampling technique was proposed and discussed at some length in [HC06]. The discussion there, however, focuses on the ability of block resampling to introduce variability into an experiment while preserving the so-called long-range dependence of a network trace. The method described in that paper, referred to here as *Randomized Block Resampling*, is never fully specified, so I briefly describe its operation and attempt more concretely specify it below. Randomized Block Resampling breaks a set of connection vectors into blocks of a time duration known as the window size, **W**. [HC06] describes at length the relationship between block duration and the degree to which long-range dependency is preserved in a resultant set of connection vectors, and recommended window sizes of between 30 seconds and 5 minutes.

After creating these blocks, we define a number of bins equal to the desired duration of our output trace divided by the block duration. We then populate each of these bins with a block selected randomly with replacement. If our desired load is more than twice the load of the original trace, we may add more than one block per bin in the same manner. Finally, once all the bins have been populated, individual connection vectors are added or removed at random until the total number of bytes transferred in the forward direction of the trace divided by the desired duration is equivalent to our desired load. Thus, the constraint block resampling aims to satisfy is:

$$\text{Load} = \text{Total Bytes} / \text{Duration}$$

Before doing performing block resampling, we a set of connection vectors sorted by the relative start time within the trace of each connection, we create a summary file containing the type, start time,

connection vector ID, and total bytes transferred in each direction for each connection. To more precisely describe this process, I present the *RandomizedBlockResample* algorithm below.

Given a summarized set of connection vectors **V**, a start and end time **S** and **E** relative to the start time of the trace, a block duration **W** in seconds, a desired duration **D,** and a target offered load **L'**:

RandomizedBlockResample(**V, S, E, W, D, L'**):

1. **B** <- An array of length **D/W**, each element an array of connection vectors
2. for every connection vector **v** in **V:**
3. *ignore if* **v**.start_time falls outside **S,E**.
4. **block_index** <- int(**v**.start_time / **W)**
5. **B**[**block_index**].add(**v**.id, **v**.start_time, **v**.bytes_forward)
6. **resampled_blocks** <- An array of length **D/W**
7. **total_fwd_bytes** <- 0
8. for every block **b** in **resampled_blocks**:
9. for i in range(1, int(**L'**,**L**)):
10. **block_index** <- *uniformly randomly select a block index from* **B**
11. **b** <- **B[block_index]**
12. for each connection vector **v** in **b**:
13. **total_fwd_bytes** <- **total_fwd_bytes** + **v**.bytes_forward
14. if **total_fwd_bytes** > **L'** :
15. while **total_fwd_bytes** > **L'**:
16. **cvec** <- *uniformly randomly select a connection vector from* **resampled_blocks**
17. **cvec_block_index** <- *index of block containing* cvec *from* **resampled_blocks**
18. *remove* **resampled_blocks**[**cvec_block_index**]
19. **total_fwd_bytes** <- **total_fwd_bytes** - **cvec.** bytes_forward
20. elseif **total_fwd_bytes** < **L'** :
21. **resampled_blocks2** <- *perform second resampling as in lines n-m*
22. while **total_fwd_bytes** < **L'**:
23. **cvec** <- *uniformly randomly select a connection vector from* resampled_blocks2
24. **cvec_block_index** <- *index of block containing* cvec *from* resampled_blocks2
25. **resampled_blocks**[**cvec_block_index**] <- *add* cvec
26. **total_fwd_bytes** <- **total_fwd_bytes** + **cvec.** bytes_forward
27. return **resampled_blocks**

While resampled_blocks does not contain an actual set of connection vectors with load **L'**, creation of a corresponding set of connection vectors can be achieved trivially and mechanically. Because the resampled_blocks array is essentially a re-ordered listing of connection vector IDs, a new set of connection vectors can be created simply by iterating through the array, selecting the corresponding connection vector from the original set of connection vectors, and saving each of those connection vectors to a file.

III. Methodology

Experimental Setup

To understand the behavior of the block-resampling algorithm, I performed a series of experiments to evaluate how well a replay of traces produced by block resampling reflected the characteristics of a replay of the original network trace. Replays were performed using the Realistic Traffic Generator (Tmix) on UNC's Netlab experimental testbed, which consists of 30 pairs of end systems connected by a pair of software routers, simulating a gateway link at a peering point between two ISPs.

This system was rigorously calibrated to ensure that it did not present any bottlenecks that could affect experimental results. Each of the 60 end systems are connected to switches with 1Gbps links, and each switch is in turn connected to a router with a 10Gbps link. There is a 1Gbps bottleneck link between the two routers. We demonstrated that this system is capable of performing replays at the load we use in our experiments without congestion at the routers, and that all of the traffic generators are able to produce the load required for these experiments.

In order to emulate congestion, the link between the routers was artificially capped such that the offered load of the emulated trace was 95% of link capacity. This is referred to as "constrained mode". Thus, the majority of my experiments were capped at 424Mbps (for an offered load of 400Mbps), with a couple others capped at 368Mbps (for an offered load of 350Mbps). Experiments run without either of these caps in place were only constrained by the 1Gbps bottleneck link; because essentially no congestion was seen in this configuration, I refer to this as "unconstrained mode".

Original Packet-Header Trace

The original set of connection vectors I used in this experiment were derived from an anonymized packet header trace taken from a border link on the network of a large Fortune 500 company. This trace was captured on October 10, 2006, at 2:20PM. While the trace was originally 1 hour and 14 minutes long, only the first hour was considered in these experiments. All bytes seen after the one-

hour mark were simply ignored, just as if the original capture had ended at that point. Connections in the trace that were initiated or terminated outside the duration of the trace were artificially completed so Tmix could replay them.

The one-hour trace contained 2,785,054 connections. Of these, 2,733,996 connections (98.2%) were sequential connections, while 51,058 (1.8%) were concurrent. The mean offered load in direction 1 was 404.4Mbps, with a standard deviation of 39.3Mbps. The mean offered load in direction 2 was 366.4Mbps, with a standard deviation of 28.7Mbps. The mean round-trip time (RTT) was 92.3ms, with a standard deviation of 143.8ms. The mean number of epochs per connection was 8.8, with a standard deviation of 123.4, reflective of a number of very long connections present in this trace. The average connection size was 129.3KB, with a standard deviation of 5445.7KB.

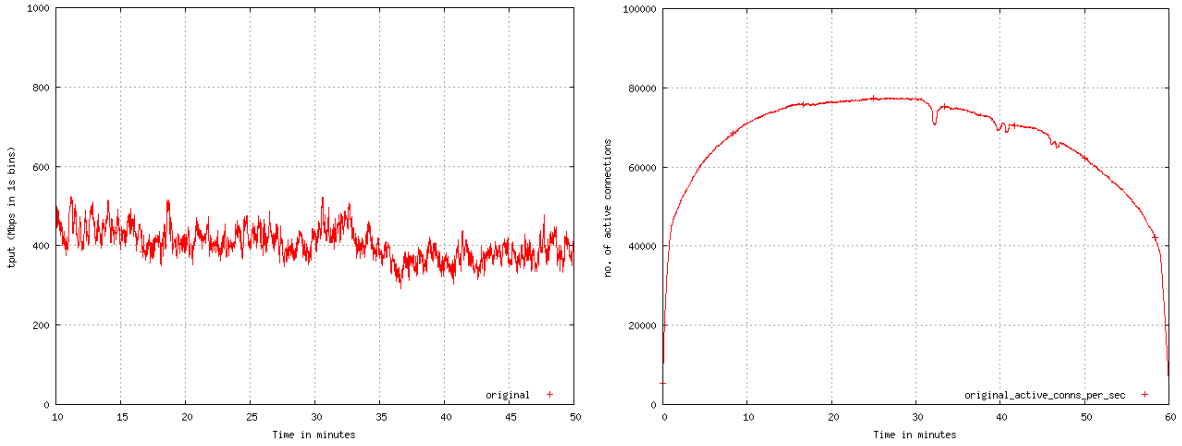


Figure 1. Throughput and active connections from a replay of original set of connection vectors produced from the packet header trace. Average throughput was 404Mbps. Note the non-stationarity in throughput over the length of the trace: the first half of the trace is at a consistently higher throughput than the last 25 minutes.

IV. Unpredictability in Block Resampling

Variability in Block Resampling Experiments

The block resampling process is not deterministic. Indeed, uniform random selection of both blocks of connections as well as individual connection vectors plays a key role in its operation. While this allows RandomizedBlockResample to introduce variability into a derived set of connection vectors,

it is not known how significantly that variability affects key performance measures. Such knowledge is necessary for a large class of experiments, including those that seek to compare a block resampled set of connection vectors with a non-block resampled one.

Thus, my first set of experiments was geared towards understanding what type of variability exists within connection vectors produced by the block resampling process. In order to achieve this goal, I produced sets of connection vectors corresponding to different permutations of input parameters for block resampling, described below. A Tmix replay of the original set of connection vectors derived from the Corporate packet header trace was used to represent ground truth for this set of experiments, and performance metrics measured from that replay served as a point of comparison for the results of my other experiments.

Despite its non-deterministic nature, we would expect block resampled traces to at least generally resemble the real-world trace from which they derive. However, this does not appear to be the case. I first constructed three sets of connection vectors with a target load of 400Mbps, a desired duration of 1hr, and a window size of 30 seconds (as recommended by [HC06]). For each, I accepted the full hour's worth of connections from the original connection vector set. I chose a target load of 400Mbps in order to enable direct comparison between my derived connection vector sets and my original set of connection vectors, which displayed an offered load of 400Mbps. While in theory no scaling of the trace should be necessary, RandomizedBlockResample essentially builds a resultant set of connection vectors from scratch, creating entirely new sets of connection vectors.

Even though these three sets of connection vectors had been produced with the exact same input parameters, Tmix replays of each produced widely divergent results. Fig. x illustrates the time series of active connections and throughput for each replay.

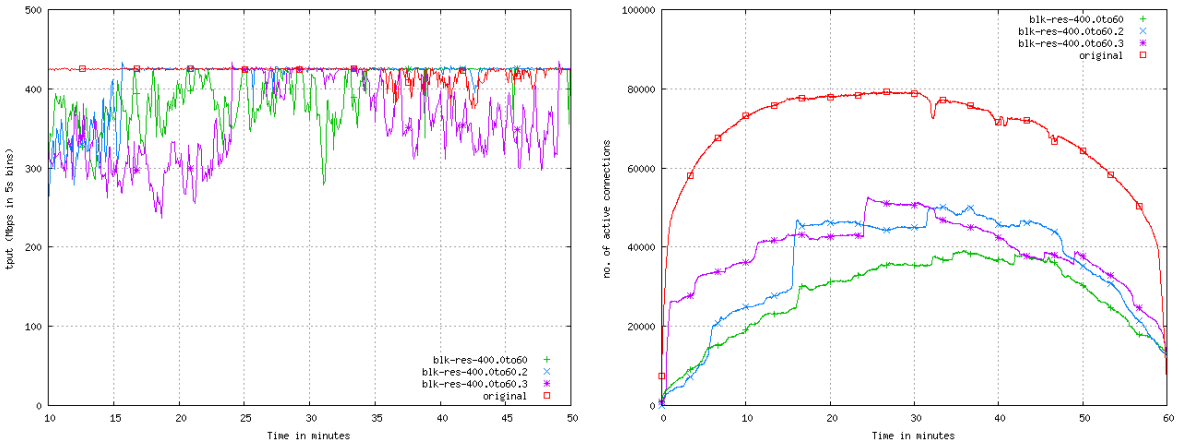


Figure 2. Throughput time series and active connection time series from a replay of the original connection vector set (red) and three randomized block resampled connection vector sets.

Block resampled traces showed two important differences from the original replay. First, the throughput for the original trace remained relatively constant throughout the duration of the replay, hovering near the link limit, whereas in each of the block resampled replays throughput varied significantly over the replay, often significantly dipping below the link limit. The second significant difference from the original replay was that each of the block resampled replays maintained between 30,000 – 40,000 fewer active connections at any given time. Moreover, connection starts occurred in a distinctly different pattern in the resampled replays than the original replay. More specifically, the original replay demonstrated a smooth growth and decline in number of active connections over the course of the replay, peaking close to 80,000 active connections after reaching a relatively stable plateau after minute 10. The large increase at the beginning of the replay can be attributed to start up effects that are artifacts of artificially completing connections that started before the original trace was captured. The drop-off at the end is due to a similar effect caused by artificially completing connections that finished after the trace began.

This variation in throughput and active connection patterns led to a different pattern of congestion and hence a different pattern of queue activity. The original replay exhibited some degree of congestion (i.e., had a non-zero queue length) during 92.2% of the replay. While one of the three

block resampled replays came close to this level of congestion (non-zero queue length 87% of the replay), the median queue length of each of these replays differed substantially, with 2614 packets for the original versus 5106 for the block resampled trace. At the same time, the two other block resampled replays exhibited far less congestion, with median queue lengths of 136 packets and 0 packets (this latter replay was in an uncongested state 56% of the replay).

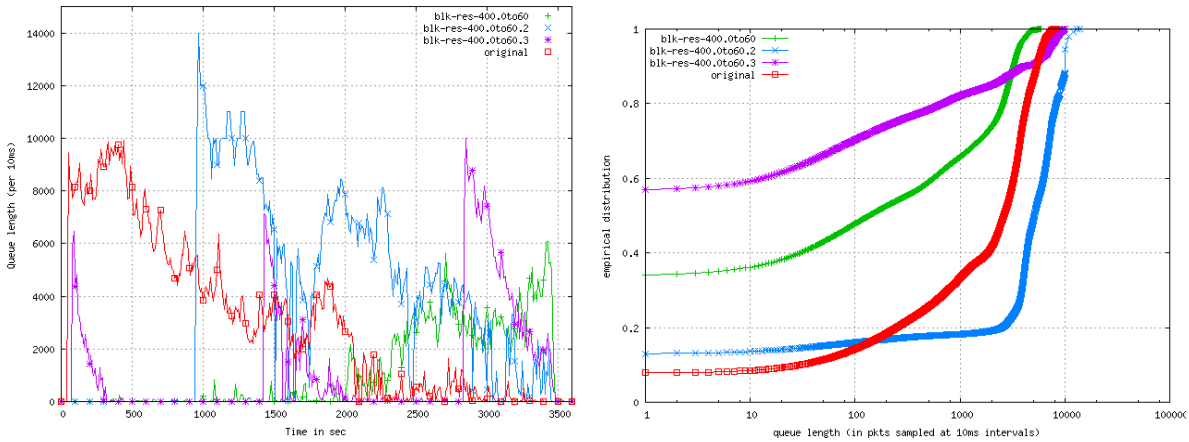


Figure 3. Time series of queue lengths and cumulative distribution of queue length sizes for a replay of original (red) and block resampled connection vector sets.

Beyond a difference in the distribution of router queue lengths, the time series of queue lengths during the replay are substantially different. The original trace demonstrated gradually decreasing queue lengths across almost the entire duration of the trace, finally reaching a non-congested state in the last ten minutes of the replay. None of the other replays, one again, came close to matching this behavior. The least-congested trace demonstrated brief spikes in queue lengths followed by long periods of no congestion. One trace demonstrated a spike in queue length followed by a gradual decline, which at first appears to be generally in line with the original trace. However, this large spike in queue length occurred very shortly after the number of active connections in that trace's replay jumped from approximately 29000 to over 46000 in the span of 45 seconds. This 58% increase in active connections, and the resulting spike in queue lengths, are do not seem to be reflective of the

original trace, and such an event would be unexpected to occur in the real world. That is to say, this spike in queue length is merely an artifact of the block resampling process.

Despite this variation in queue length dynamics, several other attributes of the block-resampled traces matched that of the original. Given that two traces transmit a roughly equivalent number of bytes over a one hour period, yet the block resampled one has 20,000-40,000 fewer active connections at any given time, it seems logical that the block-resampled trace would be comprised of connection vectors that are either shorter than the original or transmit more data per connection. However, this does not appear to be the case, and in fact all block resampled traces seemed to closely resemble the original trace in these respects.

I first considered connection durations. A connection vector's duration is defined as the total amount of time a connection is active, from being opened to being closed. All of the block resampled traces closely resembled the original trace's distribution of connection durations, with median connection durations of 643ms, 829ms, and 653ms for each of the block resampled traces compared with 789ms in the original trace. This minor variation alone cannot account for the dramatic differences in active connection patterns or congestion dynamics.

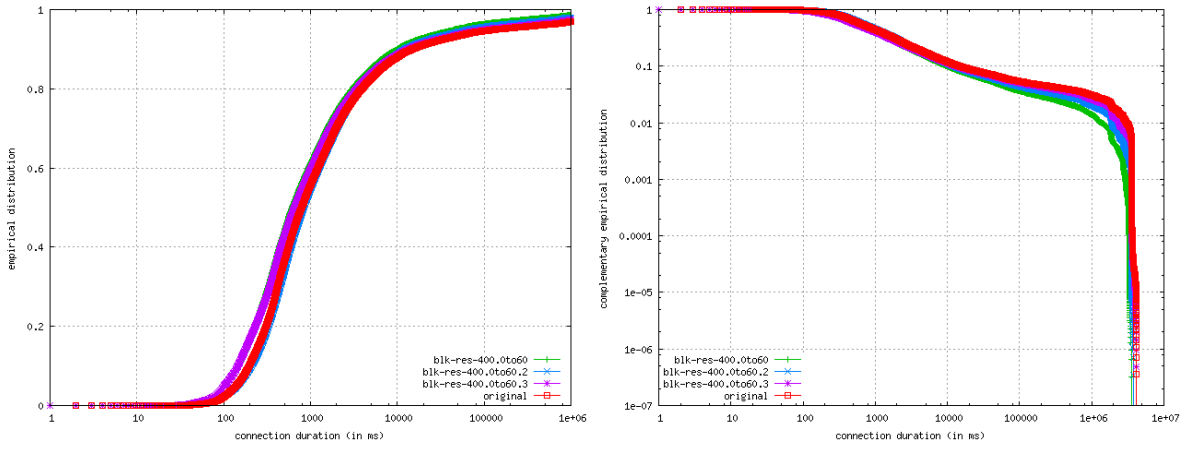


Figure 4. Cumulative distribution and complementary cumulative distribution of connection durations from replays of the original connection vector set (red) and three randomized block resampled connection vector sets.

Connection vector sizes also did not vary significantly between the block resampled traces and the original trace. Connection vector size is defined here as the total number of bytes transferred across the entire duration of the connection vector; more precisely, this is the sum of all a's and b's over every epoch in a connection.

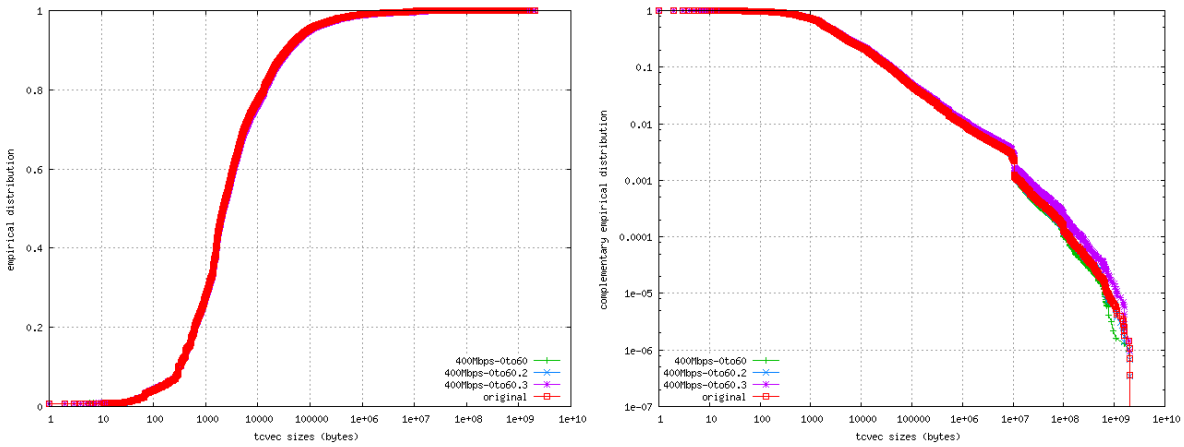


Figure 5. Cumulative distribution and complementary cumulative distribution of connection vector sizes in original connection vector set (red) and three randomized block resampled connection vector sets.

The distribution of connection vector sizes matches that of the original trace almost exactly for each of the three resampled traces. While this does not explain the variation exhibited among the three block resampled traces, it does intuitively make sense in the context of the *RandomizedBlockResample* algorithm. Assuming essentially uniform selection of connection vectors during the block resampling process, the resulting distribution of connection vectors should be essentially the same as the input connection vectors, which these charts demonstrate to be the case.

Finally, I considered whether connection start times were distributed differently in the resultant block resampled traces than the original trace, or if they were not evenly distributed throughout. The start time of a connection is defined as the relative time in the set of connection vectors the connection vector is to begin transmitting data. Again, the distribution of start times is essentially identical among the block resampled and original traces.

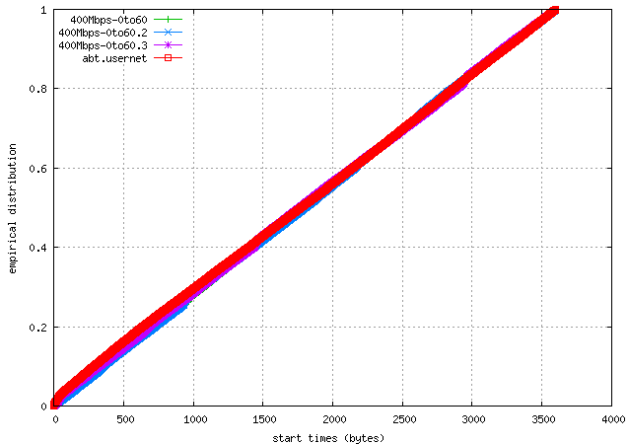


Figure 6. Cumulative distribution of connection start times in original connection vector set and three randomized block resampled connection vector sets.

Double block resampling

Another set of experiments I performed evaluated how the performance characteristics of a given trace changed after repeated block resampling. If one assumes that block resampling produces realistic traces that remain true to the original trace properties, performing block resampling a trace that was itself created by block resampling should produce on that continues to reflect the original trace characteristics. This would be true if block resampling were a transitive process.

To perform this evaluation, I block-resampled the original input trace from its original load of 404Mbps down to 300Mbps, and then block resampled that resultant 300Mbps again to produce a final resultant trace with an offered load of 400Mbps. For each trace, I used the same input parameters as my previous set of experiments, namely an input duration of the full one-hour original trace and a block duration of 30 seconds.

Repeated block resampling does not maintain original trace characteristics and in fact produces resultant traces bearing little resemblance to the original. This replay had a much higher number of active connections than any other block-resampled connection vector set, at times exceeding even the original trace to peak at almost 90,000 active connections.

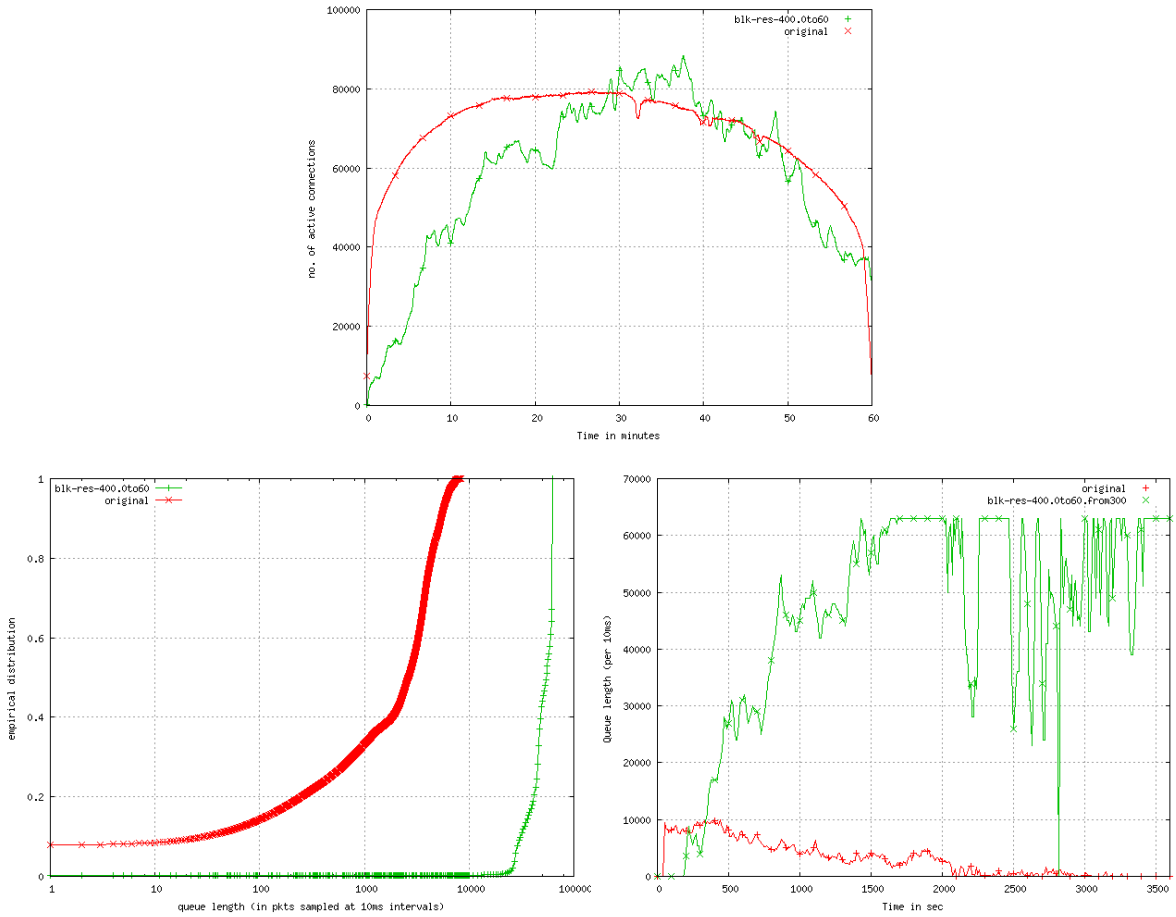


Figure 7. Comparison of active connections and queue length behavior for replay of original connection vectors (red) a connection vector set (green) that had been scaled down to 300Mbps, then back up to 400Mbps, using *RandomizedBlockResample.* The high level of congestion produced an erratic and non-representative throughput time series, so it is not presented here. Suffice it to say that throughput was at the link limit of 424Mbps for the entirety of the replay.

Congestion was also much more severe in the double-resampled replay, with a median queue length of 55,000 packets. Congestion was so great that this replay experienced loss due to full queues on the routers. In fact, queue behavior was highly erratic, particularly in the second half of the replay.

Most striking, and unexpected, was that the distribution of connection durations was significantly heavier for the double-resampled replay, with a median of 4048ms compared to the original's median of 789ms. This was likely due to packet loss experienced during the time that the queue had filled.

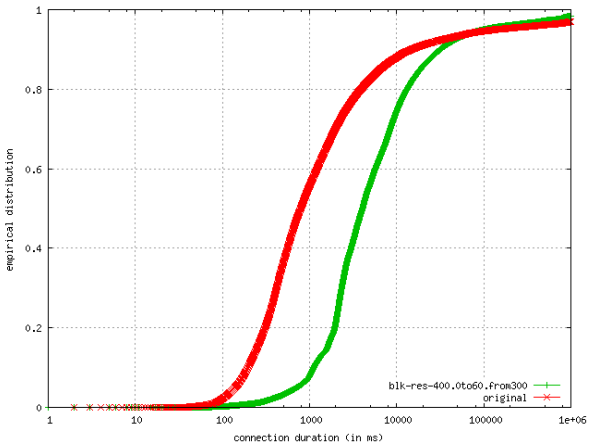


Figure 8. Comparison of connection durations for replay of original connection vectors (red) a connection vector set (green) that had been scaled down to 300Mbps, then back up to 400Mbps, using *RandomizedBlockResample*. The median connection duration of the latter was over 5 times that of the original.

Clearly, then, performing multiple rounds of block resampling does not produce connection vector sets that bear resemblance to the original trace from which they derived.

V. *Self-block resampling*: A more faithful block resampling technique

These variations demonstrated that the block resampling technique, as outlined in [HC06], yields highly unpredictable results, even when given the same input parameters and the same input network trace. In particular, I observed that congestion and queue length behaviors were not preserved between the original input trace and the derived block resampled traces. While it could be argued that comparing the congestion dynamics of the resultant traces to the original trace is to use something of an arbitrary baseline, this in fact highlights the danger of simply accepting these block resampled traces as "realistic enough". We turn to traces directly derived from real-world Internet traffic in order to capture traffic characteristics that are difficult or impossible to model; congestion dynamics fall into this class of complicated behaviors. The fact that queue length behavior in each of the three block resampled traces differs so dramatically from the original, and from each other, motivates the need for an approach for modifying offered load while remaining more faithful to the original trace's dynamics.

One of the design goals of *RandomizedBlockResample* was the ability to introduce variability into a set of experiments. It accomplishes this by synthetically creating a set of new connection vectors by randomizing blocks of connections from an original input trace. While this accomplishes the goal of introducing variability into a set of connection vectors, as I have shown above the results of this transformation are unpredictable and may not be desirable for certain types of experimentation.

If introducing variability is a non-goal, this randomization step becomes unnecessary, and removing it offers the promise of creating resultant traces very similar to the original input trace yet with scaled offered loads. This simple modification to the *RandomizedBlockResample* technique, which I refer to as *SelfBlockResampling*, is in all respects identical to *RandomizedBlockResample* save for the allocation of blocks. Rather than randomly allocating blocks of connections from the original trace to blocks in the resultant trace, *SelfBlockResample* maintains the ordering of connection vector blocks from the original trace to the resultant trace. Thus, the first block of the input trace will also be the first block of the resultant trace; the second block of the original trace will also be the second block of the resultant trace; and so forth. After all sufficient rounds of block allocation have occurred, the trace is then scaled up or down to its target offered load by adding or removing connection vectors that have been selected uniformly at random from the population of all connection vectors, just as in *RandomizedBlockResample*. Because of their significant similarities, I do not formally present *SelfBlockResample* here; the listing for *RBR* above provides a sufficient description for this algorithm as well. The only difference is the selection of blocks in line 10.

Evaluating Self-Block Resampling

The ideal test to evaluate preservation of important performance characteristics by a offered load scaling technique would be to compare a block-resampled version of a network trace taken at T1 with a network trace taken at T2, a point at which throughput on the real world network has organically reached the target offered load. Unfortunately, this was not possible with the available resources and in the available timeframe for this project. However, I approximated this situation as

follows. I first randomly removed 10% of the connections from my original set of connection vectors, thus creating a "base" set of connection vectors with some reduced offered load. I created three "base" connection vector sets in this manner. Next, I created two sets of block resampled connection vectors for each base set using each block resampling technique. I scaled the offered load in each case to 400Mbps, the offered load of the original connection vector set.

This procedure created six new sets of connection vectors—two block-resampled connection vector sets for each of the three "base" connection vector sets. I considered the original set of connection vectors to be my baseline, analogous to the T2, organic-growth case outlined above. Each of the bases cases was analogous to the T1 trace. I performed the same evaluations I had previously considered for each of these pairs: throughput, active connections, queue length behavior, and connection durations.

SelfBlockResample produced resultant traces that very closely matched the original input trace in regard to both throughput over time and active connections. Connection vector sets produced using the self-resampling technique demonstrated a pattern of throughput very similar to the original trace, maintaining a congested link until minute 35 of the experiment, and then undergoing a modest decline in throughput after that. On the other hand, randomized block resampled connection vector sets were again highly variable in the results they produced. In fact, one set displayed an offered load of well under 424Mbps (the link capacity), thus causing no congestion over the course of that trace.

The pattern of active connections seen in the self-resampled replays also closely matched that of the original. As before, *RandomizedBlockResample* replays had far fewer active connections at any given time than the original connection vector set and exhibited large bursts of connection starts throughout the replay, including a jump of over 15,000 connections in a one minute span.

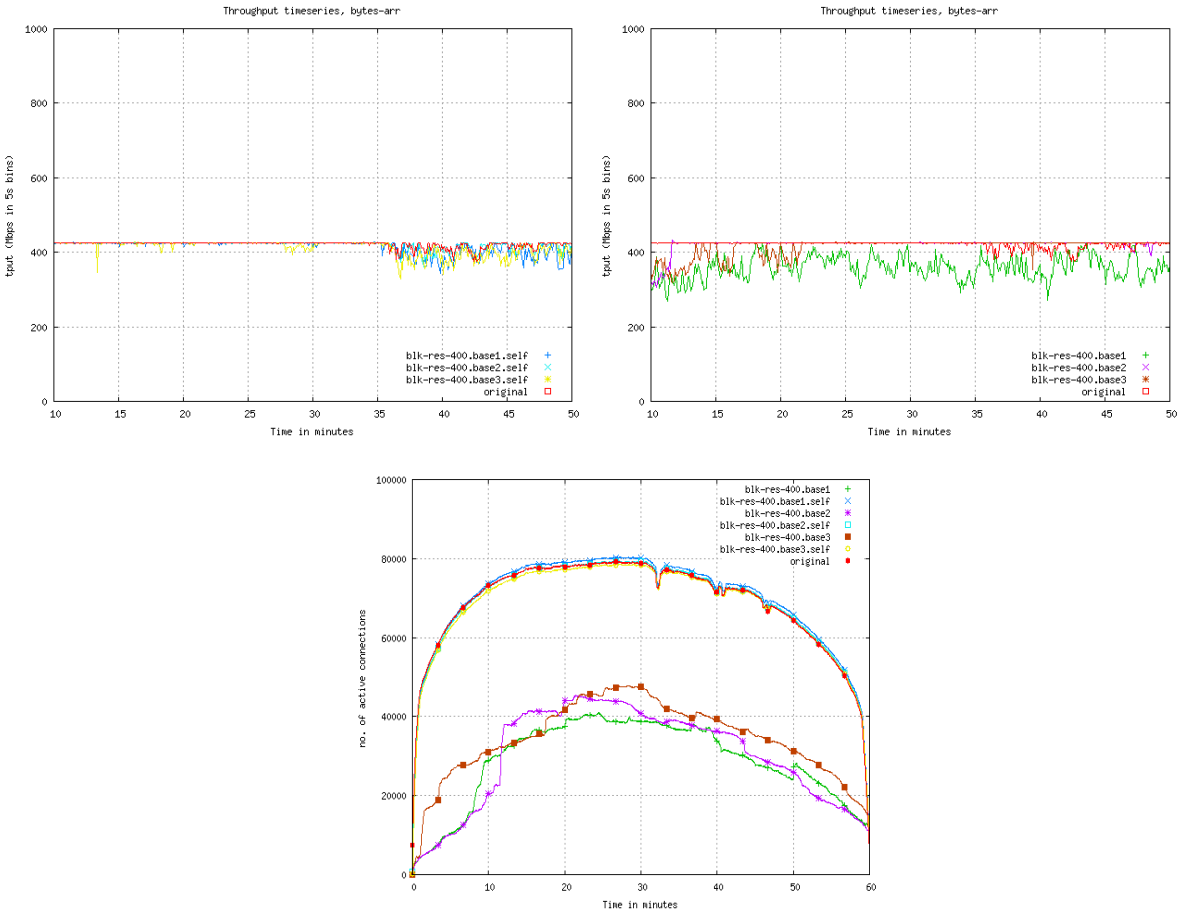
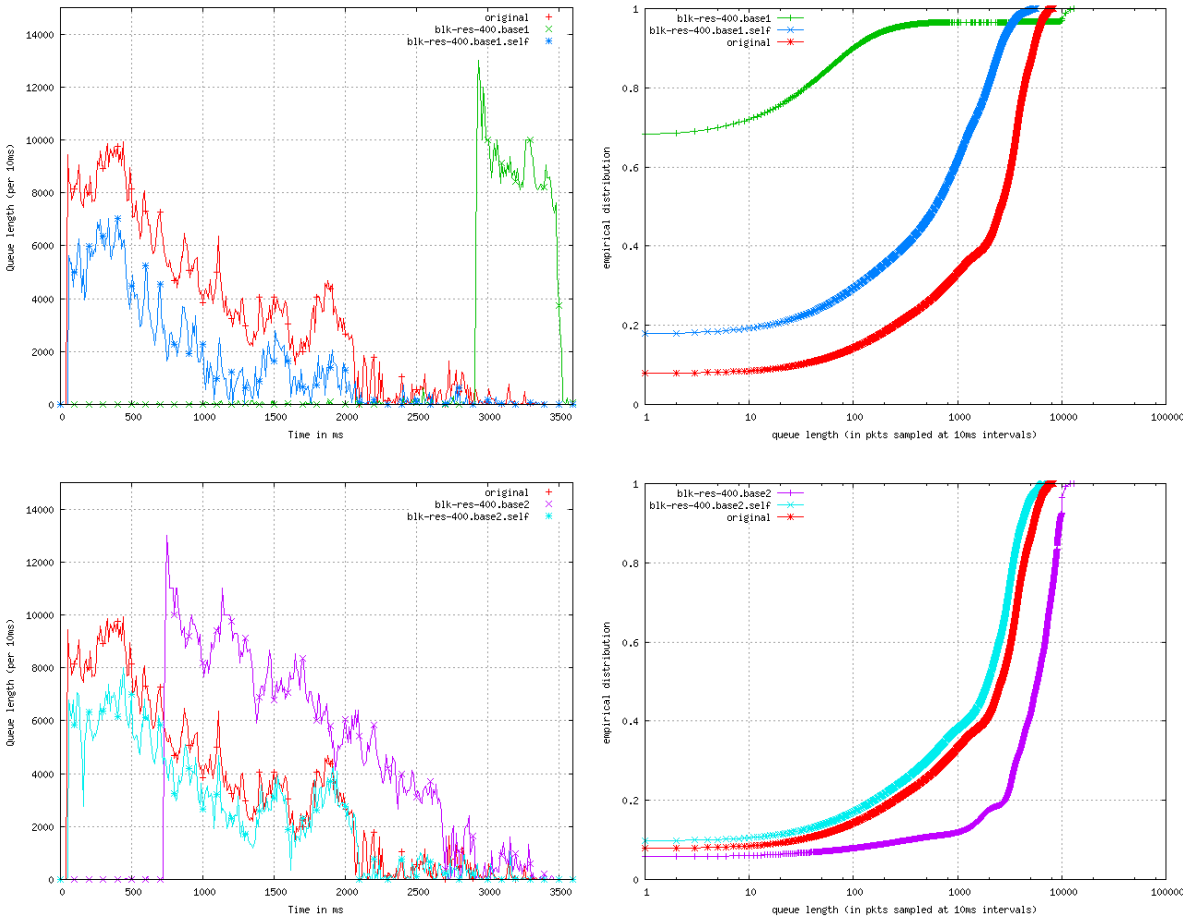


Figure 9. Throughput and active connection comparisons for the "base" set of experiments, comparing randomized block resampled traces with self block resampled ones. *Top left:* Throughput time series for original replay (red) and three self block resampled replays. *Top right:* Throughput time series for original replay (red) and three randomized block resampled replays.

As would be expected, this result had an impact on queue length behaviors as well. Each of the self-block resampled traces exhibited queue length behavior similar to that of the original, with a spike in queue lengths toward the beginning of the replay and a gradual decline over the duration, which is to be expected given the non-stationarity of the throughput in the original trace. On the other hand, randomized block resampled trace again exhibited substantially different queue length behavior from the original trace. In the second set of experiments ("base2"), the randomized block resampled replay once again exhibited a spike in connection starts about 12 minutes into the trace that correlated with a spike in queue lengths soon after.

Distribution of queue lengths also provides evidence that self-block resampling better preserves queue length behavior from the original trace. In the first two sets of experiments, the distribution of queue lengths for the self-resampled replays match the original much more closely. In the third set, the randomized-resampled replay matches the original more closely, but the shape of the self-resampled distribution is essentially that of the original shifted upwards. Indeed, in all cases, the shape of the self-resampled distribution closely matches that of the original, indicating that while the *scale* of congestion may not be precisely maintained, the *relative levels* of congestion are matched quite closely. Finally, the resultant traces from self-resampling demonstrate much more predictable queue length behaviors compared to the randomized-block resampled ones, which were observed to produces queue length distributions both substantially heavier and substantially lighter than that of the original.



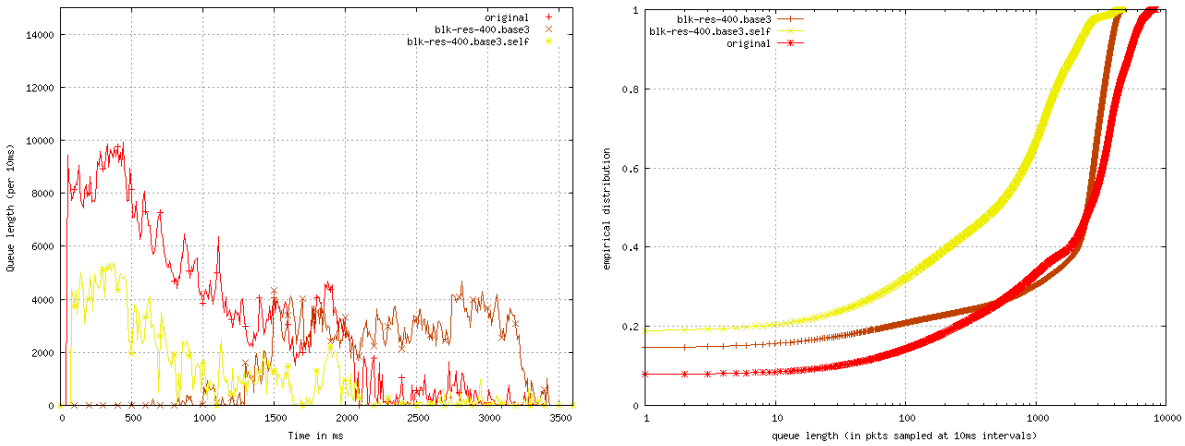


Figure 10. Queue length time series and cumulative distribution for each of the three "base" set of experiments. Each row corresponds to one "base" connection vector set produced by removing uniformly at random 10% of the connection vectors from the original trace. Both Self Block Resampling and Randomized Block Resampling were then used to scale these base connection vector sets back up to the original offered load of 400Mbps.

Connection durations did not exhibit any significant variation between resampling techniques. In all cases, the distribution of durations matched the original almost exactly, as expected.

Scaling Traces with Self-Block Resampling

Thus far I have focused on using block resampling techniques to "scale" network traces by a factor of 1, back to their original throughput. As stated before, this enabled the direct comparision of block resampled traces to the original trace, which was essential in order to be able to answer the fundamental question of how traces produced by block resampling mirror the original trace from which they derive. Based on these results, it appears that self-block resampling more faithfully preserves both first-order (throughput) and second order (active connections, queue lengths, connection durations) performance characteristics from the original trace.

Of course, the real purpose of any of these techniques is to scale the load of a trace to some new target than its current offered load. Because I have been taking a $\mathbf{L}' \approx \mathbf{L}$, self-block resampling has only needed to make minor modifications to the original trace, where the number of connection vectors added to or removed from the original connection vector set is much less than the total

number of connection vectors present. Thus, I compared the results of both *RandomizedBlockResample* and *SelfBlockResample* when scaling the original connection vector set down to 350Mbps from 400Mbps (due to limitations of our experimental testbed, I could not scale the load to a significantly higher level). As before, I performed block resampling using the standard parameters of accepting connection vectors from the full one-hour duration of the original trace and using a block duration of 30 seconds. I reduced the capacity of the bottleneck link between the routers from 424Mbps to 368Mbps to ensure that the new 350Mbps connection vector sets were running at 95% of the link capacity.

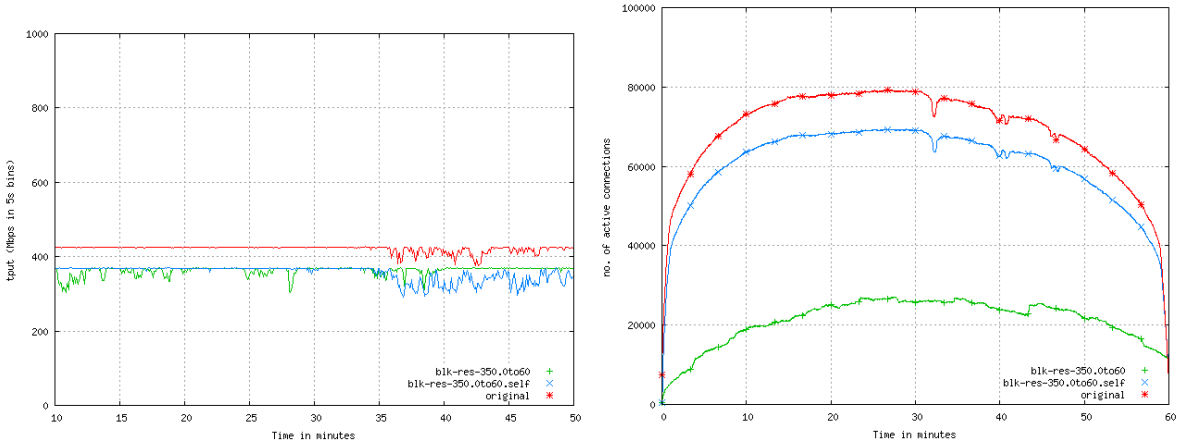


Figure 11. Comparison of replay of original connection vectors (red) with traces rescaled to 350Mbps using Self-Block Resampling (blue) and Randomized Block Resampling (green). *Left:* Throughput time series. Note the original experiment was constrained at the bottleneck link to 424Mbps, while the block resampled experiments were constrained to 368Mbps. *Right:* Active connection time series.

The self-block resampled connection vector set once again closely resembled the throughput and active connection patterns of the original connection vector set. Throughput shifted downward to 350Mbps, yet still maintained the pattern of non-stationarity exhibited in the original trace, with the replay running at the link capacity until the 35th minute and dipping below afterward. In the same manner, the active connection plot for the self-block resampled set of connection vectors exhibited the same smooth increase and decrease over the duration of the replay. This is in contrast to the

randomized block resampled replay, which preserved neither the pattern of non-stationarity observed in the original trace nor the magnitude of active connections from the original replay.

The pattern of router queue lengths was once again only preserved in the self-block resampled replay. Despite having the same level of congestion as the original replay, the replay of both scaled connection vector sets exhibited substantially lower congestion over the duration of the trace. The median queue length for the self-block resampled and randomized-block resampled replays were 550 packets and 563 packets, respectively. The self-resampled replay appears to have captured the shape of the original distribution more accurately, but in this case the difference between the two resampling techniques is not significant. Connection durations were once again not substantially different in any of the cases.

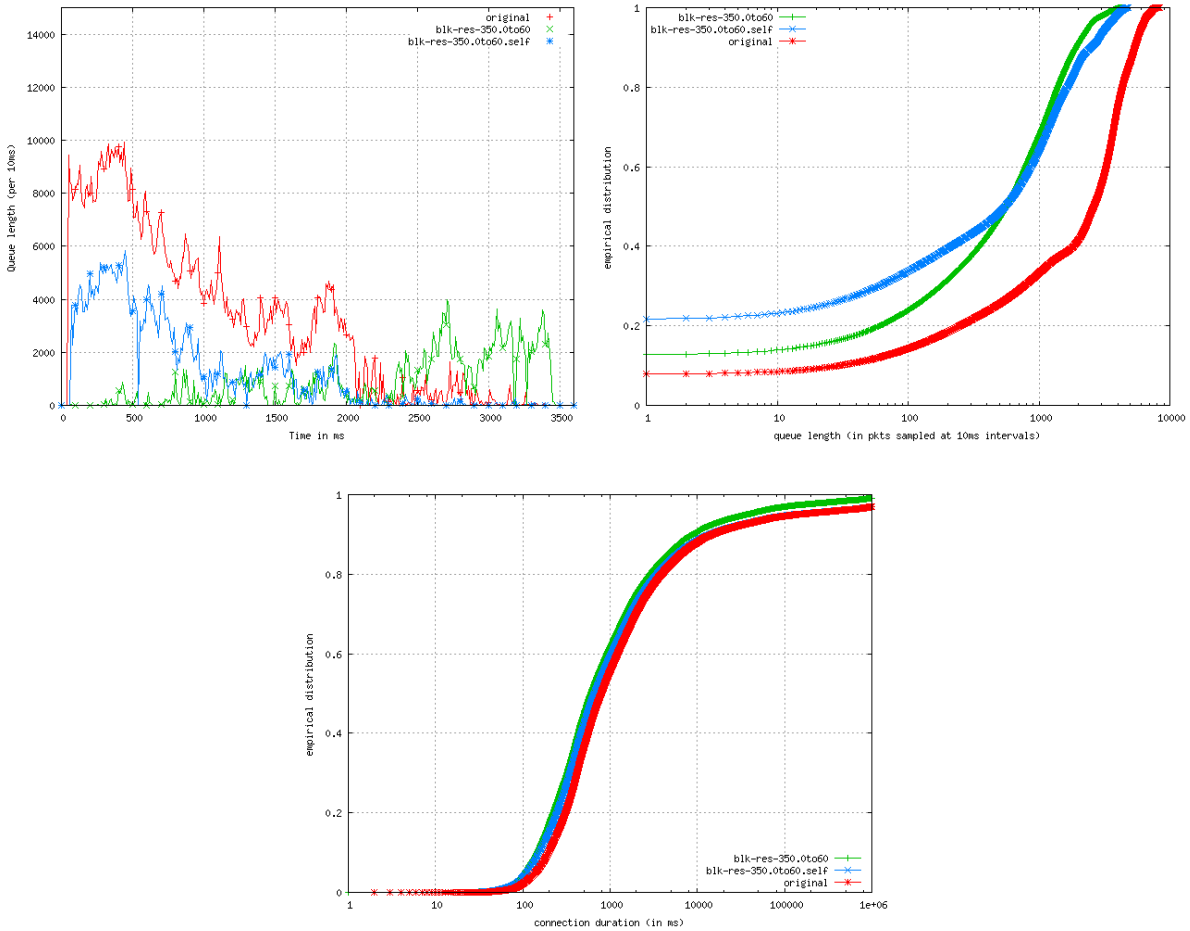


Figure 12. Comparison of replay of original connection vectors (red) with traces rescaled to 350Mbps using Self-Block Resampling (blue) and Randomized Block Resampling (green). *Top left:* Queue length time series. *Top right:* Cumulative distribution of queue lengths. *Bottom:* Cumulative distribution of connection durations.

In any case, the connection vector set scaled with *SelfBlockResample* appears to preserve all of the measured performance indicators at least as well as that which was scaled with *RandomizedBlockResample*. Thus, it appears that random reordering blocks of connection vectors is responsible for creating the unpredictable experimental results for second-order performance indicators seen in the previous section. Indeed, it is likely the case that random addition and removal of connections also introduces unpredictable variability into network traces, though at a significantly smaller level.

VI. Future Directions

While I have demonstrated that block resampling introduces significant unpredictable variability into resultant traces, the reason this variation is occurring remains unclear. *SelfBlockResample* offers a potential solution, creating resultant traces that are much more faithful to second-order performance metrics that characterize the original trace. Despite this, *SelfBlockResample* itself still relies upon random uniform sampling of connection vectors to actually scale a trace to a new offered load. While it preserves the bulk of the structure of the original trace in this process, the randomized sampling once again produces unpredictable results: simply adding and removing connections to a set of connection vectors disregards any relationship that may be present among those connection vectors. Intuitively, we know that connections do not randomly start and end: different application workloads generate patterns of connections. Consider, for instance, an email client that connects to a server periodically over the course of a day, or a peer-to-peer client that maintains dozens of connections over the duration of a file transfer. Each of these applications generates a pattern of connections that have a relationship to one another, and randomly adding or removing connections from a network trace disrupts this underlying traffic structure.

At the end of the day, our goal in scaling network traffic is simulating additional users on a network, or at least simulating some additional workload. The traffic created by those users or that workload will have some structure to it that random sampling does not capture. Thus, it seems that a more sophisticated approach to scaling would take this "session structure" into account, rather than operating simply on a connection basis. Unfortunately, extracting this information from a raw packet trace is a challenging problem, and would in itself likely require substantial effort.

One area for further exploration using the existing block resampling techniques would be to take into account both the start and end times of connections. Both *RandomizedBlockResample* and *SelfBlockResample* only consider the start times of connections. Approximately 8% of connections persisted for more than 30 seconds, the block duration used throughout this work; 2.5% of connections last over 30 minutes! Connections that span multiple blocks are only considered in the first block in which they appear. Thus, multiple long-lived connections could be placed in subsequent blocks and raise the offered load for some subset of the full trace duration to a value substantially higher than the targeted average offered load of the entire trace. This is particularly likely when entire blocks of connections are randomized to create a new one, as is done in randomized block resampling.

In fact, this also motivates the need to evaluate the effects of scaling traces at different time scales. Because some connections spanned a substantial portion of the entire network trace, a one hour trace may be too short of a time scale on which to evaluate the traffic behaviors seen here; the connection vectors used here may not reach an effective "steady state", free from start-up effects of the experiment or artifacts from artificially completing connections that started before or after the original trace was captured. A longer trace might also permit longer block durations to be explored. If block duration were long enough to capture the duration of a user session, this could allow for rough approximation of a "session driven" offered load scaling technique.

Finally, some of the differences may be explained by the connection vectors simply not being faithfully generated by Tmix. Evaluation of this was out of the scope of this paper but would be vital to rule out in future investigation.

VII. Conclusions

Scaling offered load remains a highly desirable capability for networking researchers and administrators. Unfortunately, existing methods for performing this scaling can produce resultant traces that are radically different than their original input trace. The technique I have discussed in this work, block resampling, succeeds in scaling offered load of a set of connection vectors, but it does not preserve important second-order characteristics of the original network trace from which those connection vectors derive. I have further shown that randomized block resampling is a non-transitive process, and that it is a nondeterministic, even unpredictable, process.

The *SelfBlockResample* technique greatly reduces this variation while accurately performing offered load scaling. However, this approach still relies upon random sampling of individual connections, and does not account for relationships among connection vectors. Moreover, by not performing randomization of connection vector blocks while building a resultant trace, *SelfBlockResample* is unable to introduce significant variation into a set of connection vectors. While this makes the technique more predictable, it may or may not be a desirable outcome depending on the intentions and requirements of the researcher. In general, *SelfBlockResample* produces output sets of connection vectors whose second-order characteristics more closely mirror that of the original trace than those produced by *RandomizedBlockResample*. Finally, I have proposed a series of extensions to the block resampling techniques discussed here that may more accurately resolve the problem

The direct reasons for the variability introduced via randomization remains unclear; this is left as an area for future work. However, if nothing else, this work has demonstrated that simply modifying the offered load of a network trace can have significant and unforeseen effects upon second-order performance metrics. Indeed, the entire concept of "realism" in network traffic generation is poorly

defined at best, and as a result the findings of this work should not be taken as a repudiation or endorsement of one particular technique. Instead, I have sought to more fully describe the non-obvious, arguably counter-intuitive, effects that scaling the offered load of a network trace can have on experiments that make use of such traces. There remains substantial work to be done to more fully understand what scaling network traffic means, as well as to understand how to model network traffic in such a way that it can be reliably and realistically scaled. Hopefully this work can serve as a guide for part of this continuing research.

VIII. References

[FP01] S. Floyd and V. Paxson, *Difficulties in simulating the internet*, IEEE/ACM Transactions on Networking, 9(4):392–403, August 2001.

[HC06] F. Hernandez-Campos, Generation and Validation of Empirically-Derived TCP Application Workloads, PhD thesis, University of North Carolina at Chapel Hill, August 2006.

[WAH+06] M. C. Weigle, P. Adurthi, F. Hernandez-Campos, K. Jeffay, and F. D. Smith, *A tool for generating realistic TCP application workloads in ns-2*, ACM Computer Communication Review, 36(3):67–76, July 2006.