The Impact of Background Network Traffic on Foreground Network Traffic

George Nychis
Information Networking Institute
Carnegie Mellon University
gnychis@cmu.edu

Daniel R. Licata
Computer Science Department
Carnegie Mellon University
drl@cs.cmu.edu

***Abstract*— We study the impact of background network traffic on foreground network traffic. Foreground applications, such as Web browsers, are those that require low latency for their network traffic, often because a user is actively waiting for the network's response. Background applications, such as peer-to-peer file sharing systems, are those that do not require fast responses to individual requests. Current foreground applications often do not, overall, transfer that much data, whereas current background applications are often more data-intensive. Thus, one would hope to provide low latency to foreground applications while simultaneously providing high bandwidth to background applications.**

In this paper, we show empirically that there are circumstances under which a current TCP implementation does not achieve this goal. In particular, we identify situations in which background traffic on a single machine increases the latency of foreground traffic on that same machine. We describe a study of the impact of various sources of background traffic on hosts of varying connection speeds; in our results, background traffic is problematic when and only when it saturates a host's upstream (sending) link.

I. Introduction

Foreground applications are applications such as Web servers, e-mail clients, and real-time automated stock traders that require low-latency network traffic. For example, a Web browser requires low latency because the user actively waits for the result of making an HTTP request; a stock trading application requires low latency because the information it acts upon quickly becomes dated. *Background applications*, such as peer-to-peer file sharing systems, are those that do not require fast responses to individual requests. For example, a user does not notice the per-packet latency of a BitTorrent file download, only the overall transfer time. For current applications, it is often (but not always) the case that foreground applications do not transfer much data overall; on the other hand, it is often the case that background applications are more data-intensive. These circumstances suggest the following goal: provide low latency to foreground applications while simultaneously providing high bandwidth to background applications.

In the current work, we describe a study of the impact of background traffic on foreground traffic. Our study considers the impact of background traffic in a variety of situations. First, we consider both nodes connected to a high-speed university network and nodes connected to a residential cable modem; we call the connection of a node its *environment*. Next, we consider four *scenarios* of background traffic: sending only, receiving only, both sending and receiving, and sending and receiving with multiple partners. The first three scenarios use controlled background traffic generated using the Iperf tool; the fourth considers more real-world background traffic generated using BitTorrent.

Our study identifies circumstances under which the goal of low-latency foreground and high-bandwidth is *not* achieved by a current transfer control program: we show that the response-time of foreground traffic from one machine is, in some circumstances, adversely affected by the presence of background traffic on that machine. In particular, in our results, background traffic is problematic when and only when it saturates a host's upstream (sending) link.

The remainder of this paper presents our experiment and analysis. In Section II, we describe the experiment that we performed. In Section III, we summarize the data that we gathered. In Section IV, we discuss and attempt to explain these results. Finally, in Section V, we discuss related work.

II. Experiment Setup

To study the impact of background traffic, we recorded TCP header information for various scenarios of network usage; we describe the variables between these scenarios here.

A. Variables

Presence of Background Traffic: To answer our primary question, we chose HTTP requests as a representative type of foreground traffic and ran HTTP requests both with and without background traffic.

Connection to the Internet: In addition to our primary variable, we also chose to study how high bandwidth and low bandwidth environments are affected by background traffic. To this end, our experimental setup consisted of two computers in each of two different network environments, Carnegie Mellon University's campus network (at the INI) and Comcast's cable Internet service.

Source of Background Traffic: Finally, we chose to study the effects of four scenarios of background traffic on the HTTP foreground traffic. In three of the scenarios, we used the Iperf tool to generate background traffic; this allowed us

Scenario Name	BG Type	Sending	Receiving
ic	iperf	comcast	ini
ii	iperf	ini	comcast
ib	iperf	both	both
bt	bittorrent	both	both

Fig. 1. Background Traffic Scenarios

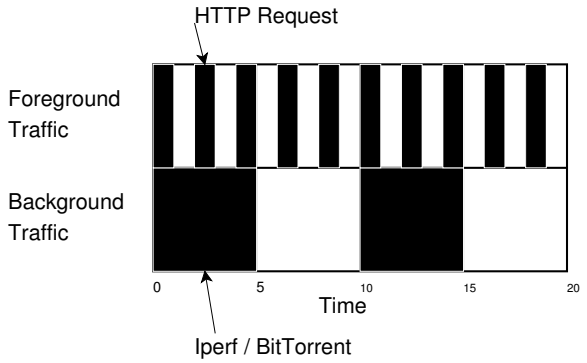

Fig. 2. Interleaving of background traffic

to generate exclusively upstream background traffic, exclusively downstream traffic, and both upstream and downstream traffic. Scenario **ic** is exclusively upstream for Comcast and exclusively downstream for INI; scenario **ii** is the reverse; scenario **ib** included both upstream and downstream traffic for both bandwidth environments. In each of these scenarios, the only background traffic is this transfer between the two environments. In the fourth scenario, **bt**, each host exchanged files with arbitrary other computers using a BitTorrent client; this scenario included both upstream and downstream transfer for each host. Figure 1 summarizes these scenarios.

B. Experiment Details

We ran all experiments for two days to reduce effects of network conditions (for example, Comcast's cable has slightly lower performance during peak hours). During the eight-day period, over 500GB of data was transferred and 80GB of headers were captured for the analysis.

Controlling for Network Conditions: To help ensure that any differences observed were actually due to the presence or lack of background traffic, we attempted to control for changes in the ambient network conditions. To this end, we alternated between five minute intervals with background traffic and five minute intervals without such traffic. We ran HTTP requests on the even minutes of every hour to generate the foreground traffic; we started background traffic every ten minutes but ran it for only five minutes. Figure 2 illustrates this interleaving.

Generating Foreground Traffic: We claim that HTTP requests are typical of foreground traffic: a user usually waits actively for the Web site to load, and Web pages generally do not carry that much data.

We had to choose the Web site targeted by our foreground traffic carefully. First, we required a site that did not have dynamic content to ensure that over the days our data was collected the length of the HTTP flows was constant. Second, it was necessary that the website did not lock us out for requesting pages every two minutes over an eight day period across four different machines. Finally, the website needed to be resilient to flash crowds (or simply never experience any) so as to also not skew our results. For these reasons, we chose `www.microsoft.com`.

Generating Background Traffic: To generate BitTorrent traffic we used several techniques that gave us a level of control, such that we had some minimum amount of upstream and downstream at all times. It was desired to have this minimum amount such that the link was saturated at all times. To do this we created two groups of torrents, *download* and *upload*.

The torrents were placed in to the groups depending on the number of seeders and leechers. Any torrent with 100+ leechers was considered an upload torrent since we could pre-download the content and seed it to over 100 BitTorrent clients to generate upstream traffic. Any torrent with 100+ seeders was considered a download torrent since we could quickly saturate our downstream by starting multiple torrents of this type.

Since the background traffic was run at a small interval of five minutes, we needed to generate the BitTorrent traffic as quickly as possible. Torrents are known to have slow startup times while peers are located. Although we could not directly minimize this latency, if we chose torrents with large numbers of peers and started multiple torrents we could attempt to minimize it indirectly. We had two separate techniques for the *download* and *upload* torrent groups.

The first technique, for the *download* group, was to restart the torrents at the beginning of each background traffic generation interval. By restarting the torrents we ensured that we needed the most popular seeded chunks. Since the most popular seeded chunks are often the very first chunks of a file in BitTorrent, restarting the torrents ensured that with even a small number of seeders we could quickly saturate the download link.

The second technique, for the *upload* group, was to pre-download the entire content of the torrent before running the fourth experiment. By pre-downloading the content we ensured that our BitTorrent clients could seed even the rarest chunks of the torrents. This would give us the highest probability of saturating the upstream quickly and keeping it saturated with a large number of leechers.

Although several BitTorrent clients have application level throttling of its traffic, we ran all clients with unlimited upstream and downstream bandwidth.

III. Presentation of Data

We consider three network performance metrics which can be extracted from the collected headers:

1) The overall round-trip transfer time (RTT) of an HTTP request—i.e., the time between when the initial request

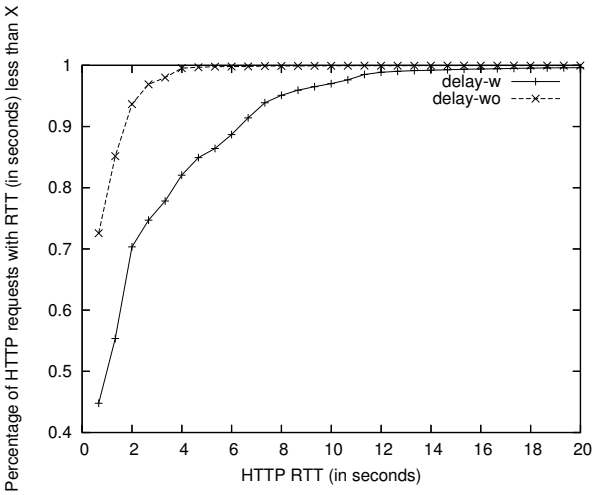

Fig. 3. HTTP RTT from all scenarios

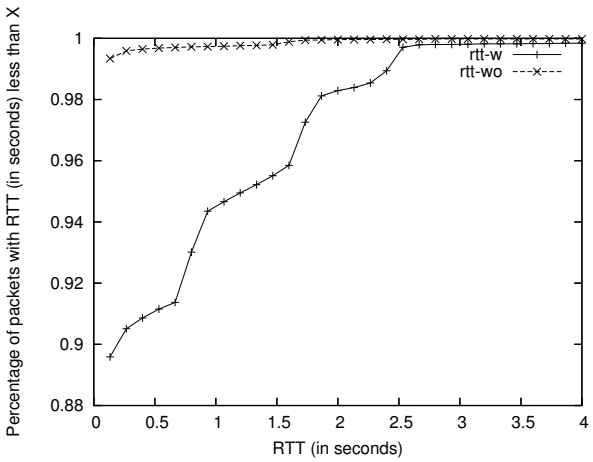

Fig. 4. RTT per packet from all scenarios

is sent and when the last data content from the Web page is received.

2) The individual round-trip transfer time of a packet in an HTTP request.
3) The percentage of HTTP request packets that needed to be retransmitted.

We first discuss the overall differences in these metrics between requests with and without background traffic; next we examine the details of the various environments and background traffic scenarios.

A. Overall Results

Figure 3 presents cumulative histograms, aggregated over all scenarios and environments, for the HTTP-request round-trip times both with and without background traffic. Because having more smaller round-trip times is desirable, the top line is better than the bottom: as one would expect, the graph shows that more requests came back faster without background traffic than with background traffic. For example, nearly all requests had been served within four seconds when there was no background traffic, but the same took close to twenty seconds for requests with background traffic. The graph also shows that only 45% of all requests had been served within one second with background traffic, whereas 72% had been served in one second without the background traffic.

Figure 4 presents the analogous graph for the per-packet round-trip times. This graph shows the same overall trend, but note that the difference between the lines is smaller than for HTTP requests. This suggests that the HTTP RTT difference might be additive in the differences for various individual packets in the stream (rather than, for example, a single packet limiting the RTT time for the entire HTTP request).

Figure 5 presents the percentage of packets that were retransmitted in each experiment. For the present purposes, the important observation is that the number of retransmitted packets always stays under 1.232% in the Comcast environment and under 3.130% in the INI environment. Thus, even though background traffic sometimes increases the percent retransmissions by a significant factor, the overall number of retransmissions stays within, we claim, an acceptable range. Note that the additional retransmissions with background traffic are not very pronounced for INI, and that for Comcast the percentage of retransmissions *with* background traffic is in all cases less than the percentage of retransmissions for INI *without* background traffic.

B. Differences Between Environments and Scenarios

Figure 6 shows the HTTP RTTs by scenario for the hosts in the INI environment. For three of the four scenarios, the two lines for the scenario, which represent traffic with and without background transfer, track each other nearly exactly. The scenario that showed the most difference was BitTorrent, but even here the difference was slight. This leads us to conclude that our background traffic had little effect in the INI campus network environment.

Figure 7 shows the HTTP RTTs by scenario for the hosts in the Comcast environment. Because we observed an overall difference in HTTP RTTs in the previous section, it is expected that we will see differences in at least some of the scenarios presented here. In fact, we see pronounced differences in three of the four scenarios: the **ii** scenario, in which the Comcast nodes were only receiving data, is the only one that does not suffer very much in the presence of background traffic.

Referring back to the percent retransmissions in Figure 5, we consider the retransmissions in the four scenarios. First, observe that background traffic did not cause a significant increase in retransmissions in any scenario in the INI environment. In the Comcast environment, note that no experiment caused both a significant percent difference of retransmissions and a significant difference in the delay of the HTTP request RTT solely—indeed, the scenario with the largest percent increase in retransmissions with background traffic, **ii**, was also the scenario that showed the least difference in overall HTTP RTT.

IV. Discussion of Results

The retransmission data suggest that retransmissions were not the cause of the increased RTT for HTTP requests.

Scenario	Environ.	BG Traffic	% Retrans.
ic	Comcast	w	0.058
		wo	0.050
	INI	w	2.840
		wo	2.747
ii	Comcast	w	0.848
		wo	0.025
	INI	w	2.444
		wo	2.376
ib	Comcast	w	0.071
		wo	0.048
	INI	w	2.786
		wo	2.758
bt	Comcast	w	1.232
		wo	0.141
	INI	w	2.223
		wo	3.130

Fig. 5. Foreground Retransmissions

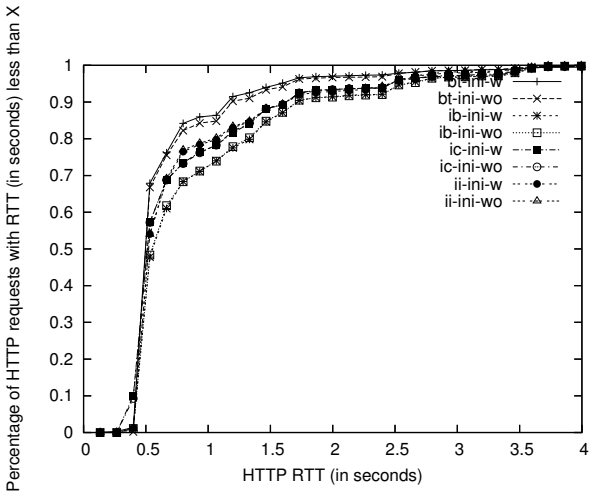

Fig. 6. HTTP RTT by scenario for INI environment

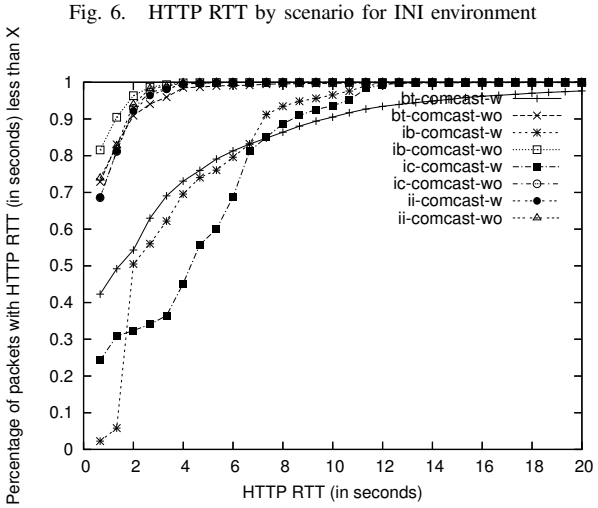

Fig. 7. HTTP RTT by scenario for Comcast environment

What, then, could the cause have been? Our data suggest the following clues: in the INI environment, the BitTorrent scenario is the only one in which much of a difference was observed; the Comcast environment, noticeable differences were observed except when Comcast's upstream was not being used. Thus, the increased HTTP RTT seems correlated with scenarios and environments in which a host's upstream is saturated: because Comcast's downstream bandwidth is much smaller than INI's upstream, BitTorrent is the only scenario in which INI's upstream could have been saturated; because cable modems have a relatively small upstream bandwidth, any scenario in which it is used for background traffic is likely to saturate Comcast's upstream.

This conjecture is supported by various pieces of related work. Balakrishnan et al. [1] study the effects of asymmetric links on TCP's performance, showing degradation in TCP when the reverse link, such as our upstream link, has greater delay or less bandwidth than the forward link. These authors and others suggest that degradation of a downstream link, in the presence of a saturated upstream link, is the effect of *ACK compression* and *ACK loss* on TCP's acknowledgment-based rate control mechanism [1], [7], [3]. The ACK compression problem is as follows: TCP relies on Acknoweldgements to slide its window and transmit more data. When ACKs become bunched together on a receiver's upstream link, the sender will hold off on sending more packets, leaving the link idle for a period of time; consequently, ACK compression results in a lower overall throughput than if the same Acknoweldgements had been received at an even rate. For example, consider the Comcast environment in scenario **ic**, where Comcast's upstream link saturated while making the HTTP requests. Our results in Figures 7 and 5 show a significant increase in HTTP RTTs but an insignificant increase in the percentage of retransmissions. The ACK compression explanation for these results is as follows: The Comcast node's acknowledgments of the receipt of the HTTP request packets are queued in the node's upstream because there was so much Iperf traffic to INI. With acknowledgments being queued and bunched together, the ACK-clocking at the HTTP sender causes the HTTP performance to suffer. Since we did not observe a significant increase in retransmissions, we conjecture that the increase in HTTP RTT was not due to lost acknowledgments.

Though we have not yet quantified these results, we have attempted to confirm these conjectures by examining the average delay between a window of packets from the HTTP sender during scenario **ic**, where the Comcast node's upstream link was saturated. With background traffic, the delay was often between one and two seconds. Without background traffic, the delay was less than a tenth of a second. The large

delay between windows of packets offers some support for ACK compression and ACK clocking being the significant factor.

V. Related Work

There have been several systems which have been designed to reduce the interference of background traffic on foreground traffic. The two most developed works are TCP-LP [4] and TCP Nice [6]. TCP-LP requires that all applications considered background traffic use the TCP-LP protocol. The protocol throttles these connections based on delay measurements computed from one way packet delays using the TCP timestamp option. Using these measurements, TCP-LP rate limits the connections to the sum of the available bandwidth algorithmically.

HSTCP-LP works at merging the work of HighSpeed TCP [2] and TCP-LP to provide the performance benefits of HighSpeed TCP while prioritizing flows with the work of TCP-LP [5].

TCP Nice takes the approach of modifying three components of TCP Vegas to reduce the interference of background traffic while maintaining 100% bandwidth utilization [6]. The first modification is adding a more sensitive congestion detector, the second is monitoring round trip times to multiplicatively reduce windows, and the last of which being the capability of reducing window size below a value of one. The authors show through their evaluation that TCP Nice can achieve 100% bandwidth utilization while avoiding interference.

While many of these works contain some experiments demonstrating that background traffic has ill-effects on foreground traffic, none of these experiments consider the high-bandwidth/low-bandwidth and upstream/downstream scenarios that we do here. These considerations are the contribution of the present work.

VI. Conclusion

The study presented in this paper supports the claim that background network traffic has ill-effects on foreground network traffic when the background traffic saturates a host's upstream, and that such saturation is easier to achieve for Comcast cable service than for a univeristy network. We observed significant impacts on the round-trip time of foreground traffic and less-severe impacts on the retransmissions rate of foreground traffic; this suggests that background transfer slows foreground traffic down but does not stop it.

We envision several opporunities for future work:

- It remains to be seen whether our observations can contribute to any useful solutions to ill-effects of background traffic.
- Our study was relatively small: four nodes, two environments, one type of foreground traffic. An interested reseacher could confirm or refute our results at a larger scale.
- An interested research could study impact of background traffic from one machine on the foreground traffic of other nearby machines (i.e., machines that share a network link); the present work considers only the impact of background traffic on foreground processes on the same machine.
- A tangent suggested by the data: without the background traffic at all, why was the retransmission rate for INI so much higher than that for Comcast?

Acknowledgment

We thank Alexander Giamas for running several of the experiments during his work on this project, Sara Rodites for providing us usage of her Comcast service during all of the experiments, and Hui Zhang and Aditya Ganjam for their help and guidance.

References

[1] H. Balakrishnan, V. Padmanabhan, and R. Katz. The effects of asymmetry on tcp performance. In *ACM Mobile Networks and Applications*, 1998.

[2] S. Floyd. Highspeed tcp for large congestion windows. RFC 3649, Experimental, December 2003.

[3] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan. Performance of two-way tcp traffic over asymmetric access links. In *Proceedings of Interop '97 Engineers' Conference*, 1997.

[4] A. Kuzmanovic and E. W. Knightly. TCP-LP: A distributed algorithm for low priority data transfer. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, April 2003.

[5] A. Kuzmanovic, E. W. Knightly, and R. L. Cottrell. HSTCP-LP: A protocol for low-priority bulk data transfer in high-speed high-rtt networks. Submitted for publication.

[6] A. Venkataramani, R. Kokku, and M. Dahlin. TCP-Nice: A mechanism for background transfers. In *Proceedings of Operating System Design and Implementation*, 2002.

[7] L. Zhang, S. Shenker, and D. Clark. Observations and dynamics of a congestion control algorithm: the effects of two-way traffic. In *SIGCOMM*, 1991.