TCP Vegas Performance with Streaming Media

Sean Boyden Anirban Mahanti Carey Williamson
Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, AB, Canada T2N 1N4
{boyden, mahanti, carey}@cpsc.ucalgary.ca

Abstract

In this paper we study the use of TCP Vegas as a transport protocol for streaming media. We also consider TCP NewReno as a transport protocol for streaming media. We find that: 1) TCP is able to transport streaming media with good performance in a wide variety of scenarios; 2) TCP Vegas performs better than TCP NewReno in many cases; and 3) for viable media streams, both TCP variants need to achieve mean throughputs that are at least 1.5 times the encoding rate of the media objects being carried.

1. Introduction

With the increase in "last-mile" and core capacity in the Internet, the use of streaming media has increased dramatically in recent years. The wide availability of software, including RealNetworks' RealPlayer, Microsoft's Media Player, Apple's QuickTime Player, and Flash has also contributed to the increase.

Streaming media places unique demands on the Internet. Encoded media files consist of a series of temporally-related data fragments, which must be received in-order within the bounds of the temporal relationship. If portions of the media object are not received by the client before the preceding portion is decoded, then the decoding of the media object will be interrupted. While streaming media is time-sensitive, it is loss-tolerant to some extent, depending on the nature of the encoding protocol used. The client decoder can recover lost data through interpolation, within reason, without a noticeable loss in quality.

Streaming media can be delivered across the Internet using either the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP). UDP has been generally regarded as the transport protocol of choice for streaming media due to its simplicity. The flow control, congestion control, and retransmission mechanisms in TCP have been regarded as detrimental to media flows as they can introduce extra delay and rate fluctuation in the delivery of media data. The control mechanisms in TCP, however, are considered necessary for the stability of the Internet [9]. These contrasting demands have resulted in much research into developing application-layer control protocols over UDP [10, 13, 26, 30]. Many application-layer rate control protocols are, in fact, similar in behaviour to TCP [13, 26, 30]. This poses the question: why not simply use TCP for streaming media?

Several different variants of TCP exist. These variants are typically distinguished by the particular congestion control and packet loss recovery mechanisms incorporated into the protocol. Some of the important variants are TCP Tahoe [15], Reno [15], NewReno [11], SACK [7], and Vegas [5]. The most widely deployed of these is TCP NewReno [22].

TCP NewReno has a characteristic "sawtooth" pattern of throughput caused by its congestion avoidance and control mechanisms. This throughput pattern is often cited as one of the problems TCP has for streaming media. The sawtooth is a byproduct of TCP's linear increase in throughput during periods of no packet loss followed by a multiplicative decrease in throughput upon packet loss. This causes unstable throughput for the stream, which is considered undesirable. TCP Vegas, being delay-based, rather than loss-based, does not create this sawtooth pattern.

With TCP being used for streaming on a large scale [17, 28], it is important to investigate whether or not TCP Vegas is a more suitable alternative. Intuitively, this would seem to be the case because Vegas touts steadier throughput.

In this paper we carry out a comprehensive simulation-based study of TCP Vegas with respect to the unique needs of streaming media. Through progressively more complex simulations, we describe the behaviour of TCP Vegas under varying network, application, and load parameters. We also run TCP NewReno through the same simulations to provide an opportunity to compare the performance of both variants when delivering streaming media.

Our results demonstrate that TCP Vegas is indeed a suitable transport-layer protocol for streaming media in many cases. Furthermore, we demonstrate that TCP NewReno also performs well in this role. Both variants perform sim-

ilarly in many cases. Vegas shows advantages when the media stream is subjected to high random loss, and when there is much competing background traffic. We find that to achieve a viable media stream, both variants need to achieve a mean throughput of approximately 1.5 times the media encoding rate in the majority of cases examined. These results show that media streaming via TCP is viable, and, in some cases, using TCP Vegas would be preferable to NewReno.

The rest of this paper is organized as follows: Section 2 presents relevant background information for this work. Section 3 discusses some related work. Section 4 discusses the methodology, parameters, and performance metrics used in the simulations. Section 5 presents and discusses some key results from this study. Conclusions and future research directions are presented in Section 6.

2. Background

2.1. TCP Congestion Control

Congestion control in TCP is achieved using four intertwined, yet distinct, congestion control algorithms. Each algorithm plays an important role in the overall system, while being influenced by the other three at the same time. The first of these is an operating mode called Slow Start (SS) which, as the name suggests, starts TCP off slowly to avoid introducing congestion. The second is known as Congestion Avoidance (CA) mode which, once reaching a certain predefined threshold (*ssthresh*) takes over from SS mode. CA attempts to maintain the highest possible throughput for TCP without causing undue congestion. The final two algorithms are closely linked and often grouped together: Fast Retransmit and Fast Recovery. These algorithms were implemented in TCP as a solution to the long delays sometimes created by TCP timeouts. Fast Retransmit and Fast Recovery allow losses to be detected and dealt with quickly.

2.2. TCP Vegas

Proposed by Brakmo *et al.* [5], TCP Vegas is a radical departure from the congestion control and avoidance mechanisms found in TCP Reno and its descendants NewReno and SACK. The first of the changes allows Vegas to detect losses more quickly than Reno's fast retransmit or coarse-grained timeout, the second allows Vegas to detect congestion without inducing loss, and the third allows Vegas to probe for available network capacity without creating losses. A summary of some of the differences between NewReno and Vegas is given in Figure 1.

The changes in Vegas were motivated by the observation that Reno creates congestion as a means to detect the capacity available in the network path. This seems paradoxical, but the mechanisms that increase the congestion window do

	NewReno	Vegas
SS:	Init: `cwnd = 1` Each successful ACK: `cwnd++` Until loss or `ssthresh` reached	Init: `cwnd = 2` Increase every other RTT Dynamic switch to CA when `actual` throughput drops below `expected`

CA:	Increase `cwnd` by `1/cwnd` for each successful ACK	Actual rate estimated per RTT.
		`Expected = cwnd/baseRTT`
		`Diff = Expected - Actual`
		`If Diff < α; cwnd++`
		`If Diff > β; cwnd--`
		`Else no change`

FR/FR:	Three DUPACKs trigger Set `cwnd = ½ cwnd`	Single DUPACK to trigger with fine grained-timers Set `cwnd = ¾ cwnd`

Figure 1. Features of NewReno and Vegas

so until a loss is induced; loss is used as an indicator of both congestion and available capacity. TCP Vegas attempts to avoid creating congestion by relying on network observations in addition to losses.

TCP Vegas samples the round-trip time (RTT) for every transmitted segment. This estimate is then compared against a baseline RTT (*baseRTT*), which is set to the lowest RTT observed so far during the data transfer. Vegas uses this comparison to infer whether there is increasing or decreasing network congestion based on the notion that most fluctuations in RTT are caused by queuing delays at the routers in the network path.

TCP Vegas modifies Reno's Fast Retransmit/Fast Recovery algorithm by using a fine-grained clock to record a RTT for every segment sent. Vegas also sets a fine-grained timer based on RTT measurements. When a duplicate acknowledgement (DUPACK) is received, the timer is checked to see if it has expired. If so, Vegas retransmits the segment. This method is faster than waiting for three DUPACKs as in Reno's fast retransmit, and can detect losses that would be missed by Reno fast retransmit (e.g., large burst losses) because one DUPACK can trigger a retransmission.

After the retransmission, the first and second non-DUPACKs have their timers checked. If the timer has expired, then Vegas will retransmit these segments as well. This allows Vegas to catch losses that may have occurred before the retransmission. Vegas only allows one congestion window reduction per RTT, not multiple reductions as TCP Reno does in some situations. Vegas decreases its congestion window on retransmission to three-quarters the previous congestion window, while Reno reduces to one-half the previous congestion window.

Vegas, once per RTT, measures the actual throughput by recording the time between a segment transmission and the receipt of its acknowledgement, and the number of bytes transmitted between those two events. It compares the difference between these two values against the expected value given the current congestion window and *baseRTT* mea-

surement ($Expected = \frac{cwnd}{baseRTT}$). The difference between the actual throughput and the expected throughput is calculated, and the difference is divided by the TCP segment size, providing a number representing the difference in terms of segments (*Diff*). Two static parameters are involved at this stage: α and β. These specify the lower and upper bounds on the number of unacknowledged segments Vegas wishes to have in the network, and hence, the number of buffers it is willing to occupy in the core routers.

Vegas adjusts its congestion window according to the following logic:

$$\texttt{cwnd} = \begin{cases} \texttt{cwnd++} & \text{if } Diff < \alpha \\ \texttt{cwnd} & \text{if } \alpha < Diff < \beta \\ \texttt{cwnd--} & \text{if } Diff > \beta \end{cases}$$

As the actual measured throughput drops below the expected throughput, it implies there is increasing congestion. As a result, Vegas will linearly decrease its congestion window in an effort to ease the pressure on the network. Similarly, as the RTT decreases, Vegas perceives less congestion and linearly increases the congestion window. It is known that Vegas can treat competing Reno (thus NewReno) flows unfairly in some situations; setting α equal to β helps alleviate this [14].

The slow-start mechanism of NewReno introduces data into the network slowly to avoid creating congestion. The *ssthresh* threshold is set to avoid overshooting this goal, but without *a priori* knowledge of what a suitable congestion window size is, this state variable may be inappropriate. The Vegas variant of TCP attempts to correct this shortcoming by incorporating a dynamic switch between SS and CA mode. In Vegas SS, the initial congestion window is set to two segments, and the congestion window is only increased *every other* RTT in the same exponential fashion as Reno. The same congestion avoidance mechanisms found in CA mode are also present in SS mode, however, they are slightly modified from those used in CA mode. When the actual throughput falls below the expected throughput by one segment size, it switches Vegas from SS to CA. This mechanism is intended to reduce the likelihood of creating losses in slow start mode by increasing the congestion window too quickly.

3. Related Work

Recent empirical studies indicate streaming using TCP is as prevalent on today's Internet as streaming using UDP [17, 28]. Streaming using TCP has several advantages. First, TCP is, by definition, TCP friendly. Second, TCP can pass through firewalls and Network Address Translation (NAT) with ease. Third, TCP is congestion-aware, and there do exist solutions to smooth out some of the effects of the more undesirable characteristics of TCP on streaming media. In essence, short-term fluctuations can be addressed using rate-switching techniques, many of which are essentially the same as the rate control protocols developed for streaming over UDP. There are pros and cons to using either transport protocol, and there are proponents for both techniques. While there exists a substantial volume of work on using UDP to transmit streaming media, there exists very little work regarding the use of TCP for streaming. The remainder of this section presents an overview of some previous work on using TCP for media streams.

In [29], Wang *et al.* develop discrete-time Markov models for both live and stored video streaming using TCP, and validated these models using *ns-2* and Internet experiments. Using these models they explore the parameter space including loss rate, RTT, and timeout values in TCP, and the impact of video playback rate for both *constrained* (i.e., rate-limited streams) and *unconstrained* (i.e., TCP is free to send as fast as it can). They found that TCP is suitable for streaming when the achievable throughput of TCP is roughly twice the bitrate of the stream. For large RTTs, high loss rates, and high timeout values this requirement increases to more than double the encoding rate. They also investigated the ratio of late packets to total packets for different stream types. They found that the fraction of late packets in constrained media streams increased with increasing length of the media. For unconstrained streams they found that the fraction of late packets decreased with increasing length of media. This suggests that unconstrained streaming over TCP is a better choice than constrained streaming. In this work, unconstrained streaming is used in the majority of simulations based on this finding.

In [2] Bohacek develops a model of TCP using stochastic differential equations and validates the model using *ns-2*. Bohacek also applies this model to derive the probability distribution of TCP's congestion window size and uses it to create a TCP-friendly rate for non-TCP video flows using the MPEG-1 video codec. The author only examines constrained streaming, but mentions unconstrained streaming as a possible direction for future work.

4. Simulation Methodology

4.1. Network Model

The network model used in this work is the dumbbell network. A dumbbell is characterized by two core routers connected by a single bottleneck link. All edge nodes are connected to one of these two routers.

In this work, both the media source and background traffic sources are connected to one of the core routers with a 100 Mbps full-duplex connection with negligible latency. The receivers for each type of traffic are similarly connected

to the other router. The properties of the central link between the two routers are manipulated to create varying network conditions. For example, we can simulate different end-to-end latencies by altering the propagation delay of the bottleneck. Each of the two routers is simulated with a 50 kilobyte Drop-Tail queue. The packet size used is 1 kilobyte, but it is possible to have smaller packets due to transfer sizes that are not integer multiples of 1 kilobyte.

4.2. Application Agent

The network was simulated with the *ns-2* network simulator utilizing a set of application agents created for this work. These agents are built at the application layer in *ns-2*, allowing them to interface with the transport layer, much as applications in the real world would be built.

The application agents are meant to be paired; one associated with a TCP source in *ns-2*, and the other a TCP sink (receiver). The sender calculates how much data should be given to the transport layer at a time and how often it should be given to the transport layer. This information is provided via parameters supplied to the simulator by the simulation script, or by command-line arguments.

Although the agents contain some more advanced functionality, the mode of operation used in this work is that of a simple streaming model. This model simulates the interactions of a sender and receiver as if they were streaming via a Web-based model: the client downloads, via HTTP, a meta-file that is then opened, resulting in the launch of a media player. The media player then contacts the media server using the information in the meta-file and the content is then delivered to the media player. This model is the primary mode of operation of many Web-based streaming media sites. It is suitable for studying the basic properties of a transport protocol in relation to its suitability for streaming.

The basic operation of the agent pair is as follows [4]:

1. **Initialization**: Upon simulation start, the initialization routines in the sending agent calculate the amount of data that is to be fed to the transport layer via a simple calculation: *chunk = xfactor*mediarate*heartbeat*. *xfactor* multiplies the rate at which the application agent gives data to the transport layer.

2. **Buffering Phase**: The sender commences sending data in *chunk* sizes as often as *heartbeat* specifies. The receiver receives this data and stores it in the buffer until a certain predefined buffer size is filled.

3. **Streaming Phase**: Once the receiver buffer is filled, the receiver begins to drain the buffer at the specified media encoding rate (*mediarate*) to simulate the decoding of streaming media. While this is happening, the receiver continues to receive data and buffer it. The buffer usage shrinks and grows based on the rate of the stream. If the stream throughput is less than the media encoding rate, then the buffer usage shrinks. If the throughput is greater, the buffer usage grows. The agent has the ability to use a finite buffer, but in this work large buffers are used such that the receiver does not lose data if data arrives faster than it is consumed.

4.3. Simulation Factors

This study considers 9 factors for the simulations. These can be broadly classified as network-oriented and application-oriented. The network-oriented factors are round-trip propagation delay, bottleneck capacity, protocol parameters, router queue discipline, bottleneck loss percentage, and background traffic. Application-oriented factors that affect streaming media are application buffer size, *xfactor*, and media encoding rate. The default values of the factors used in the simulations (unless otherwise indicated) are shown in Figure 2.

Factor	Default Level
α, β	8
Bottleneck	10 Mbps
Buffer size	5% of media length
Media length	300 sec.
xfactor	8
httpbkgflows	10
ftpbkgflows	4
Round-trip prop. Delay	50 ms
Router buffer	50 kB

Figure 2. Default Levels for Simulations

In this paper, we present results from the variation of round-trip propagation delay, induced random uniform loss, and background traffic scaling. For experiments with other factors, the reader is referred to [4].

The round-trip propagation delay is a factor in the simulations. Controlling the round-trip propagation delay affects the RTT experienced by the media flows. Although it is not the only determinant of RTT, in these simulations the delay introduced by router queues is much smaller than that of the propagation delay. The throughput of TCP NewReno is inversely proportional to RTT. This relationship exists for TCP Vegas as well in some cases [27].

Another factor examined is random packet loss. In most of the simulations, any loss experienced by the media streams or background traffic is caused by congestion at the routers. This random loss is introduced to study the effects when a high degree of statistical multiplexing of flows occurs at the routers. The random loss levels are varied from

0.125% to 8%.

The HTTP background flows are set up with a source node attached at one side of the bottleneck and the sink on the other. The HTTP flows use TCP NewReno as a transport protocol. The request inter-arrival times are exponential with mean 0.5 seconds. The transfer size is drawn from a Pareto distribution with a mean of 50 kB and a shape parameter of 1.2. The median transfer size is 50 kB (50 packets). These values are consistent with the heavy-tailed distribution of HTTP responses that have been observed in measurement studies [1, 19]. This distribution creates a wide variety of transfer sizes, from small to large. Similar models for HTTP traffic have been considered in recent simulation studies [18, 27].

The FTP background flows merely transfer data as fast as possible for the duration of the simulation. They also utilize TCP NewReno as the transport protocol and have a source and sink on opposite sides of the bottleneck.

4.4. Performance Metrics

Five performance metrics are used in this work. They can be classified into application-level metrics and network-level metrics. The application-level metrics include achieved throughput, application buffer fill time, and late packets. The network-level metrics are concerned with the router queues. The first of these is the mean queue length, and the next is a measure of jitter, namely the standard deviation of the queue length calculated using *ns-2*'s queue monitoring capabilities. This method of indirect jitter measurement was selected for its simplicity.

The mean throughput metric is an important measure. We have found, generally, that if the media stream can achieve a mean throughput approximately 1.5 times the encoded rate, then in most cases, the media stream can be considered viable. The achieved throughput of any particular stream in simulation depends on many simulation factors.

The application buffer fill time is the next metric. The buffer fill time is directly related to the achieved throughput at the beginning of the simulation. This gives insight into the operation of both TCP variants in the initial slow start mode. If either variant experiences a loss in the initial slow start, this can greatly increase buffering time.

The percentage of late packets is another important metric. Because late data is unusable, this is an indirect measure of the client perceived playback quality. Generally, the lower the percentage is, the better. What the tolerable level of late packets is depends on the method by which the client and media encoding algorithm can reconstruct, conceal, or otherwise cope with missing data [8, 24]. For example, Boyce *et al.* [3] show that packet loss rates of as little as 3% can affect up to 30% of the frames in MPEG-1 video. This is without any advanced interpolation, concealment, or other methods of dealing with loss. In this work, we consider a late percentage of 10% and above to be unacceptable for media playback.

5. Selected Results

This section presents selected simulation results. Specifically, we examine the performance of TCP Vegas and NewReno in a simple case with no cross-traffic, when round-trip propagation delay is varied, when different router queue disciplines are used, when uniform random loss is induced on the bottleneck link, and when background traffic is scaled.

5.1. Basic Streaming

In Figure 3 we see throughput traces for media streams in a simple simulation scenario. In these cases, the media encoding rate is increased from 1 Mbps to 16 Mbps over a 10 Mbps bottleneck. The streams are *unconstrained* and there is no cross traffic present. We see that both NewReno and Vegas perform similarly in these simple cases.

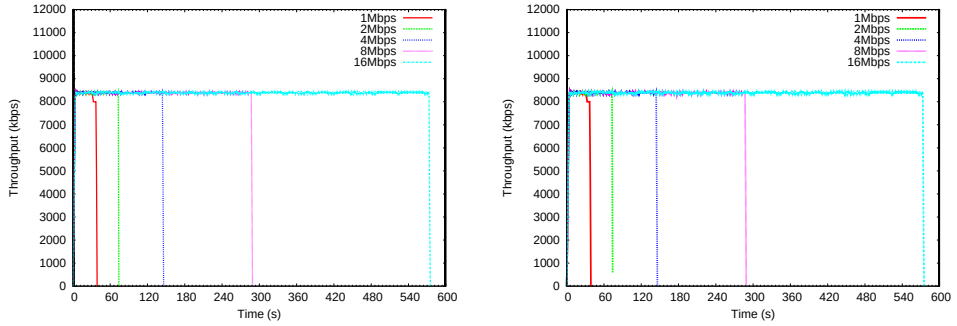
5.2. Latency

We now investigate the impact of the round-trip propagation delay from the sender to the client on the media streaming session. We expect that a higher RTT will be detrimental to media flows.

Stochastic models of congestion window evolution of popular TCP implementations established that the throughput of TCP Reno [6, 16, 20, 21] and its variants such as NewReno [23] is inversely proportional to the product of RTT and square root of packet loss rate. Recently, Samios *et al.* [27] developed a model for TCP Vegas from which throughput can be estimated. Their work shows that Vegas' throughput is not related to RTT when there is no packet loss, but the dependence on RTT increases as loss increases.

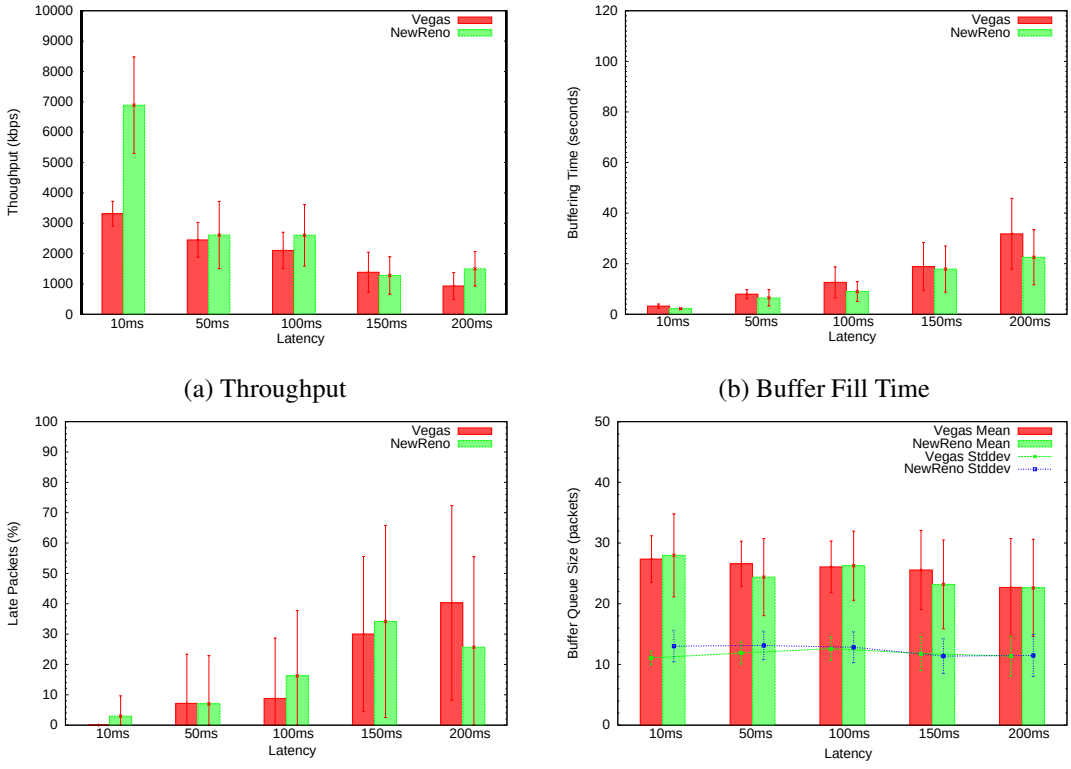
In our experiments Vegas does experience some loss, so there is some relation, but unfortunately there is no simply stated rule regarding the relationship. In the simulations that follow, we vary the round-trip propagation delay amongst some common values found to destinations over the Internet: 10, 50, 100, 150, and 200 ms. Recall RTT is the sum of both the round-trip propagation delay and queuing delays in the network path.

Figure 4 shows the results of the simulation varying round-trip propagation delay for a 1 Mbps media stream. Figure 4(a) displays the mean achieved throughput for both variants. Figure 4(b) shows the buffer fill times for both variants. Figure 4(c) displays the percentage of late packets, and Figure 4(d) shows the mean queue length and jitter measurements. The error bars (in this and subsequent



(a) Streaming with TCP Vegas (b) Streaming with TCP NewReno

Figure 3. Results of Unconstrained Streaming with No Cross Traffic



(a) Throughput (b) Buffer Fill Time

(c) Lates (d) Queue

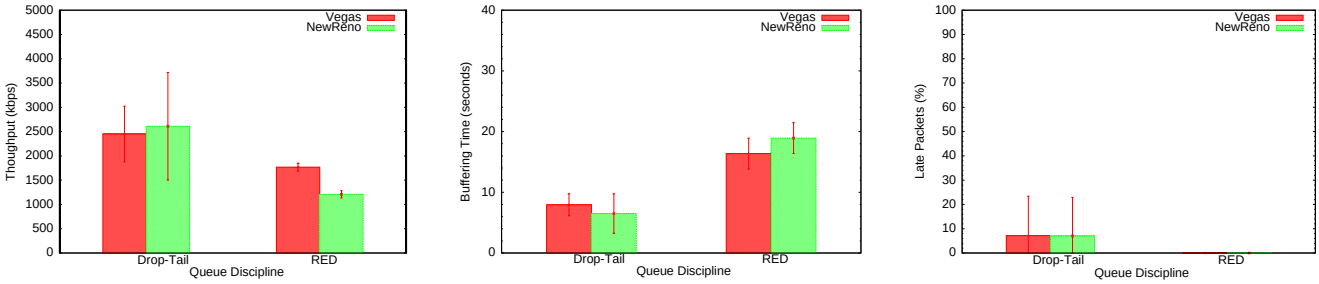
Figure 4. Results of Latency Adjustment for Media Rate 1 Mbps

graphs) represent the 95% confidence interval for 10 simulation runs.

We can see that there is a relationship between Vegas throughput and the propagation delay. It is not strictly inversely-proportional, but is of that nature. Examining the late packet percentage confirms that although the mean throughput is generally greater than the encoding rate of the media stream there is still a significant percentage of late packets, suggesting some loss for TCP Vegas. This supports the notion that there is some relation between RTT and throughput due to packet loss for TCP Vegas.

NewReno also sees a relationship, although it is also not inversely proportional. Doubling the round-trip propagation delay does not reduce NewReno's throughput by a factor of two. This is probably because the loss rates experienced by the NewReno flow with a round-trip propagation delay of 50 ms and 100 ms are different.

With the decrease in mean throughput, the expected increase in buffer fill times is seen for both variants. Late packets also increase for both, from 0% and 4% at a prop-



(a) Throughput (b) Buffer Fill Time (c) Lates

Figure 5. Results of Queue Disciplines for Media Rate 1 Mbps

agation delay of 10 ms to 40% and 26% at 200 ms for Vegas and NewReno, respectively. There is little significant change in mean queue size or jitter for both variants in this series of simulations.

5.3. Router Queue Discipline

The queuing disciplines examined are Drop-Tail and Random Early Detection (RED) [12]. With Drop-Tail queues a network router simply buffers as many packets as it can and any new packets arriving when the buffer is full are simply dropped. RED routers constantly monitor the incoming queue size and drop packets based on statistical probabilities.

Generally RED is considered to be more fair than Drop-Tail in allocating available resources to competing flows because the more traffic any particular flow contributes to the queue, the more drops that flow will experience. (The degree of fairness is often defined as the extent to which each connection on the network path receives an equal share of the available bandwidth.) Previous work with Vegas [25] suggests that RED is a better choice when Vegas flows are present on a network. Their results in simulated single-bottleneck network scenarios show that both the degree and stability of fairness of TCP Vegas using RED routers are significantly higher than using Drop-Tail routers for multiple bulk transfers.

The results for the simulation runs with a 1 Mbps media stream are shown in Figure 5. The throughput plot shows that both protocols achieve a higher mean throughput when the routers use Drop-Tail rather than RED queue management. Vegas achieves a mean throughput of approximately 2.5 Mbps with Drop-Tail and 1.75 Mbps with RED, while NewReno achieves a mean throughput of approximately 2.5 Mbps with Drop-Tail and 1.25 Mbps with RED. Both variants see a marked decrease in throughput with RED queue management. With this decreased throughput both variants see increased buffer fill times. Vegas takes about twice as long and NewReno about 2.5 times as long with RED than with Drop-Tail.

In these simulations, RED eliminates late packets for both variants. Both variants experienced approximately 8% late packets with Drop-Tail. Although RED reduces the average mean throughput and late percentage for both TCP variants in the 1 Mbps case, simulation results in [4] indicate that the gain for NewReno does not continue as the stream encoding rate is increased; for Vegas it does.

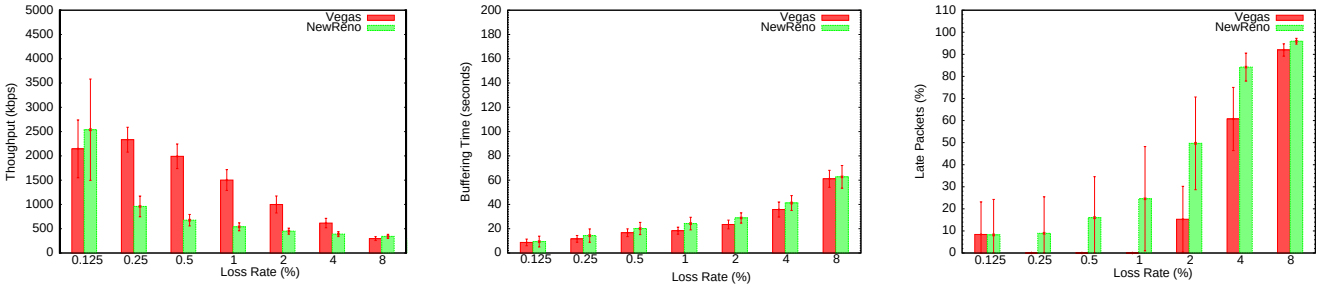
5.4. Random Loss

Most streaming media clients incorporate an application buffer. One of the network effects that can create a need for these buffers is packet loss. Packet loss occurs primarily at the routers when they are congested. Other causes of packet loss include faulty network equipment (e.g., damaged cables), and transmission medium (e.g., wireless networks). Regardless of the cause, both TCP variants tolerate and recover from packet loss. However, the detection and reaction to loss differs between variants and it is this behaviour and its impact on media streams that is interesting.

Packet loss is simulated by randomly dropping packets at the bottleneck link; these simulated packet losses are in addition to any packet losses that occur because of congestion at the bottleneck router. This packet loss thus impacts both the foreground media stream and the background traffic load. The random loss is uniformly distributed and the loss rate is varied from 0.125% to 8% by factors of two.

In Figure 6 the results for the 1 Mbps case are presented. The main observation from the throughput plot is that NewReno is affected much more by random packet loss than Vegas. In fact, NewReno's mean throughput decreases from approximately 2.5 Mbps at a loss rate equal to 0.125% to under 0.5 Mbps at loss rate 8%. In the same interval, Vegas's throughput decreases from approximately 2.1 Mbps to under 0.5 Mbps at loss rate 8%, however, the rate of decrease is significantly less than that of NewReno.

Both NewReno and Vegas have buffer fill times increase as the packet loss rate increases due to decreased throughput. NewReno has slightly higher fill times at each level, but within confidence intervals this is not significant. Tied to the decreased throughput is the result for late packets.



(a) Throughput (b) Buffer Fill Time (c) Lates

Figure 6. Results of Random Loss Adjustment for Media Rate 1 Mbps

There is a low (approximately 8%) mean late packets at 0.125% loss for both variants that continues to increase for NewReno to approximately 95% at 8% packet loss. Given the throughput results for NewReno this is not unexpected. Vegas, however, experiences decreases in its late packet rate in the three subsequent levels of packet loss.

At 0.25%, 0.5% and 1% packet loss, Vegas is able to perform well enough not to experience any late packets. At 2% and above, however, it does experience lates, but until 8% packet loss the rate is significantly lower than NewReno, even if they are too high to produce viable media streams.

With regard to queuing metrics, Vegas and NewReno see a small decrease in mean queue length, as well as jitter as the packet loss rate increases [4].

5.5. Scaling Background Traffic

In this section, we present results from simulations where the background traffic load is varied. Specifically, we systematically vary the number of FTP flows from 1 to 8, and the number of HTTP flows from 3 to 20.

Examining Figure 7 for 1 Mbps streams there is a steep decrease in achieved mean throughput for both TCP variants until the case with 4 FTP and 10 HTTP background flows (4:10). Throughput is reduced by approximately 50% for both Vegas and NewReno from 1 FTP flow and 3 HTTP flows to 4 FTP flows and 10 HTTP flows. From the 4 FTP and 10 HTTP background flows case to the 8 FTP and 20 HTTP background flows case throughput for both variants is further reduced, though not significantly for Vegas ($<$10%), but a further 33% for NewReno.

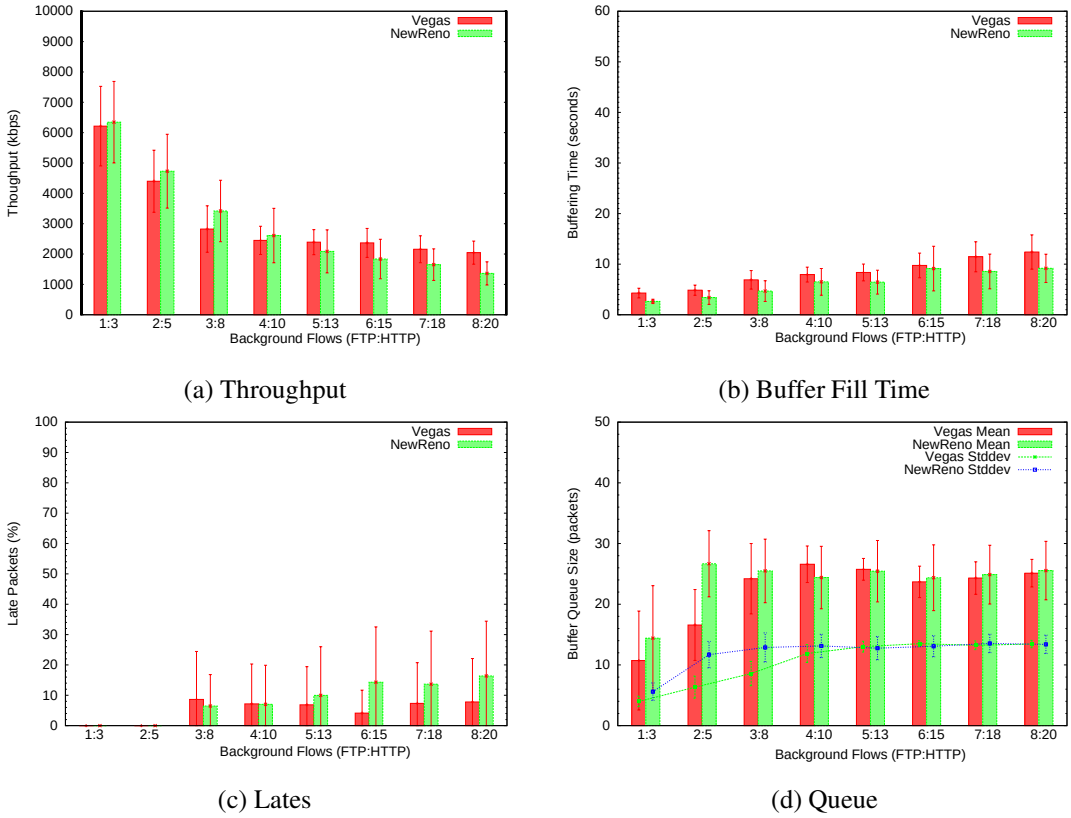
Looking at the late packets metric, we see that both variants have no lates until the 3 FTP and 8 HTTP background flow case. Vegas's proportion of mean lates does not increase past the level (approximately 8%) seen with 3 FTP and 8 HTTP background flows; in some cases the proportion of Vegas lates is lower. NewReno does experience increasing lates from approximately 6% with 3 FTP and 8 HTTP background flows to 15% with 8 FTP and 20 HTTP background flows. Queue sizes for both variants increase significantly from operating with 1 FTP and 3 HTTP background flows to 4 FTP and 10 HTTP background flows, almost doubling, but leveling off after that point. The jitter measure experiences the same trend, increasing as background traffic increases to 4 FTP and 10 HTTP flows and leveling off. The initial increase in jitter is faster for NewReno than Vegas.

The leveling-off effect observed after the case with 4 FTP and 10 HTTP background flows is caused by the level of background traffic. Although the background traffic is increased by 100% between the case with 4 FTP and 10 HTTP background flows and the case with 8 FTP and 20 HTTP background flows, the available bandwidth on the bottleneck is not increased. At the level with 4 FTP and 10 HTTP flows, if one ignores the HTTP traffic, each long-term TCP flow (i.e., the media streams and FTP flows) should receive approximately 2 Mbps of the bottleneck capacity.

Because the RTTs of the background traffic are randomized between [20,440] ms (plus queuing delays), and some of the background TCP flows have higher RTTs than the foreground media streams (50 ms + queuing delays), we see the media flows receiving approximately 2.75 Mbps and decreasing slowly from that point on. Attempting to introduce additional background flows into the already congested network with 4 FTP and 10 HTTP background flows primarily punishes the background flows due to the higher RTTs, although the small decrease in mean throughput with Vegas suggests that it is slightly more resilient to this increased congestion at α=β=8 than NewReno.

6. Summary, Conclusions, and Future Work

This work examined and compared the characteristics and performance of TCP Vegas against those of TCP NewReno. In this paper, we examined selected results including those for performance under varying round-trip propagation delays, varying random loss, and varying background traffic loads. Our work provides the following high-level observations:



(a) Throughput

(b) Buffer Fill Time

(c) Lates

(d) Queue

Figure 7. Results of Background Traffic Scaling for Media Rate 1 Mbps

- In many scenarios, TCP NewReno and TCP Vegas perform similarly. Both protocols perform well in "easy" cases; both perform poorly in "difficult" cases; and both perform better if RTT is low.

- In some scenarios, TCP Vegas shows advantages. TCP Vegas is particularly beneficial under conditions of high background load and under conditions of high packet loss.

In the above summary, "easy" cases refer to simulation scenarios where available bandwidth is plentiful, where RTT is relatively low, and there is little loss. "Difficult" cases include those where the media encoding rate exceeds the capacity of the bottleneck link, or the level of background traffic is very high and the effective capacity of the bottleneck link is low.

From the results presented, we can draw the following high-level conclusions:

- Both TCP NewReno and Vegas are suitable for media streaming under a wide variety of conditions.

- In many scenarios, TCP Vegas is a better choice for media streaming than TCP NewReno.

- To achieve $\leq$ 10% late packets, TCP Vegas must achieve a throughput of approximately 1.5 times the media encoding rate.

This work demonstrates that using TCP for streaming media is feasible under many varying conditions. Empirical studies have shown that this is already the case, so understanding the specific behaviors of TCP variants is valuable. As Vegas and NewReno perform similarly in many cases, choosing to use one or the other may be answered pragmatically: many good implementations of NewReno are available, while those for Vegas are generally experimental. However, having seen that TCP Vegas performs better in some scenarios, and generally performs as well as NewReno, one could conclude that it might be a better choice for streaming media. Because TCP Vegas is a sender-only modification to TCP, it could be easily deployed by a purveyor of streaming media without requiring any action from the user. This has some pragmatic obstacles, such as a lack of a Vegas stack for any operating system other than Linux 2.6, and some known problems with the protocol, including fairness and rerouting issues.

One interesting future work direction would be to augment results presented here and in [4] using experiments over the Internet. Another future direction could involve

comparing the performance of other TCP variants, such as SACK, against Vegas, or a Vegas-based protocol. Developing and testing dynamic α and β scheme for media streams could also be valuable. We have examined long-term Vegas flows primarily in this work; the examination of short-term TCP flows, such as HTTP requests and replies is also important when considering the deployment of TCP Vegas.

Acknowledgments

Financial support for this work was provided by the Natural Sciences and Engineering Research Council (NSERC) of Canada, as well as by the Informatics Circle of Research Excellence (iCORE) in the Province of Alberta. The authors thank the anonymous IEEE IPCCC reviewers for comments that helped improve this paper.

References

[1] M. Arlitt and C. Williamson. Internet Web Servers: Workload Characterization and Performance Implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, October 1997.

[2] S. Bohacek. A Stochastic Model of TCP and Fair Video Transmission. In *Proc. of IEEE INFOCOM*, pages 1134–1144, San Francisco, USA, April 2003.

[3] J. Boyce and J. Gaglianello. Packet Loss Effects on MPEG Video sent over the Public Internet. In *Proc. of ACM Multimedia*, pages 181–190, Bristol, UK, September 1998.

[4] S. Boyden. Streaming Media Performance with TCP Vegas. Master's thesis, University of Calgary, October 2006.

[5] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proc. of ACM SIGCOMM*, pages 24–35, London, UK, August 1994.

[6] C. Casetti and M. Meo. A New Approach to Model the Stationary Behavior of TCP Connections. In *Proc. of IEEE INFOCOM*, pages 367–375, Tel Aviv, Israel, March 2000.

[7] K. Fall and S. Floyd. Simulation-based Comparison of Tahoe, Reno, and SACK TCP. *ACM Computer Communication Review*, 26(3):5–21, July 1996.

[8] N. Feamster and H. Balakrishnan. Packet Loss Recovery for Streaming Video. In *Proc. of International Packet Video Workshop*, Pittsburgh, USA, April 2002.

[9] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Trans. on Networking*, 7(4):458–472, 1999.

[10] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based Congestion Control for Unicast Applications. In *Proc. of ACM SIGCOMM*, pages 43–56, Stockholm, Sweden, August 2000.

[11] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. IETF RFC 2582, 1999.

[12] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[13] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control. IETF RFC 3488, January 2003.

[14] U. Hengartner, J. Bolliger, and T. Gross. TCP Vegas Revisited. In *Proc. of IEEE INFOCOM*, pages 1546–1555, Tel Aviv, Israel, March 2000.

[15] V. Jacobson. Berkeley TCP Evolution from 4.3-Tahoe to 4.3-Reno. In *Proc. of 18th IETF*, Vancouver, Canada, August 1990.

[16] A. Kumar. Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link. *IEEE/ACM Transactions on Networking*, 6(4):485–498, August 1998.

[17] M. Li, M. Claypool, R. Kinicki, and J. Nichols. Characteristics of Streaming Media Stored on the Web. *ACM Transactions on Internet Technology*, 5(5), November 2005.

[18] A. Mahanti, D. Eager, and M. Vernon. Improving Multirate Congestion Control Using a TCP Vegas Throughput Model. *Computer Networks*, 48(2):113–136, June 2005.

[19] A. Mahanti, C. Williamson, and D. Eager. Traffic Analysis of a Web Proxy Caching Hierarchy. *IEEE Network*, 14(3):16–23, May/June 2000.

[20] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM Computer Communication Review*, 27(3):67–82, July 1997.

[21] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proc. of ACM SIGCOMM*, pages 303–314, Vancouver, Canada, September 1998.

[22] J. Padhye and S. Floyd. On Inferring TCP Behavior. In *Proc. of ACM SIGCOMM*, pages 287–298, San Diego, USA, August 2001.

[23] N. Parvez, A. Mahanti, and C. Williamson. TCP NewReno: Slow-but-Steady or Impatient? In *Proceedings of IEEE ICC 2006*, Istanbul, Turkey, June 2006.

[24] C. Perkins, O. Hodson, and V. Hardman. A Survey of Packet Loss Recovery Techniques for Streaming Audio. *IEEE Network*, 12(5):40–48, September/October 1998.

[25] A. Raghavendra and R. Kinicki. A Simulation Performance Study of TCP Vegas and Random Early Detection. In *Proc. of IEEE IPCCC*, pages 169–176, Phoenix, USA, February 1999.

[26] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proc. of IEEE INFOCOM*, pages 1337–1345, New York, USA, March 1999.

[27] C. Samios and M. Vernon. Modeling the Throughput of TCP Vegas. In *Proc. of ACM SIGMETRICS*, pages 71–81, San Diego, USA, 2003.

[28] K. Sripanidkulchai, B. Maggs, and H. Zhang. An Analysis of Live Streaming Workloads on the Internet. In *Proc. of ACM Internet Measurement Conference (IMC)*, pages 41–54, Taormina, Italy, October 2004.

[29] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia Streaming via TCP: An Analytic Performance Study. In *Proc. of ACM Multimedia*, pages 908–915, New York, USA, 2004.

[30] Q. Zhang, W. Zhu, and Y. Zhang. Network-Adaptive Rate Control and Unequal Loss Protection with TCP-Friendly Protocol for Scalable Video Over Internet. *VLSI Signal Processing Systems*, 34(1-2):67–81, 2003.