Evaluating Distributed Systems: Does Background Traffic Matter?

Kashi Venkatesh Vishwanath and Amin Vahdat
University of California, San Diego
{*kvishwanath,vahdat*}*@cs.ucsd.edu*

Abstract

Evaluating novel networked protocols and services requires subjecting the target system to realistic Internet conditions. However, there is no common understanding of what is required to capture such realism. Conventional wisdom suggests that competing background traffic will influence service and protocol behavior. Once again however, there is no understanding of what aspects of background traffic are important and the extent to which services are sensitive to these characteristics.

Earlier work shows that Internet traffic demonstrates significant burstiness at a range of time scales. Unfortunately, existing systems evaluations either do not consider background traffic or employ simple synthetic models, e.g., based on Poisson arrivals, that do not capture these burstiness properties. In this paper, we show that realistic background traffic, has qualitatively different impact on application and protocol behavior than simple traffic models. One conclusion from our work is that applications should be evaluated under a range of background traffic characteristics to determine the relative merits of applications and to understand behavior in corner cases of live deployment.

1 Introduction

There has been significant interest in understanding the characteristics of network services and protocols under realistic deployment conditions [2, 3, 4, 13]. Application developers typically have two high-level options when evaluating their prototype: live deployment or emulation/simulation. Live deployment on a testbed such as PlanetLab allows developers to subject their system to realistic network conditions and failures. However, experiment management/control and reproducibility become more difficult. Further, while the deployment environment is realistic, there is no guarantee that it is representative or that any series of experiments will experience a range of desired potential network conditions.

Emulation and simulation on the other hand simplify experiment management and make it relatively easy to obtain reproducible results. However, while recent emulation environments [15, 20] allow unmodified applications, they must still simulate some target network conditions, including topology, failure characteristics, routing, and background traffic. Practitioners are left with the unenviable task of developing appropriate models for each of these important network characteristics. Thus, while emulation offers the promise of evaluating applications under a range of conditions, the huge space of potential conditions makes it difficult, if not impossible, for most developers to take advantage of this flexibility.

One long term goal of our work is to enable developers to use emulation to evaluate their applications under a range of scenarios with realistic models of traffic, topology, routing, and host characteristics. We leave a study of the relative sensitivity of applications to different aspects of the network, e.g., routing protocols versus traffic characteristics [21, 19] versus topology [7, 11], to future work. Our goal in this paper is to understand application sensitivity to background traffic. It is clear that applications behave differently when competing with other traffic. Thus, it is not surprising that researchers have begun to include background traffic models in their evaluations (see Section 2 for a summary). However, it is also well-known that Internet traffic has rich and complex properties not captured by models for background traffic, e.g., self-similarity and burstiness at a range of timescales [8, 22]) not captured by the simple models employed by practitioners. A natural question to answer then, is whether these complex but realistic models, warrant attention as candidates for background traffic.

In this paper, we quantify application sensitivity to a range of background traffic characteristics. That is, are simple models of background traffic, such as constant bit rate, Poisson arrivals, or deterministic link loss rates, sufficient to capture the effects of background traffic? Or do we require more complex background traffic models that capture the burstiness on a particular network link? We begin with a literature survey to understand the common techniques for modeling background traffic. We also leverage recent work [19, 21] on modeling and recreating background traffic characteristics for existing Internet links. Using accurate, real-time network emulation, we subject a number of applications to a spectrum of background traffic models and report variations in end-to-end application behavior.

We find qualitative differences in application behavior when employing simple synthetic models of background traffic rather than realistic traffic models. We investigate the cause of this difference and present our initial findings. *Specifically, we find that differences in burstiness between two background traffic models at a range of timescales significantly impacts overall application behavior, even when network links are not congested.* Existing synthetic models for background traffic do not demonstrate the rich variances in the packet arrival process for competing traffic present in live network traffic. Thus, studies employing these synthetic models may mischaracterize the impact of background traffic. *Further, we find that even seemingly small differences in the burstiness of background traffic for realistic traffic models can lead to important differences in application behavior.* Hence, one conclusion of this work is that studies of network application behavior should include experiments with a range of realistic background traffic models. Ideally, the research community would evolve a suite of background traffic models representative of a range of network conditions and locations. We hope that our findings in the paper will serve as a means to spur sufficient interest in the community to collectively develop such an appropriate benchmark suite in the future.

In summary, this paper makes the following contributions. This work is the first to quantify the impact of a range of background traffic characteristics on a number of applications. Prior to this work, it was not possible to deterministically subject application traffic to a range of realistic network conditions while accounting for the complexity of real network traffic, e.g., as determined by TCP. We present a methodology for doing so and use this methodology to carry out a systematic sensitivity study of applications to a range of network characteristics. We show that techniques such as replaying a pre-existing trace packet-by-packet do not exhibit the responsiveness of real Internet traffic. Similarly, we show that common models for generating background traffic, such as transmitting traffic at a constant bit rate, traffic with a Poisson arrival process, or deterministically setting loss rates to network links has significantly less impact on application traffic than realistic Internet traffic.

Investigating the cause of these observations, our detailed performance evaluation shows that the properties of Internet traffic, in particular its burstiness across a range of time scales, can have unpredictable impact on a range of applications relative to simpler traffic models. We establish that it is not enough to simply use “some” bursty source as a background traffic model. As another example, we reproduce the results of an earlier study that employed synthetic traffic models to compare bandwidth estimation tools. After validating the original results, we found that some of the conclusions of the earlier study may have been reversed when employing realistic traffic models.

2 Motivation and Related Work

Consider the problem of determining the sensitivity of a given application to a range of background traffic characteristics. One approach would simply be to run the application across the Internet on a testbed such as PlanetLab. Unfortunately, the difficulty to measure the characteristics of background traffic at any point in time is compounded by the fact that one cannot guarantee reproducing a particular set of background traffic characteristics. Finally, experiments would be restricted to the type of background traffic experienced in a particular deployment scenario, making it difficult to extrapolate to sensitivity in other settings.

Hence, a careful study of application sensitivity to background traffic must run in an emulation or simulation environment prior to live deployment. This begs the question as to what kind of background traffic to employ. One approach is to take a trace of background traffic at a particular point in the Internet and to replay that traffic packet-by-packet in an emulation environment. As we quantify later, such background traffic will not be *responsive* to the application traffic. It will blindly transmit data in a preconfigured sequence; real Internet traffic responds and adapts to prevailing traffic conditions as a result of end-to-end congestion control. Other simple approaches involve playing back traffic at a constant bit rate, according to a Poisson arrival process, or using deterministic loss rates. Unfortunately, these simple techniques are known not to reproduce the characteristics of Internet traffic and, as we quantify later, will result in incorrectly estimating the impact of background traffic.

In this paper, we present a methodology for quantifying the impact of realistic Internet traffic on a range of applications. We build on our earlier work [21] that shows how to create traffic that is both *realistic* and *responsive*. By realistic, we mean that the traffic matches the complex characteristics of traffic across some original link, including traffic burstiness at a range of timescales. By responsive, we mean that the background traffic adapts to application traffic in the same manner that they would in the wild. That is, the flows in aggregate ramp up and recover from loss in a similar manner that they would across the Internet, e.g., as determined by TCP’s response to round trip times, bottleneck bandwidths, etc.

Critical to our methodology are techniques to reproduce the application- and user-level characteristics of the flows in some original trace, e.g., session initiation according to user behavior, packet sizes according to protocol behavior, etc. We also recreate the bandwidths, latencies, and loss rates observed in the original trace. Reproducing these network conditions is important to enabling responsiveness of our generated background traffic to the characteristics of the foreground/application traffic.

Explanation	(%)	Project/Paper Title and the Conference Name
No Background Traffic	25.6	SIGCOMM ’06 - Churn in distributed systems, SpeakUp, SIGCOMM ’04 - Modeling P2P, Mercury, OSDI ’04 - FUSE, NSDI ’05- Quorum, Low bandwidth DHT routing, NSDI ’04 - Macedon, Thor-SS, SIGCOMM ’07 - Structured streams, NSDI ’07 - SET
Constant Bit Rate Traffic	2.33	SIGCOMM ’04 - CapProbe
Fixed Loss Rate	2.33	OSDI ’04-FUSE
Lowered Link Capacity	2.33	NSDI ’06 - DOT
Only Latencies	2.33	NSDI ’06-Colyseus
“Some” Background Flows	2.33	NSDI ’05 - Trickles
“Some” TCP source	2.33	SIGCOMM ’04 - CapProbe
Custom Built Simulator	4.65	NSDI ’05 - Myths about structured/unstructured overlays,Glacier
Pareto Flow Arrival	4.65	SIGCOMM ’05 - VCP, NSDI ’06 - PCP
Fixed Length Flows	2.33	NSDI ’06 - PCP
Long Lived flows	4.65	SIGCOMM ’05 - TFRC, SIGCOMM ’07 - PERT
LRD Traffic	2.33	SIGCOMM ’04 - CapProbe
Pareto Length Flows	2.33	NSDI ’06 - PCP
SpecWeb	6.98	OSDI ’06 - TCP offload, NSDI ’06 - Connection conditioning, NaKika.
Run on PlanetLab	18.6	NSDI ’06 - CoBlitz, OASIS, NaKika. NSDI ’05 - Shark, Botz4Sale, NSDI ’04 - Saxons, NSDI ’07 - BitTyrant, SET
Real World Deployment	9.3	NSDI ’06 - Overcite, NSDI ’04 - BAD-FS, TotalRecall, NSDI ’07 - BitTyrant
Harpoon	2.33	SIGCOMM ’05 - End-to-end loss rate measurement
RON Testbed	2.33	NSDI ’05 - MONET

Table 1: Literature survey of SIGCOMM, SOSP/OSDI and NSDI from 2004-2007.

Background. To motivate the importance of background traffic for a range of studies, we conducted a literature survey of SIGCOMM, SOSP/OSDI and NSDI from 2004-2007. We determined whether each paper contained a performance evaluation of a distributed system and, if so, what types of background traffic were employed in the evaluation. Overall, we found 35 papers that conducted a total of 43 such experiments. Table 1 summarizes a subset of these experiments, along with a descriptive project name and the publication venue. We divide the set of techniques into four main categories.

Our study is vulnerable to sampling bias, however we make the following high-level observations. More than 25% (11/43) of the experiments use no background traffic (NBG). The application to be evaluated is typically run on a cluster of machines with high-speed interconnect. Another 14% of the experiments account for congestion using simple models such as constant bit rate (CBR) traffic or simply constraining link latencies/capacities in a synthetic topology.

At the other extreme (bottom of Table 1), approximately 30% of the experiments employ live deployment on testbeds such as PlanetLab, RON, and Harpoon. Finally, in the middle of the table we have 25% of experiments that are done with some sophisticated models to account for background traffic, for instance, Caprobe experiments use Long Range Dependent (LRD) traffic.

Based on this study, we observe significant confusion in the community regarding whether background traffic is an important consideration in carrying out experimental evaluations. Further, there is no consensus as to what type of background traffic should be employed. Finally, in virtually all (29/35) papers just *one* model of background traffic is used with no analysis of application sensitivity to different background traffic conditions; this can partially be attributed to the large space of possible models of traffic.

A goal of our work is to enable the community to make informed decisions about whether background traffic should be considered in a particular scenario, and if so, the particular characteristics of background traffic that are important to consider. Thus, we consider the interaction of a variety of applications with a range of competing background traffic. Ideally, we would consider all of the background traffic models summarized in Table 1 against all of the 43 experiments that we found in those papers. In this paper, we take a few steps toward this goal. For instance, we show that CBR and Poisson traffic have very similar impact on the applications we consider and that setting probabilistic loss rates does not capture the complexity of real interactions with background traffic. We also study the impact of realistic background traffic models on application performance.

Related Work. WASP [14] is perhaps most closely related to our work in spirit. It shows that HTTP performance can be significantly impacted by setting realistic delays and loss rates for the flows. The work concludes that web services cannot be evaluated on high speed LANs, but must instead consider wide-area networking effects. Relative to this effort, we consider a number of application classes and background traffic conditions. We show that simple models of wide-area conditions (such as higher round trip times and non-zero loss rates) are insufficient to capture the effects of realistic Internet background traffic either.

Our work will benefit from ongoing work in producing realistic Internet traffic, including Tmix [9], Harpoon [19], Surge [6], and Swing [21]. We chose to generate realistic background traffic using Swing, but we expect qualitatively similar results had we employed alternative tools. We make no claims, positive or negative, about whether the traffic we generate is realistic or not. Rather, we consider a range of qualitatively and quantitatively *different* traffic conditions and show the resulting effect on application performance. However, we do like to add that before the advent of Swing, it was not possible to create realistic and responsive network traffic in a testbed environment [21]. This partially explains why researchers have been using ad-hoc traffic models in their experiments to date.

3 Methodology

We begin by describing the architecture used for carrying out the experiments followed by the list of applications we used. We then describe the background traffic models used followed by the experiments we conducted.

3.1 Architecture

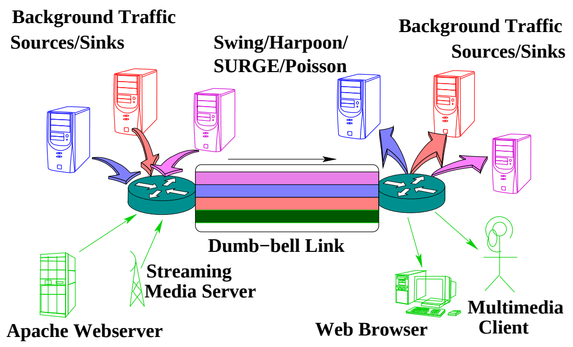

Background Traffic Sources/Sinks
Swing/Harpoon/ SURGE/Poisson
Background Traffic Sources/Sinks
Dumb−bell Link
Streaming Media Server
Apache Webserver
Web Browser
Multimedia Client

Figure 1: Evaluation architecture.

Figure 1 depicts our approach to quantifying and understanding the impact of background traffic on individual applications. We place traffic sources and sinks on either side of a constrained/dumb-bell link such that all traffic in both directions crosses the common link. Two classes of sources and sinks generate the traffic crossing the link. In the first class depicted at the bottom, nodes generate application traffic, for instance, Apache Web server and httperf clients. Then on top, we have sources and sinks responsible for generating *background* traffic for the target link, for instance using Swing [21], Harpoon [19], SURGE [6] or simpler traffic sources such as Poisson or Constant Bit Rate (CBR). Because application traffic and background traffic share the common link, we can quantify the impact on the application as a function of a range of background traffic characteristics.

In a real network environment, applications must compete with background traffic at multiple links between the source and destination. However, there are no known techniques to model desired background traffic characteristics at multiple successive links in some larger topology. For instance, there may be strong correlations between the background traffic characteristics of links in the topology. For the purposes of this study, we feel it reasonable to quantify and understand the impact of background traffic at a single link before attempting to extrapolate to more complex scenarios. In all likelihood, the effect of background traffic at multiple links will be even more pronounced than our findings. Hence, our results should be interpreted as a conservative estimate of the effects of background traffic, while still demonstrating application sensitivity to varying background traffic characteristics.

3.2 Applications

Of course, the impact of background traffic heavily depends upon the characteristics of the particular application under consideration. For this study, we chose three applications with diverse communication patterns and requirements: Web traffic, multimedia streaming, and end-to-end bandwidth estimation. Note that each of these applications exercise one end-to-end path, and hence, we explicitly omit more distributed applications such as BitTorrent that simultaneously exercise multiple independent Internet paths. While this class of application is important, considering complex topologies is beyond the scope of this paper (see above). We did run experiments (not discussed further here) for the case where all BitTorrent clients were subject to a dumbbell topology; these results were qualitatively similar to our findings for HTTP.

Web Traffic For Web applications, we set out to determine the effect of background traffic on the response times perceived by end clients. We placed a single Apache Web server on one side of the dumbbell (e.g., the bottom left in Figure 1). We programmed httperf clients to fetch objects of various sizes from the server and placed them on the other side of the dumbbell. The links connecting the clients and server to the dumbbell have large capacity and low latency, such that the dumbbell is the constrained link for all client-server communication (we vary the capacity of the dumbbell link in various experiments described below).

To generate background traffic, we place sources and sinks of the appropriate traffic generator on either sides of the target link (top left/right in Figure 1). We set the bandwidths, latencies, and loss rates of the links connect-

Trace ↓	**Secs**	**Trace BW** **Aggregate** (Mbps)	**Dir0** (Mbps)	**Trace Collection Date**	**Number of flows**	**Dominant applications**	**Unique IPs** (1000s)
Auck	600	5.5	3.3	June 11, 2001	155 K	HTTP, SQUID	3
Mawi	900	17.8	7.8	September 23, 2004	476 K	HTTP, RSYNC	15
Mawi2	900	11.9	10.8	December 30, 2003	160 K	HTTP, NNTP	8

Table 2: Trace characteristics for three different links.

ing background traffic sources and sinks based on the traffic generation model. For instance, we simply play back CBR traffic over unconstrained links; whereas for Swing, we assign assign latencies, bandwidths and loss rates based on observed path characteristics in some original packet trace [21].

Multimedia Traffic The second application class we consider is video streaming. Video clients are sensitive to the arrival times of individual packets, whereas web clients are typically sensitive to end-to-end transfer times. Overall, we wish to quantify the impact of various types of background traffic on client-perceived video quality. For streaming audio/video we use the free version of Helix on the server side and Real Player on the client side. We generate background traffic across the dumbbell topology as with Web traffic.

Bandwidth Estimation Tool We chose bandwidth estimation for our third application. While not an end application, it displays fundamentally different characteristics than our first two applications and is a building block for many higher-level services. We employ Pathload [10], and pathChirp [17] tools for our study. We place bandwidth senders and receivers along with competing traffic generators across the dumbbell topology identically to our configuration for Web and video streaming.

3.3 Traffic Generation

We consider four techniques for generating competing background traffic, in increasing order of complexity and realism. First, for constant bit rate (CBR) traffic, we wrote simple sources to generate packets at a specified rate to sinks on the opposite side of the dumbbell link. In aggregate, the sources generate a target overall rate of background traffic. Second, for Poisson traffic, we modify the sources to generate traffic with byte arrival per unit time governed by a Poisson process with a given mean. We evaluated variants of CBR and Poisson using both UDP and TCP transports.

While CBR and Poisson processes do not capture the complexities of real Internet traffic [16], we wish to quantify the resulting differences in end-to-end application behavior relative to more realistic, but complex traffic generation techniques. Hence, for our third technique, we modify the sources and sinks to play back packets in the exact pattern specified by an available tcpdump of traffic across an existing Internet link (Table 2). One drawback of this approach is that the generated background traffic is not congestion responsive. That is, the traffic will be played back in exactly the same pattern as the original trace irrespective of the behavior of the application traffic. Another drawback is that it is difficult to extrapolate to alternative, but similar, scenarios when playing back a fixed trace (e.g., changing the available bandwidth across the constrained link, the distribution of round trip times between sources and sinks, etc.).

Thus, for our fourth technique, we use Swing [21] to generate responsive and realistic network traffic. Swing is a closed-loop, network responsive traffic generator that qualitatively captures the packet interactions of a range of applications using a simple structural model. Starting from a packet trace, Swing automatically extracts distributions for user, application, and network behavior. It then generates live packet traffic corresponding to the underlying models in a network emulation [15, 20] environment running commodity network protocol stacks. Because Swing generates the traffic using real TCP/UDP stacks, the resulting traffic is responsive both to foreground traffic and varying characteristics of the constrained link and end-to-end loss rates/round trip times.

Swing extracts a range of distributions from an original trace to model user behavior, e.g., think time between requests, application behavior, e.g., distribution of request and response sizes, and network characteristics. One particularly important aspect of network traffic that Swing is able to reproduce is burstiness in the packet arrival process at a range of time scales [21]. Doing so requires Swing to assign latencies, bandwidths, and loss rates to the links leading to and from the shared link in our evaluation architecture based on the observed distribution of round trip times, link capacities, and loss rates in some original trace. This way, TCP flows ramping up to their fair share bandwidth or recovering from loss will do so with statistically similar patterns (including burstiness in aggregate) as in the original trace conditions.

An additional benefit of employing high-level models of user, application, and network characteristics for background traffic is that it becomes possible to modify certain model parameters to extrapolate to alternative scenarios [21]. For instance, when reducing the round

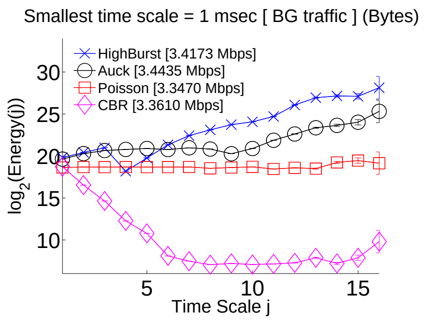


Figure 2: Auck-based background traffic (Energy plot).

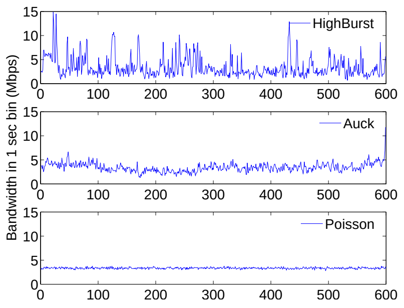


Figure 3: Auck-based background traffic (Time Series).

trip times for flows, overall burstiness tends to increase because flows tend to increase their transmission rates more quickly. We refer to alternate background traffic generated by perturbing distributions of various parameters in Swing as *variants* of the original trace.

We run all our experiments, including the sources and sinks for both foreground (Web, multimedia, bandwidth estimation) and background traffic (CBR, Poisson, Playback, Swing) in the ModelNet emulation environment [20]. For our experiments, we configure all sources and sinks of both foreground and background traffic to route all of their packets through ModelNet. Briefly, ModelNet subjects each packet to the per hop bandwidth, delay, queueing, and loss characteristics of the target topology by inspecting both the source and destination of the packet and routing it through the emulated topology. ModelNet operates in real time, meaning that it moves packet from queue to queue in the target topology before forwarding it on to the destination machine assuming that the packet was not dropped. Earlier work [20] validates ModelNet's accuracy using a single traffic shaper at traffic rates up to 1Gbps (we can operate at higher speeds by employing multiple traffic shapers in parallel).

3.4 Topology and Experiments

We run our experiments on a cluster of commodity workstations, multiplexing multiple instances on each workstation depending on the requirements. For the experiments in this paper, we use eight 2.8 Ghz Xeon processors running Linux 2.6.10 (Fedora Core 2) with 1GB memory and integrated Gigabit NICs. We generate background traffic using 1000 nodes in the emulated topology (meaning that we multiplex hundreds of emulated nodes onto each physical machine).

For example, for a 200Mbps trace, assuming an even split between generators and listeners (four machines each), each generator would be responsible for accurately initiating flows corresponding to 50Mbps on average. Each machine can comfortably handle the average case, though there are significant bursts that make it important to "over-provision" the experimental infrastructure. To avoid contention between physical resources with the traffic generator we use a separate set of machines to host the target application; for instance, httperf clients. In this fashion, any performance impact and variations that we measure results purely from network behavior.

We first describe the traces we use to generate realistic background traffic for the experiments in this paper. We use traces from Mawi [12], a trans-Pacific line, as well as an OC3c ATM link traces from the University of Auckland, New Zealand [5]. These traces come from different geographical locations (New Zealand, Japan) and demonstrate variation in application mix, average throughput, number of users etc. (see Table 2). All traces were taken on 100Mbps links. The Mawi traces were constrained to 18Mbps over long time periods (though it could burst higher).

While all original and the corresponding Swing-generated traces are bidirectional, we focus on the impact of competing traffic in one direction of traffic (Dir0 in Table 2) for simplicity in our plotted results. In other words, we design all experiments such that the dominant direction of application traffic (HTTP responses, video, bandwidth estimation packet train) matches Dir0 of the generated background traffic.

We use wavelet-scaling plots [1, 8, 22] to characterize traffic burstiness. Intuitively, these plots allow visual inspection of burstiness for a range of timescales. The x-axis of these plots shows the time scale on a log scale and the y-axis shows the corresponding energy value. Higher levels of energy correspond to more burstiness. Figure 2 plots burstiness corresponding to five variants of the Auck trace for a 20Mbps link (along with the average bandwidth for each variant in square brackets). We configured Swing to generate constant-bitrate (CBR) and Poisson traffic with the same average bandwidth as the Auckland (Auck) trace. In addition to reproducing the Auck trace using Swing, we also created a very bursty traffic variant, called *HighBurst* (HB), by setting the round trip times artificially to $4ms$ while generating traffic using Swing. The lower round trip times (relative to

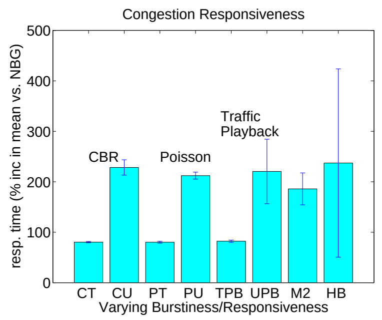


Figure 4: "Simple" models are inaccurate.

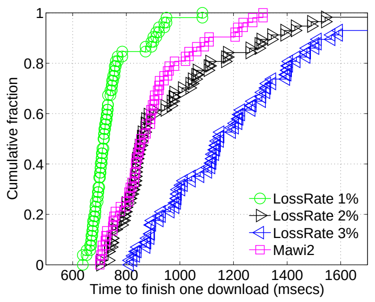


Figure 5: Deterministic loss rates never suffice.

the distribution in the original trace) means that TCP traffic ramps up more quickly, recovers more quickly from losses, etc. For an alternate visualization of the relative difference in burstiness, consider the time series plots for the same traffic models shown in Figure 3. The Poisson variant generates a relatively fixed bandwidth whereas HighBurst variant peaks to 15Mbps at times, compared to Auck. Note that while the average bandwidth consumption for all traces are comparable, finer-grained behavior varies significantly. Further, none of the variants come close to congesting a 20 Mbps link.

4 Results

We subject each of our three target application classes to the various types of background traffic described above with the goal of answering the following questions:
i) What aspects of "realism" should we reproduce? Is it sufficient to simply "replay" individual packets in some measured trace or does the generated background traffic need to be TCP responsive and react appropriately based on the end-to-end network characteristics of the traffic sources and sinks? (§ 4.1)
ii) Can probabilistic packet drops susbstitute for real competing background traffic? (§ 4.2)
iii) Does burstiness of background traffic matter or is it sufficient to reproduce average bandwidth? (§ 4.2, 4.4)
iv) Are some applications more sensitive to background traffic than others? (§ 4.3)
v) Is application behavior sensitive to slight variations in the background traffic characteristics? (§ 4.4)

4.1 Background Traffic Responsiveness

The first question we consider is the importance of realistically playing back background traffic characteristics. We consider three techniques for doing so: i) scheduling per-packet transmissions using UDP connections to match the exact timings (at 1 ms granularity) and packet arrival processes found in some original trace; ii) scheduling per-packet transmissions using a single TCP connection to attempt to match the packet arrival process found in the original trace; and iii) extracting user, application, and network characteristics from the original trace and playing back TCP flows whose communication patterns are statistically similar to the original without making an attempt to match the patterns found in the original trace.

The first technique (UDP) is not responsive to the characteristics of foreground traffic. Thus, it will not back off in the face of congestion. The second technique (TCP) is responsive, but, unfortunately it becomes impossible to perform precise packet scheduling for TCP connections. Further, because we have no knowledge of the end-to-end network characteristics for the TCP flows we play back, it is not possible to verify that the response to congestion would match the behavior of flows from the original trace (e.g., because of variations in round trip times or losses elsewhere in the network). Finally, because it employs a single connection to multiplex the behavior of a much larger number of flows, its behavior is unpredictable. The third technique, corresponding to Swing-based [21] playback, promises to be the most faithful but is also the most complex and requires more resources (i.e., logic to source and sink traffic from individual hosts) for trace playback.

To establish the accuracy for each of these techniques, we run httperf clients requesting 1 MB files from an Apache Web server sharing the bottleneck link with the Mawi2 trace. We set the shared link (the point of contention between httperf and background traffic) to 15 Mbps. We choose 15 Mbps to ensure that the background traffic attempts to consume a significant portion of available resources (§ 4.2 onwards relaxes this assumption). We are interested in understanding the response time for HTTP as a function of the characteristics of the background traffic. As described earlier, the links connecting the httperf/Apache nodes to the shared link are unconstrained (large capacity and low latency), so that traffic shaping takes place only at the target link. During each experiment we fetch files back-to-back, using a single client-server pair for 10 minutes.

Figure 4 shows the results for different classes of background traffic. For each scenario, we plot the mean and standard deviation of response time increase relative to the NBG (No Background Traffic) case. Background

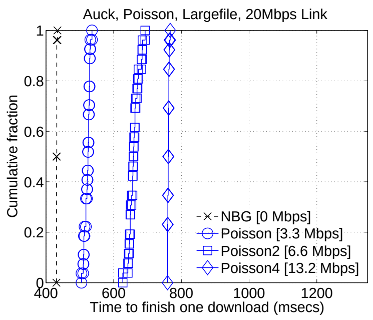


Figure 6: Poisson background traffic.

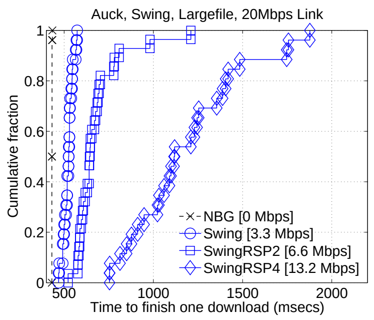


Figure 7: Swing background traffic.

traffic consumes 66% of the shared link's capacity in all cases. The first two bars plot slowdown for CBR traffic with both TCP and UDP playback (CT and CU) configured to consume as much average bandwidth as the Mawi2 trace. The next two bars show slowdown for TCP and UDP variants of Poisson (PT and PU). For both of the models (CBR and Poisson) UDP variants impacted responses time much more (200% increase vs 100% increase) than TCP variants exposing the limitation of UDP based traffic generators. Because UDP sources are not congestion responsive, they have a much larger impact on HTTP. The next two bars show the slowdown for TCP- and UDP-based playback (TPB/UPB) of Mawi2. TPB has much less impact than Swing-based playback of Mawi2 (M2), likely because its use of a single TCP connection means that the generated background traffic is less bursty. Finally, UPB results in larger slowdown than Swing-based playback because it is not congestion responsive.

Given the above results, we conclude that *simple techniques for "playing back" background traffic, such as UBP and TBP, may result in significant inaccuracy as the aggregate traffic across a link approaches the link's capacity.* Thus, for the remainder of this paper, we employ Swing to play background traffic corresponding to some original network condition and compare it to other variants such as CBR/Poisson.

Another popular technique in the literature for capturing the complexity of real background traffic is to set loss rates for particular links, with the goal of capturing the effects of losses caused by competing traffic. To determine whether this technique could capture the effects of more complex traffic scenarios, we next measure the performance of httperf when setting various loss rates for the shared link. In all other respects, this experiment is identical to the case where we run with no background traffic. Figure 5 shows that httperf's behavior when crossing a link with a range of loss rates differs from its behavior when competing with realistic Mawi2 traffic, again, across a 15Mbps bottleneck link. For instance, with losses of 1% the CDF of retrieval times is too far to the left of Mawi2. On the other hand, higher loss rate settings, while shifting the CDF to the right, increase the size of the tail too. We hypothesize that the *difference in behavior results from the independent nature of losses when setting any fixed loss rate (relative to the bursty losses common for Internet links) and the fact that setting loss rate alone does not capture any of the effects of increased queueing delay (and hence increased round trip times) leading up to the point of loss.*

4.2 Httperf/Apache

Having established the appropriate technique for transmitting background traffic, we now turn our attention to understanding the impact of background traffic on HTTP transfers. In order to arrive at a conservative estimate of the impact of background traffic we first experiment with the Auck trace (lowest bandwidth and least bursty). We begin by examining the impact of varying bandwidths of background traffic (based on Auck) on httperf performance. We fetch 1 MB files across a 20Mbps link and vary the load placed by background traffic by generating traffic for three different average throughputs (3.3, 6.6, and 13.2 Mbps) corresponding to variants of the Auck trace. In the baseline (3.3 Mbps) case, we employ Swing parameterized by the original Auck trace. In the alternative cases, we modify the distribution of response sizes such that the average bandwidth increases by a factor of 2 and 4 (traces SwingRSP2/SwingRSP4), resulting in average bandwidths of 6.6 Mbps and 13.2 Mbps respectively. We compare the impact of Swing-generated traffic to those of TCP-generated CBR and Poisson traffic of the same average bandwidth.

Figures 6 and 7 show the CDFs of download times corresponding to various bandwidth/burstiness combinations for background traffic. Along with the legend name we show in square brackets the average bandwidth of the background traffic for reference. The effects of CBR and Poisson traffic are similar, so we only plot the results for Poisson (relative to Poisson, the CBR curves are virtually vertical with the same median value). As shown in Figure 6, the impact of Poisson (and hence CBR) traffic is almost entirely predicted by the average level of back-

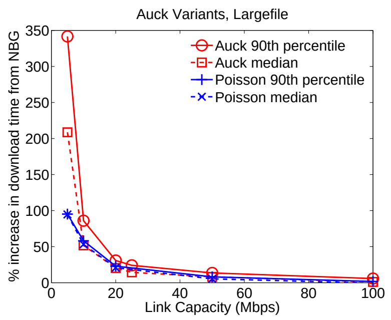


Figure 8: Varying link capacities.

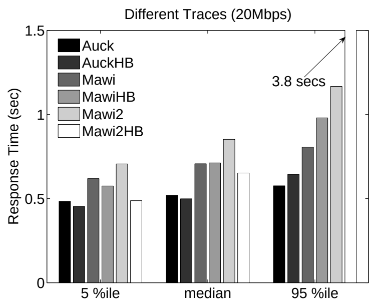


Figure 9: Simply generating "bursty" traffic is not enough.

ground traffic bandwidth. The distribution of download times shifts to the right, but with little variation in response times as the average level of background traffic increases. The Swing-generated Auck trace has a more varied impact on application performance (Figure 7). With low levels of competing traffic (3.3 Mbps), the distribution of download times is similar to Poisson. Also of interest is the fact that the performance for the 5th-percentile of retrievals is actually faster for Auck than for Poisson traffic for both the 6.6 Mbps (SwingRSP2 vs. Poisson2) and the 13.2 Mbps (SwingRSP4 vs. Poisson4) bandwidths. In these cases, the flows were lucky to be subject to less competing traffic than their counterparts in the Poisson trace. Bursty traffic means that periods of high activity coexist with periods of lower activity. Moving to higher levels of average utilization, the curves become significantly skewed. For instance, the 90th percentile of download time for the Auck (AuckRSP4) trace at 13.2 Mbps is 1484 ms compared to 759 ms for Poisson (Poisson4) traffic.

Thus, less bursty background traffic means that performance is governed by the average amount of available bandwidth and, expectedly, there is relatively small variation in download times across individual object retrievals. When background traffic is steady, HTTP performance is predictable. As burstiness increases, the mean download time increases, as does the variation in performance. Some flows can get lucky, behaving almost as if there is no background traffic; while others may be unlucky with significantly worse performance than the mean.

Next, we consider the sensitivity of HTTP performance to background traffic characteristics as a function of the fraction of shared link capacity occupied by the background traffic. That is, it could be the case that background traffic characteristics (e.g., burstiness) are only important when consuming a significant fraction of link capacity. Figure 8 shows the impact on download times for varying levels of background traffic with average bandwidths of 3.3Mbps (same as the Auck trace). The y-axis shows the slowdown (median as well as 90th-percetile) of HTTP transfers of large 1 MB files relative to the NBG case as a function of the capacity (5-100 Mbps) of the shared link on the x-axis. On the left side of the graph we have cases corresponding to a highly utilized link; while on the right side, the background traffic consumes a small fraction of overall capacity. For large files and low levels of link utilization, the graphs show that burstiness of background traffic does not matter. For instance, for a 50Mbps link the impact on median response time is independent of the burstiness; the slowdown largely corresponds to the fraction of the link consumed by the background traffic.

When background traffic consumes a significant portion of link capacity however, it is sufficient to cause significant losses for the HTTP transfers. Thus, at the 90th percentile of slowdown, we find that transfers competing with bursty traffic completed significantly more slowly, around a factor of 1.5 for Auck, than with a less bursty traffic source, i.e., Poisson. However, unlike median response times, this relative ordering is present even when overall capacity is high (e.g., 100 Mbps). *Thus, for large files, burstiness of traffic matters at all levels of link utilization, but more so at high levels of utilization.*

The impact on transfer times for different file sizes as a function of different levels of burstiness of background traffic is futher explored in § 5.2. *Overall we conclude that impact on download time for web transfers is a function of the size of the download and the average bandwith of background traffic as well as its burstiness.*

Finally, we consider whether it is sufficient to simply playback a "bursty" traffic source or whether traffic sources with different burstiness characteristics will have different impacts on HTTP performance. Thus, we considered the impact of six different bursty background traffic characteristics competing for a shared 20 Mbps link. We considered background traffic corresponding to Auck, Mawi, and Mawi2. We further modified each of these sources to be high burst variants ("HB") by setting the round trip times for all flows to $4msec$ while generating traffic using Swing. We plot the slowdown of HTTP transfers (1MB file) relative to the NBG case at the 5th, median, and 95th percentile in Figure 9.

There are a number of interesting results here. First,

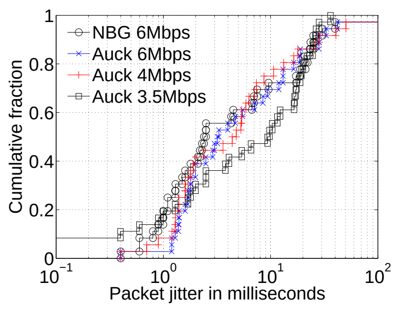


Figure 10: Varying capacity.

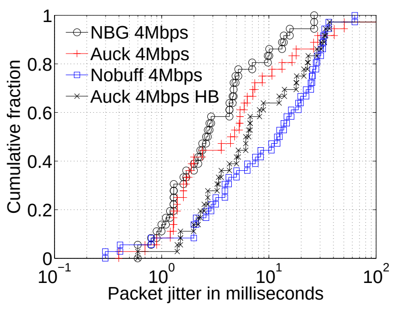


Figure 11: Varying burstiness and client-buffering.

the Mawi2 trace appears to have a larger impact on HTTP than Mawi, even though it consumes significantly less aggregate bandwidth (see § 5 for explanation). Further, the HB-variant of Mawi2 has significantly less impact on HTTP performance at the 5th and 50th percentile, but more than a factor of 5 more impact at the 95th percentile. Its highly bursty nature means that a significant number of flows are lucky and suffer comparable slowdown to the less bursty cases. However, a number of flows are extremely unlucky and suffer large slowdowns. The tail for the Mawi2HB case is significantly longer as well, an undesirable character for HTTP transfers where the humans in the loop typically value predictable behavior. *This experiment shows that although burstiness of traffic is important to consider, simply reproducing "some" bursty traffic is not enough.*

4.3 Multimedia

We next consider the impact of background traffic on multimedia traffic. We aim to understand the impact as a function of the capacity of the link, the burstiness of the generated traffic as well as the amount of client-buffering. We run the publicly available Helix Server to serve real media content to RealPlayer. As with all our experiments, the application traffic competes with various types of background traffic at the shared link. Like httperf, we consider 1 client-server pair. We tested with various streaming rates but the results in this paper are for a $450Kbps$ CBR stream encoded using RealVideo codec at 30 frames per second that runs for 62 seconds.

One important question for multimedia traffic is the "metric of goodness" for a multimedia stream, which should correspond to the quality of the video played back. However, developing such a metric based on the data stream received at clients is challenging. Thus, as a proxy for such a quality metric we use a range of statistics based on the stream delivered to the client and corroborate it with visual inspections. For a 450 kbps video stream, we can roughly assume that receiving more than 450 kbps of instantaneous bandwidth results in acceptable video quality at that particular point in time.

Another important metric of interest is *jitter*, the time between successive packets received by the client. Lower levels of jitter will correspond to higher video quality. For our first experiment, we run with Auck background traffic (3.3 Mbps average) and compare its impact on jitter for shared links with 3.5 Mbps, 4 Mbps, and 6 Mbps capacity(Realplayer was insensitive to background traffic at higher levels of link capacity). Figure 10 shows the distribution of inter-packet timing (jitter) for this experiment. We also perform an experiment in the absence of any background traffic (NBG) for baseline. For a 6 Mbps target link, background traffic does not change the quality of video (inspected by viewing the video) whereas for a 4 Mbps link the degradation is moderate. It is only when we constrain the link capacity to 3.5 Mbps that video quality suffers significantly. *One conclusion is that unless we are operating in extreme scenarios (available bandwidth approximately equal to the bandwidth of the stream), RealPlayer is relatively insensitive to bursty background traffic.*

To test this hypothesis, we next attempt to stress the limits of RealPlayer. We modify the client-side implementation to reduce the amount of buffering from a default of 10 seconds to 0 seconds. Figure 11 plots the distribution of jitter with and without buffering for the 4Mbps link. With 10 seconds of buffering, background traffic had no impact on jitter (Auck 4Mbps). However, there is significant impact (verified by visual inspection) when we remove the buffering. Without buffering, the real server attempts to retransmit lost packets more aggressively in order to meet real time deadlines, consuming more network resources and negatively impacting overall jitter. With sufficient buffering, the server can afford to be more relaxed about retransmissions, resulting in an overall smoother transmission rate.

Finally, we consider the highburst background traffic source by setting RTTs in Swing to 4msec. In this case, flows ramp up and down very quickly causing bursty traffic on the shared link. Figure 11 also plots the distribution of jitter corresponding to this experiment. The result shows that even for tight links and bursty background traffic the performance degradation in realplayer is moderate (again verified by visual inspection).

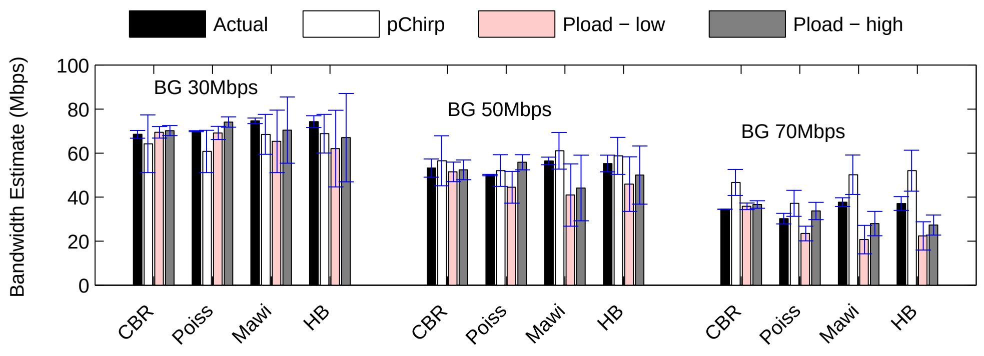


Figure 12: Augmenting to and validating pathload vs. pathchirp experiments from [17].

Overall, we conclude that for RealPlayer, unless we are operating in regimes of low buffering, low available bandwidth or very high burstiness, any reasonable background traffic model should suffice in performance evaluations for systems similar to RealPlayer; in most cases, RealPlayer's default buffering will likely result in acceptable quality of service for a range of background traffic characteristics, as long as sufficient average bandwidth levels are available. Interestingly, RealPlayer's (successful) buffering and retransmission scheme is informed by significant experience with live Internet deployments subject to a range of bursty competing cross traffic.

4.4 Bandwidth Estimation

For our final experiments, we consider the sensitivity of bandwidth estimation tools to background traffic characteristics. We use Pathload [10] and pathChirp [17] for our study because they are publicly available and because recent studies indicate that they are among the most accurate for bandwidth estimation [18]. To determine the sensitivity of existing tools to a range of background traffic characteristics, we repeat experiments from earlier published work comparing the relative merits of Pathload to pathChirp [17]. In these experiments, the authors employed Poisson and CBR models for background traffic. Below, we show that at least the conclusions from this earlier work may be reversed if the experiments had considered more realistic background traffic characteristics.

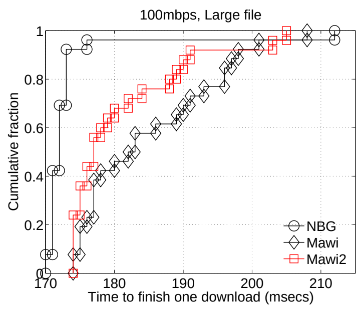
For our experiments, we overprovision all links such that the available bandwidth measured for the path is determined by the bandwidth available on the shared link in our model topology. We set Pathload's timeout to five minutes, and we report the average and standard deviation for the *low* and *high* estimates of available bandwidth across 25 runs. In practice, a small spread between the low and high estimates of available bandwidth and a low standard deviations for multiple reported values reflects likely accurate bandwidth estimates. PathChirp, on the other hand, periodically outputs an estimate of the available bandwidth and does not distinguish between a low and a high value.

Given our exact knowledge of the generated background traffic, we also calculate the true values of the available bandwidth for one second bins. In the graphs that follow, we plot the percentage error reported by each bandwidth estimation tool relative to our calculated values for available bandwidth.

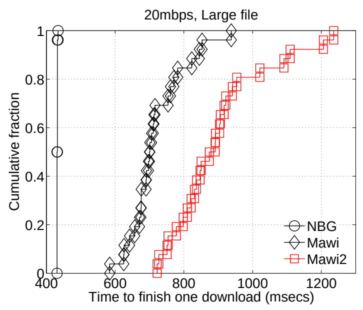
The experiments from [17] compare available bandwidth across a 100Mbps link as measured by pathload and pathchirp against various flavors of background traffic. They employ CBR UDP traffic and UDP Poisson traffic to create three different scenarios with available bandwidths of 30, 50 and 70 Mbps. Figure 12 shows these results. Additionally, we also playback Mawi, and a HighBurst variant of Mawi (labeled HB) for the three levels of available bandwidth. In each case, we increase the number of user-sessions (in Swing) by an appropriate value to match the levels of bandwidth consumed by CBR and Poisson traffic. For instance, to get the available bandwidth of 70mbps, we multiplied the number of sessions for Mawi by 3.3 times.

We initially discuss the results for Poisson and CBR models, reproducing the earlier experiments [17]. First, we confirm the earlier result [17] that when background traffic is low (BG 30Mbps) or moderate (BG 50Mbps) the estimates of pathload as well as pathchirp are close to the actual values. We also observe that indeed pathchirp takes 10-20% the amount of bytes consumed by pathload to arrive at similar estimates (not shown here). However, we note that a couple of results differed. For instance, pathchirp overestimated the bandwidth for a heavily utilized link, ie. when background traffic was around 70 Mbps. Similarly, for low link utilization (BG 30Mbps), pathchirp is slightly less accurate than pathload.

We next move to the effects of bursty background traffic, not considered by the earlier work, the bars corresponding to Mawi and HB in Figure 12. We make a number of new interesting observations here. First, the results for both tools degrade when competing with more bursty



(a) 100Mbps Link



(b) 20Mbps Link

Figure 13: Significance of traffic in the reverse direction.

traffic sources, indicated by either an inaccuracy in the mean or large standard deviation. PathChirp appears to be a bit less sensitive to burstiness of background traffic. We also no longer see that pathchirp is always conservative in its estimates. In fact, when the background traffic occupies a significant portion of the link we see that pathchirp overestimates available bandwidth, e.g., Mawi/HB traffic with 70 Mbps background traffic. The authors [17] saw that both pathchirp and pathload estimates are close to each other, however, we see that there are times when the estimates are very different from each other; for instance, Mawi for 50 Mbps.

The conclusion from these results is that it is difficult to predict a priori which bandwidth estimation tool gives better results and that realistic network traffic characteristics can impact study results. System behavior is a function of the average amount of background traffic as well as burstiness. For instance, if we hypothesize that only background traffic's average throughput is important, then we can disprove it by observing the case for 30 Mbps BG traffic. For CBR background traffic, pathload is the more accurate tool, whereas for HB, pathchirp is more accurate in its estimate. Similarly if we hypothesize that only burstiness matters, then we can disprove that by looking at HB traffic. At 30Mbps we would prefer pathchirp for HB traffic whereas at 70Mbps pathload shows superior accuracy. We leave the task of attributing the sensitivity of these tools to the underlying algorithms to orthogonal future work.

5 Case Studies

Considering our evaluation to this point, sensitivity to background traffic characteristics is application specific. Certain applications, such as bandwidth estimation are highly sensitive to background traffic while others, such as RealPlayer, are relatively insensitive except in extreme cases. We now turn our attention to some additional important characteristics of background traffic and their impact on end-to-end applications.

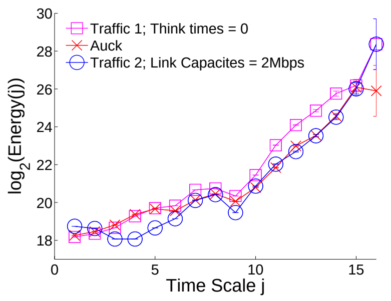
5.1 Bi-directional Traffic Characteristics

To this point, we have largely focused on traffic characteristics in one direction of a link (though in all cases, we played back bi-directional traces). We now consider a case where traffic characteristics in the reverse direction can impact application performance depending on the deployment setting. In the first experiment, we repeat retrievals of 1MB files using httperf/Apache (as Section 4.2) across a shared 100 Mbps target link. We use the background traffic models corresponding to the two Mawi traces in Table 2. The traces are in increasing order of bandwidth in the direction of flow of HTTP responses (Dir 0). One would expect that the response time distribution would correspond to the relative bandwidth of each of these traces.

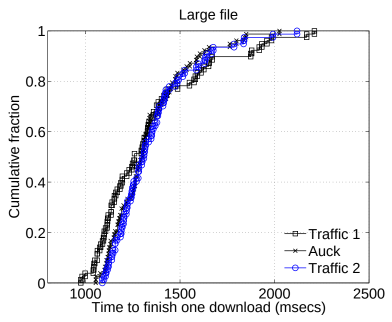
Figure 13a) plots the CDFs of retrieval times for this experiment. Mawi2 is of higher bandwidth than Mawi (Table 2) but the CDF is to the left of Mawi as a result of background traffic in the reverse path. This effect is more pronounced for Mawi as it has 10 Mbps of traffic in the reverse direction versus 1 Mbps for Mawi2. This relative ordering, however, is not present when we repeat the experiments with a 20 Mbps shared link as shown in Figure 13b). At 20 Mbps, the forward direction traffic dominates for both Mawi and Mawi2 so the effects of congestion on the reverse path is less pronounced. Thus, one simple conclusion is that that *background traffic in the reverse direction can impact application performance though the dominant direction is difficult to predict a priori.*

5.2 Burstiness at Various Timescales

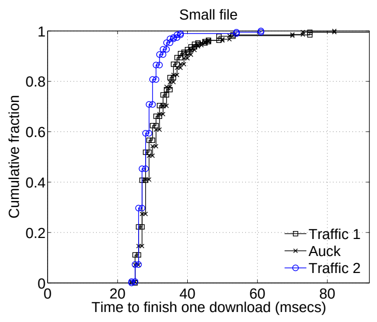
We have shown that background traffic with the same average bandwidth, but differing burstiness characteristics will have varying impact on application behavior. We now consider the question of whether burstiness at particular timescales (for instance, burstiness at millisecond versus second granularity) has differing impact on application performance. Generating traffic that selectively and precisely varies burstiness at arbitrary timescales is an open problem. However, we can alter burstiness in



(a) Burstiness of background traffic models



(b) Large files affected by large timescales



(c) Small files affected by small timescales

Figure 14: Varying burstiness at small and large timescales to understand impact on file download times.

a coarse manner at relatively small and large timescales. To vary burstiness at large timescales, we set Swing's user think time distribution to 0 for all requests; meaning, for instance, that for each user session, requests for subsequent objects will be initiated immediately after the previous request completes. Figure 14a) shows the resulting energy plot. Note that traffic becomes more bursty at timescales $11-14$ (corresponding to the 1-8 sec range) relative to the default Auck trace. To vary burstiness at timescales $3-6$ (4-32 ms), we restrict all links in the emulated topology for Swing sources to 2 Mbps. Figure 14a) also shows the energy plot for this case. Let us call these two new traces $Traffic1$ and $Traffic2$. We will next try to understand if these seemingly small differences in burstiness impacts application performance.

We run HTTP experiments across a shared 10 Mbps link. The results in Figure 14b) shows the effect of varying burstiness on a relatively relatively large 1 MB file transfer. We see that varying burstiness at small time scales (Traffic2) has relatively little impact on the distribution of response times. Burstiness at larger time scales (Traffic1) skews the CDF around the median. Few flows finish early and few take longer compared to Auck/Traffic2. We hypothesize that over a long transfer, burstiness at small time scales (milliseconds) averages out over the lifetime of the connection. Burstiness over multiple seconds however will more significantly impact transfers that complete in just over one second by default.

The situation reverses itself when we consider the distribution of download times for the same experiment but with 4 KB objects as shown in Figure 14c). In this case, increased burstiness at large timescales (Traffic1) has no impact on the distribution of performance relative to the baseline. However, the decreased burstiness at small timescales for Traffic2 relative to Auck results in improved download times, especially above the 80th percentile. Note that Traffic2 displays reduced burstiness in the 4-32 ms timescales (Figure 14a), and that retrieving a 4 KB takes 29 ms in the median case for Auck. From this initial experiment, we hypothesize that *for a given level of background traffic, its burstiness characteristics at timescales corresponding to the duration of individual application operations will have the most significant impact on overall application behavior.*

6 Discussion

While this paper shows the importance of subjecting network services to a range of background traffic conditions, there are a number of remaining open questions. First is the need for a suite of background traffic that capture the range of conditions likely to be experienced during live deployment. We have shown that the bursty traffic present on the Internet can dramatically affect application behavior relative to synthetic traffic models. However, we cannot yet characterize the full range of network conditions likely to be present on the Internet.

Next, we have not yet shown the best way to generate the background traffic. We have shown one plausible technique employing the Swing traffic generator. However, Swing requires multiple machines to generate the traffic and a network emulator to appropriately shape individual flows. And yet, our current understanding indicates that recreating Internet traffic burstiness critically depends on recreating appropriate network characteristics and closed-loop responsive traffic sources and sinks that, for example, obey the dynamics of TCP [21].

Finally, our analysis considers the effects of background traffic on a single link between sources and destinations. Under realistic deployment scenarios, application traffic must interact with background traffic at multiple links across the network. Our initial experiments, not shown for brevity, indicate that the impact of bursty background traffic is even more pronounced and unpredictable when considering more complex network topologies. We believe that capturing this full complexity will be challenging in the short term, but it would be valuable to determine whether relatively simple models can account for most of the additional impact that comes from more complex topologies.

7 Conclusion

We set out to answer a simple question: When running simulation or emulation experiments, what kind of background traffic models should be employed? Additional motivation comes from recent interest in accurately recreating realistic background traffic characteristics. While there have been significant advancements in this space, there is relatively little understanding of what aspects of background traffic actually impact application behavior. To fill this gap, we quantified the interaction of applications with a variety of background traffic models. We found that, for instance, HTTP is sensitive to the burstiness of background traffic depending on the dominant size of transferred objects; multimedia applications have been engineered to be relatively insensitive to traffic burstiness; and bandwidth estimation tools are highly sensitive to bursty traffic because unstable link characteristics make convergence to stable estimates difficult. We also observed that characteristics of background traffic in both directions of a link can impact application performance. Finally, we hypothesize that each application is sensitive to burstiness of traffic at particular application-dependent timescales.

Acknowledgements

We would like to thank the reviewers for their insightful and detailed comments that helped improve both the quality and presentation of this paper. Ryan Braud, Chris Edwards and Marvin McNett helped set up the infrastructure behind our experiments.

References

[1] ABRY, P., AND VEITCH, D. Wavelet Analysis of Long-range-dependent Traffic. *IEEE Transactions on Information Theory* (1998).

[2] ALAN SHIEH AND ANDREW C. MYERS AND EMIN GUN SIRER. Trickles: A Stateless Network Stack for Improved Scalability, Resilience, and Flexibility. In *NSDI* (2005).

[3] ANDERSON, T., COLLINS, A., KRISHNAMURTHY, A., AND ZAHORJAN, J. PCP: Efficient Endpoint Congestion Control. In *NSDI* (2006).

[4] ANNAPUREDDY, S., FREEDMAN, M. J., AND MAZIERES, D. Shark: Scaling File Servers via Cooperative Caching. In *NSDI* (2005).

[5] Auckland-VII Trace Archive, University of Auckland, New Zealand. `http://pma.nlanr.net/Traces/long/auck7.html`.

[6] BARFORD, P., AND CROVELLA, M. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *MMCS* (1998).

[7] CHEN, J., GUPTA, D., VISHWANATH, K. V., SNOEREN, A. C., AND VAHDAT, A. Routing in an Internet-Scale Network Emulator. In *Proceedings of the IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '04)* (October 2004).

[8] FELDMANN, A., GILBERT, A. C., HUANG, P., AND WILLINGER, W. Dynamics of IP Traffic: A Study of the Role of Variability and the Impact of Control. In *ACM SIGCOMM* (1999).

[9] HERNANDEZ-CAMPOS, F., SMITH, F. D., AND JEFFAY, K. Generating Realistic TCP Workloads. In *CMG2004 Conference* (2004).

[10] JAIN, M., AND DOVROLIS, C. Pathload: A Measurement Tool for End-to-end Available Bandwidth. In *PAM* (2002).

[11] MAHADEVAN, P., HUBBLE, C., HUFFAKER, B., KRIOUKOV, D., AND VAHDAT, A. Orbis: Rescaling Degree Correlations to Generate Annotated Internet Topologies. In *ACM SIGCOMM* (August 2007).

[12] MAWI Working Group Traffic Archive. `http://tracer.csl.sony.co.jp/mawi/`.

[13] NAGARAJA, K., OLIVEIRA, F., BIANCHINI, R., MARTIN, R. P., AND NGUYEN, T. D. Understanding and Dealing with Operator Mistakes in Internet Services. In *OSDI* (2004).

[14] NAHUM, E. M., ROSU, M.-C., SESHAN, S., AND ALMEIDA, J. The Effects of Wide-area Conditions on WWW Server Performance. In *SIGMETRICS/Performance* (2001).

[15] OF UTAH, U. Emulab.Net: The Utah Network Testbed. `http://www.emulab.net`.

[16] PAXSON, V., AND FLOYD, S. Wide-Area Traffic: The Failure of Poisson Modeling. In *IEEE/ACM Transactions on Networking* (1995).

[17] RIBEIRO, V., RIEDI, R., BARANIUK, R., NAVRATIL, J., AND COTTRELL, L. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *PAM* (2003).

[18] SHRIRAM, A., MURRAY, M., , HYUN, Y., BROWNLEE, N., BROIDO, A., FOMENKOV, M., AND KC CLAFFY. Comparison of Public End-to-End Bandwidth Estimation Tools on High-Speed Links. In *PAM 2005*.

[19] SOMMERS, J., AND BARFORD, P. Self-Configuring Network Traffic Generation. In *IMC* (2004).

[20] VAHDAT, A., YOCUM, K., WALSH, K., MAHADEVAN, P., KOSTIC, D., CHASE, J., AND BECKER, D. Scalability and Accuracy in a Large-Scale Network Emulator. In *OSDI* (2002).

[21] VISHWANATH, K. V., AND VAHDAT, A. Realistic and Responsive Network Traffic Generation. In *ACM SIGCOMM* (2006).

[22] WILLINGER, W., PAXSON, V., AND TAQQU, M. S. Self-similarity and Heavy Tails: Structural Modeling of Network Traffic. In *A Practical Guide to Heavy Tails: Statistical Techniques and Applications* (1998).