In Proceedings of RTAS'97: The 1997 IEEE Real-Time Technology and Applications Symposium, Montreal, Canada, June 1997.

EXPLOITING REDUNDANCY FOR TIMELINESS IN TCP BOSTON*

Azer Bestavros best@cs.bu.edu
Gitae Kim kgtjan@cs.bu.edu
Computer Science Department
Boston University
Boston, MA 02215

Abstract

*In a recently completed study, we have unveiled a new transport protocol, TCP Boston, that turns ATM's 53-byte cell-oriented switching architecture into an advantage for TCP/IP. In this paper, we demonstrate the real-time features of TCP Boston that allow communication bandwidth to be traded off for timeliness. We start with an overview of the protocol, and analytically characterize the dynamic redundancy control features of TCP Boston. Next, we present detailed simulation results that show the superiority TCP Boston compared to other adaptations of TCP/IP over ATMs. Namely, we show that it improves TCP/IP's performance over ATMs for both network-centric metrics (*e.g., *throughput and percent of missed deadlines) and real-time application-centric metrics (e.g., response time and jitter).*

1 Introduction

Recently, the premise of high-speed communication using Asynchronous Transfer Mode (ATM) technology has raised the hopes for a better support of real-time (or near real-time) applications. The ATM technology is a connection oriented, 53-byte cell-based transport technology, which offers high-speed switching for both LANs and WANs. ATM is designed to support a variety of applications with diverse requirements using both *bandwidth-reservation* and *best-effort* techniques.

While bandwidth-reservation techniques are able to offer the guarantees necessary for the delivery of real-time streams in many applications (*e.g.* live audio and video), they suffer from many disadvantages that make them inattractive (or impractical) for many others. First, the self-similarity (large and unpredictable variability) of real-time traffic (*e.g.* MPEG video streams [18, 17]) makes bandwidth reservation techniques inherently inefficient in their use of bandwidth, especially in applications that require the concurrent transmission of real-time data (*i.e.* subject to absolute and relative temporal constraints [22]) from/to a large number (potentially thousands) of sources/destinations. Setting up virtual circuits with guaranteed bandwidth for such connections is often impractical, particularly given that the nature of data transmission (*e.g.* rates and sources) may vary wildly. Example applications include real-time network monitoring and visualization, large-scale group simulations, and *etc.* Second, the setting up and tearing down of virtual circuits with guaranteed-bandwidth are expensive operations that may not be justified for short-lived connections. Such connections, involving short file transfers (*e.g.* using HTTP or FTP) are likely to constitute the majority of inter/intranet traffic in the future [13, 14, 1]. Example applications include financial trading, interactive bidding, Internet chat rooms, and *etc.*

These limitations coupled with the flexibility and popularity of TCP/IP as a best-effort transport protocol have prompted the network research community to propose a number of techniques that adapt TCP/IP to the Available Bit Rate (ABR) and Unspecified Bit Rate (UBR) services in ATM network environments. This allows these environments to smoothly integrate (and make use of) currently available TCP-based applications and services without much (if any) modifications [12]. However, recent studies [8, 19, 23] have shown that TCP/IP, when implemented over ATM networks, is susceptible to serious performance limitations. The poor performance of TCP over ATMs is mainly due to *packet fragmentation*, when a relatively large-sized IP packet flows into an ATM virtual circuit through the AAL5 (ATM Adaptation Layer 5)[2]. AAL5 is responsible for the task of dividing TCP/IP's data units (*i.e.*, the TCP/IP packets) into sets of 48-byte data units called *cells*. Since the typical size of a TCP/IP packet is much larger than that of a cell, fragmentation at the AAL is inevitable. In order for a TCP/IP packet to successfully traverse an ATM switching network (or subnetwork), all the cells belonging to that packet must traverse the network *intact*. The loss even of a single cell in any of the network's ATM switches results in the corruption of the entire packet to which that cell belongs, thus resulting in low effective throughput.

There have been a number of attempts to remedy this problem by introducing additional switch-level functionalities to preserve throughput when TCP/IP is employed over ATM. Examples include the Selective Cell Discard (SCD) [3] and the Early Packet Discard (EPD) [23]. In SCD, once

*This work has been partially funded by NSF grant CCR-9706685.

a cell c is dropped at a switch, all subsequent cells from the packet to which c belongs are dropped by the switch. In EPD, a more aggressive policy is used, whereby all cells from the packet to which c belongs are dropped, including those still in the switch buffer (*i.e.* preceding cells that were in the switch buffer at the time it was decided to drop c). Both SCD and EPD require modifications to switch-level software to allow it to be aware of IP packet boundaries—a violation of the layering principle. The simulation results described in [23] show that both SCD and EPD improve the effective throughput of TCP/IP over ATMs. It is important to note that these results were obtained for a network consisting of a single ATM switch. For realistic, multi-hop ATM networks the cumulative wasted bandwidth (as a result of cells discarded through SCD or EPD) may be large, and the impact of the ensuing packet losses on the performance of TCP is likely to be severe.

Our Research: In a recent study [7], we have unveiled a new transport protocol, TCP Boston, that turns ATM's packet fragmentation problem into an advantage for TCP/IP, thus enhancing the performance of TCP in general and its performance in ATM environments in particular. Instead of discarding cells, TCP Boston validates partial packet information, thus preserving network bandwidth. At the core of TCP Boston is the Adaptive Information Dispersal Algorithm (AIDA), an efficient encoding technique that allows for dynamic redundancy control. AIDA makes TCP/IP's performance less sensitive to cell losses, thus ensuring a graceful degradation of TCP/IP's performance when faced with congested resources. The details of TCP Boston, along with analytical and simulation results that show its superior performance, have been presented in [7].

In this paper, we unveil the real-time capabilities of TCP Boston that allow it to trade off network bandwidth for timeliness through the use of AIDA's Forward Error Correction (FEC) features. We start in section 2 with an overview of TCP Boston. In section 3, we present our simulation model and experimental setup, and proceed with the presentation of our simulation results, contrasting the performance of TCP Boston to those of TCP Reno and Vegas, and to those of TCP Reno and Vegas with the EPD switch-level enhancements. These simulations demonstrate the superiority of our protocol—measured using both network-centric metrics (*e.g.*, effective throughput and percentage of missed deadlines) and application-centric metrics (*e.g.*, response time and jitter). We conclude in section 4 with a summary and a discussion of our on-going work.

2 The TCP Boston Protocol

We start this section with an introduction to AIDA. Next, we show how to incorporate AIDA into the TCP/IP stack, and we discuss the various implementation aspects that are incorporated in the current version of TCP Boston (hereinafter interchangeably referred to by "Boston") that is used in our simulations.

2.1 An Introduction to AIDA

AIDA is a novel technique for dynamic bandwidth allocation, which makes use of minimal, controlled redundancy to guarantee timeliness and fault-tolerance up to *any* degree of confidence. AIDA is an elaboration on the Information Dispersal Algorithm of Michael O. Rabin [21].

To understand how IDA works, consider a segment S of a data object to be transmitted. Let S consist of m fragments (hereinafter called *cells*). Using IDA's *dispersal operation*, S could be processed to obtain N distinct pieces in such a way that recombining *any* m of these pieces, $m \leq N$, using IDA's *reconstruction operation*, is sufficient to retrieve S. Both the dispersal and reconstruction operations are simple linear transformations that can be performed in real-time [5].

The IDA approach is different from other *redundancy-injecting* protocols in that redundancy is added *uniformly*; there is simply *no* distinction between data and parity. This feature makes it possible to scale the amount of redundancy used in IDA. Indeed, this is the basis for *Adaptive* IDA (AIDA) [6]. Using AIDA, a *bandwidth allocation* operation is inserted after the dispersal operation but *prior* to transmission. This step allows the system to *scale* the amount of redundancy used in the transmission. In particular, the number of cells to be transmitted, namely n, is allowed to vary from m (*i.e.*, no redundancy) to N (*i.e.*, maximum redundancy).

In order to appreciate the advantages that AIDA brings to TCP Boston, we must understand the main difficulty posed by fragmentation. When a cell is lost *en route*, it becomes impossible for the receiver to reconstruct the packet to which that cell belonged unless: (1) there is enough extra cells (spatial redundancy) from the packet in question to allow for the recovery of the missing information, or (2) the cell is retransmitted (temporal redundancy).

Balancing Spatial and Temporal Redundancies: The incorporation of AIDA in TCP allows us to optimize the use of communication bandwidth by *dynamically* balancing the use of spatial redundancy (through FEC) and temporal redundancy (through retransmission) to achieve the level of timeliness desired by the application software.

Figure 1 shows the transmission window managed by AIDA in TCP Boston. Prior to a packet transmission, AIDA encodes the original m-cell packet into N cells ($N >> m$). Based on network congestion conditions, it dynamically adjusts n the *transmission window size*, which represents the size of the packet to be actually transmitted.

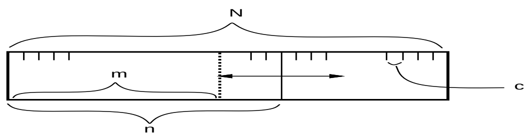


Figure 1. AIDA Transmission Window

The transmission window in TCP Boston can be custom-tuned to meet the spatial redundancy requirements of particular applications or services. For example, time-critical applications may require that the level of spatial redundancy be increased to mask cell erasures (up to a certain level), and thus to avoid retransmission delays should such erasures occur. By avoiding such delays, the likelihood that tight timing constraints will be met is increased (at the expense of wasted bandwidth). In the remainder of this section, we analytically characterize the relationship between spatial redundancy (wasted bandwidth) and temporal redundancy (wasted time). In the next section, we experimentally characterize this relationship.

Spatial Redundancy and Timeliness: The loss of a single ATM cell from a TCP packet implies that the delivery of that packet to the IP layer will be delayed by at least one round-trip (until the retransmitted packet is received). This delay could be minimized if Boston's spatial redundancy feature is used. In this section, we quantify the relationship between timeliness and spatial redundancy—*i.e.,* the relationship between wasted bandwidth and response time.

In our analysis, we assume that the network has a single congested switch, whereby $(1-p)$ is the probability that a cell will be dropped at that switch. We assume a memoryless switch behavior, which means that the dropping of a cell at a switch is an event *independent* of the feat of previous cells transmitted through that switch. To simplify the following derivations, we will assume that a constant number of redundant cells is always added to a packet, independent of the packet size. Let α denote the number of extra cells added by TCP Boston per packet. Thus for every m-cell packet, Boston sends α redundant cells for a total of $m+\alpha$ cells. For example, if $m = 10$ and $\alpha = 1$, then for every 10-cell packet, TCP Boston sends 11 cells, of which *any* 10 cells are sufficient to reconstruct the original packet.

Under these assumptions, packet retransmission would be necessary only when more than α cells are lost *en route*. Thus, the probability of a retransmission is equal to the probability that in a sequence of $m+\alpha$ Bernoulli trials (to send cells through the congested switch), more than α cells are lost. Let W denote that probability.

$$W = \sum_{i=\alpha+1}^{m+\alpha} \binom{m+\alpha}{i} (1-p)^i \cdot p^{m+\alpha-i} \qquad (1)$$

To simplify the following derivations, we assume that the probability of further retransmissions (*i.e.* beyond the first retransmission) is also W. Notice that this is a fairly conservative assumption for TCP Boston since a retransmission will involve much less than the original $m+\alpha$ cells, and the probability of losing α of these fewer number of cells will be *considerably* smaller.

Let U denote the average number of *"round trips"* necessary to deliver a packet. From equation 1, the probability of needing *one* round trip would be $(1-W)$ (*i.e.* succeeding in the first round); the probability of needing *two* round trips would be $W(1-W)$ (*i.e.* failing the first round and succeeding in the second round); ... *etc.* In general, the probability of needing j round trips to deliver a packet would be $W^{j-1}(1-W)$, which leads to the following expression for U.

$$U = (1-W) \cdot \sum_{j=1}^{\infty} j \cdot W^{j-1} = \frac{1}{(1-W)}$$

Substituting from equation 1 and simplifying, we get

$$U = \frac{1}{\sum_{i=0}^{\alpha} \binom{m+\alpha}{i} (1-p)^i p^{m+\alpha-i}} \qquad (2)$$

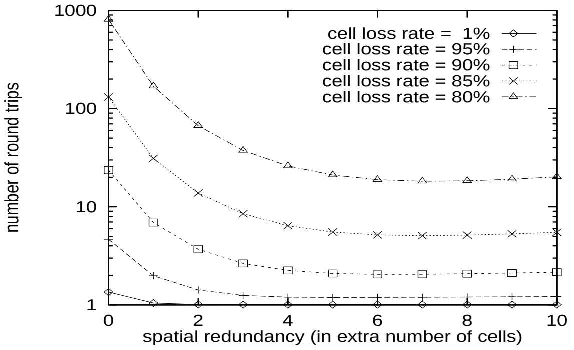
Obviously, U is a measure of response time (in terms of the number of round trips), $(1-p)$ is a measure of switch congestion (*i.e.* p is a measure of transmission reliability), and $S = \frac{\alpha}{m}$ is a measure of spatial redundancy.

The above relationships could be used to set the desired level of spatial redundancy in TCP Boston. For example, if the network Maximum Transfer Unit (MTU) (*i.e.* packet size) is 1.5 kB, thus resulting in a fragmentation at the ATM switch of $m = 30$, then figure 2 shows the relationship between the timeliness (on the Y axis) and spatial redundancy (on the X axis) as predicted using equation 2 for various cell loss rates. Using figure 2, and in order to ensure that on the average packet transmissions will be completed within 3 round trip times in an environment where the worst-case cell loss rates are known to be less than 10%, the appropriate number of extra cells (α) to be used should be 3, making the necessary percentage of bandwidth to trade off for timeliness equal to 10%. If the worst-case cell loss rates are known to be less than 5%, then the appropriate value for α would be 1, making the necessary percentage of bandwidth to trade off for timeliness equal to 3%.

2.2 Overview of TCP Boston

In this section we give a high level description of our implementation of TCP Boston detailed in [7]. The purpose of TCP Boston is to provide a reliable transfer of data for end-to-end applications. It consists of three components: *Session Manager*, *Segment Manager*, and *Flow Manager*.

Session Manager: A TCP session is managed in three phases: a *connection establishment* phase, a *data transfer* phase, and a *termination* phase. The purpose of these phases, as well as the functions performed therein, generally follow those of current TCP implementations, except that information specific to IDA (which is required by the receiver for reconstruction purposes, such as the value of m for example), is piggy-backed onto the protocol packets during the three-way handshaking at the connection establishment phase.



Average number of round trips on the *y*-axis (log scale) as a function of α, the number of cells to be added to a 30-cell (*i.e.*, 1.5-kB) packet, for cell loss rates of 1, 5, 10, 15, and 20 percent.

Figure 2. Response time Prediction

Segment Manager: Three functions that are unique to TCP Boston are: (1) *segment encoding* (at the sender), (2) *segment reconstruction* (at the receiver), and (3) *redundancy control* (at the sender).

Segment encoding: The encoder, before transmitting a data block (segment) of size b bytes, divides the data block into m cells of size c, where $m = b/c$ bytes. Then, the m cells are processed using IDA to yield N cells for some $N >> m$.

Segment reconstruction: Upon a packet receiption, the protocol first checks if it has accumulated m (or more) different cells from the segment that corresponds to that packet. If it did, it reconstructs the original segment using the proper IDA reconstruction matrix transformation. When a partial packet is received, it keeps the received cells in its buffer for later reconstruction.

Redundancy Control: By closely interacting with TCP's Flow Manager, this module estimating the redundancy rate, γ, to manages the transmission window effectively, at the time of each packet (re)transmission. Feedback information such as Round Trip Time (*rtt*) and Success Rate (*SR*) in the acknowledgment (hereinafter referred to as an ACK) can be used as network congestion information to compute γ. Prior to a packet (re)transmission, the module estimates γ ($0 < \gamma < 1.0$), and adds ($\gamma \times original\_packet\_size$) data units (*i.e.*, cells) to the original packet.

Flow manager: Flow and transmission control are the main functions of this module. Any feedback-based TCP flow control algorithm (*e.g.*, Tahoe, Reno, or Vegas) can be used with TCP Boston, with a minor modification to handle the revised feedback mechanism of TCP Boston. To manage its unique feedback scheme effectively, the data structure for ACK needs to be augmented with an integer field, *Pending*, to record the number of pending cells. The Flow Manager's end-to-end feedback and reaction scheme includes a set of provisions for handling three possible cases of packet receptions at the destination: (1) a complete packet has been received, (2) a partial packet has been received (*i.e.* some of the packet's cells were lost), or (3) an entire packet is missing (.e., all cells were lost).

3 Performance Evaluation

In this section we analyze the performance of TCP Boston, based on two different strategies. The first strategy is *bandwidth preserving*, and thus does not employ any redundancy (*i.e. no* FEC). The second strategy trades bandwidth for timeliness by allowing the use of FEC speculatively. We start with a description of our simulation environment. We follow that with an evaluation of the performance of TCP Boston under both bandwidth preserving and FEC environments (using Vegas-style flow control), and compare it with that of TCP Vegas. In particular, we concentrate on real-time performance metrics—namely response time, the percentage of missed deadlines and the variability of packet communication time (as a measure of jitter).

In the remainder of this paper, we use "Boston-BP" (or simply "Boston") to refer to the Bandwidth Preserving TCP Boston protocol, and we use "Boston-FEC" to refer to the TCP Boston with FEC protocol.

3.1 Simulation Environment

We measure the performance characteristics TCP Boston and TCP Reno (or Vegas) under UBR service in ATM networks. Figure 3 illustrates the network topology used in the simulation.

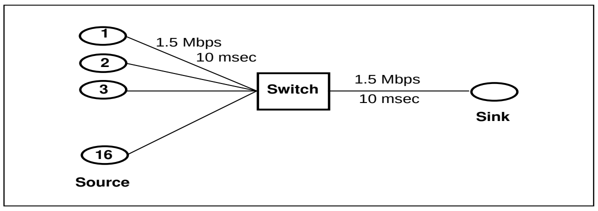


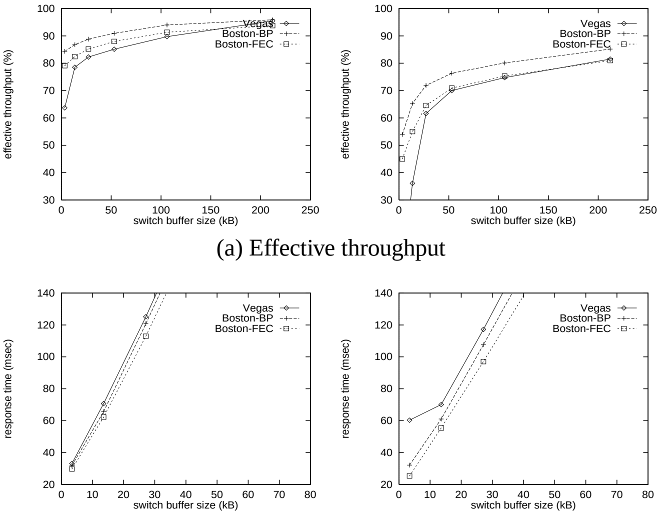
Figure 3. Configuration of simulated network.

The simulated network consists of 16 source nodes and 1 sink node. Each link bandwidth is set to 1.5 Mbps with propagation delay of 10 msec. Our simulations use a total of 16 TCP connections, each is established for one of the configuration's source-sink pairs. Each source generates an infinite stream of data bytes. Each simulation runs for 700 simulated seconds to transfer a total of 120 MB of data.

ATM Switching Model and Interface: The ATM switch is a simple, 16-port output-buffered single-stage switch [11]. The output buffer uses FIFO scheduling, and cells in input ports are served in a round-robin fashion to ensure fairness. In our simulations, the ATM Adaptation Layer (AAL) implements the basic functions found in AAL5, namely fragmentation and reconstruction of IP packets [2, 16]. To support TCP Boston the destination AAL reconstructs a packet out of the received cells even when the resulting packet is incomplete. Incomplete packets are discarded by the destination AAL for Reno and Vegas implementation.

Baseline Parameters: The parameters used in the simulation include the TCP packet size, the TCP window size, and the switch buffer size. Four different packet sizes were selected to reflect maximum transfer unit (MTU) of popular standards: 576 bytes for IP packets, 1,500 bytes for Ethernet, 4,470 bytes for FDDI link standards [20], and 9,180 bytes which is the recommended packet size for IP over ATM [4]. The values for the TCP window size are 8 kB, 16 kB, 32 kB, and 64 kB. Buffer sizes used for the ATM switch are 64, 256, 512, 1,000, 2,000, and 4,000 cells.

Simulation Engine: To simulate TCP Boston, we modified the LBNL Network Simulator (ns) [15] extensively to implement the three main modules described in the previous section, and to support ATM-like network environments. In particular, the essential functions of AAL5 were added to simulate the handling of IP packets. Also, the link layer of ns has been modified to include basic functions of ATM switches and virtual circuit management. To enable us to exploit Boston's FEC features, we have added a TCP Vegas module to complement the existing Reno module of ns.



(a) Effective throughput

(b) Average response time

Experimental setting: 1,500-byte packets (left) and 9,180-byte packets (right) under 64 kB window size

Figure 4. Vegas vs Boston-BP vs Boston-FEC

3.2 Performance Characteristics of TCP Boston

To measure the effectiveness of Boston-FEC's dynamic redundancy features, we used a TCP Vegas module as the base flow control mechanism.

In our experiments for Boston-FEC we fixed the amount of redundancy injected into the data as follows: for normal (initial) data transmissions, a fixed 3.3 percent of redundancy is injected, and for retransmitted data packets, 10 percent of redundancy is added to its original segment size. In addition, segments that belong to current transmission window are multiplexed, so that the cells that belong to a packet are interleaved in the transmission line. This multiplexing is beneficial since cell (or packet) drops are likely to occur in "bundles" [9].

For a baseline performance characterization of Boston-FEC, Boston-BP, and Vegas we rely on two metrics: effective throughput and response time. The results obtained for these metrics are similar to those presented in [7], where the performance of a Reno-based implementation of TCP Boston (as opposed to the Vegas-based implementation presented here) was characterized.

Effective Throughput Characteristics: The effective throughput (or goodput) refers to the throughput where only the bytes that are useful at the application layer are considered. Figure 4(a) gives a comparison of the goodput achieved by: Vegas, Boston-BP, and Boston-FEC, under 64 kB window size for packet size of 1,500 bytes and 9,180 bytes, where the effective throughput of the three different methods is plotted as a function of switch buffer size. Our results show that Boston-BP outperforms both Boston-FEC and Vegas throughout the range of switch buffer sizes, for both small (1,500-byte) and large (9,180-byte) packets. For both packet sizes, the effective throughput of Boston-BP is greater than that of Boston-FEC over the entire range of switch buffer sizes, with the performance gap getting larger as the buffer size gets smaller.

Response Time Characteristics: We define the *packet response time* as the elapsed time from the transmission of the first byte until the receipt of the last byte (including all the necessary dispersal and retrieval processing). Also, we define the *average packet response time* (or simply *response time*) to be average response time for all packets in a single TCP connection.

Figure 4(b) shows the response time of the three methods for the packet size of 1,500 bytes (left) and 9,180 bytes (right) under 64 kB TCP window size, where the average response time of the three different methods is plotted as a function of switch buffer size. Boston-FEC outperforms Boston-BP (not to mention Vegas) in response time characteristics over the entire buffer range. Moreover, the relative gap (*i.e.*, ratio) between Boston-FEC's performance and that of Boston-BP increases as the buffer size shrinks. In the plot, though the gap of response times for the three methods seems to increase as the buffer size increases, the actual ratio of response times—which is a more accurate measure for comparison purposes—increases as the buffer size becomes smaller. For instance, our measurements show that the ratio of response times for the three methods using a small buffer size (3.4 kB) is, Vegas:Boston-BP:Boston-FEC = 1 : 0.96 : 0.90, whereas the ratio using a large buffer size (212 kB) is 1 : 0.97 : 0.96. This means that, when network resources—

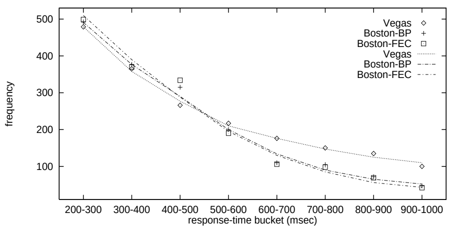
such as switch buffers—become scarce, the cell loss rate at the switch tends to increase.

3.3 Real-time Features of TCP Boston

In this section we show the effect of Boston's FEC on the percentage of missed deadlines and on jitter.

Percentage of Missed Deadlines: When retransmissions occur due to cell losses, a client (*i.e.*, receiver) may require longer wait-time to receive a packet. When network resources become limited, this phenomenon tend to be severe, and as a result, a client may have to wait for multiple RTTs to receive a single packet. For applications that require timely reception of data, packets received after a preset delay become useless. Such a preset delay could be considered as a *firm deadline* on packet communication time.

We measured the percent of packets that miss such a deadline, and compared the results for the three protocols. We fixed the dealine for packet delivery at 700 msec for 1,500-byte packets and at 1,400 msec for 9,180-byte packets.[1] The results of these experiments are shown in figure 6(a), where the percentage of packets that miss their "firm" deadline is plotted on the y-axis as a function of buffer size. Figure 6(b) is an enlarged view of figure 6(a), emphasizing the medium-sized buffer range. For 1,500-byte packets, Boston-BP showed an average of 18.9 percent improvement over Vegas, and an overwhelming average of 41 percent for 9,180-byte packets. Boston-FEC showed an average of 19.2 percent improvement over Vegas for 1,500-byte packets, and a 42 percent for 9,180-byte packets. Boston-FEC showed only a mild improvement over Boston-BP, with an average of about 1.5 percent for 1,500-byte packets and 2.2 percent for 9,180-byte packets.



Experimental setting: 64 kB TCP window, 13.6 kB switch buffer, and 1,500 byte packet size. Response times are grouped in 100-msec intervals.

Figure 5. Histogram of packet response times

Response Time Variability: Figure 5 depicts the distribution (shown as a histogram) of response times observed in a typical simulation. It shows that the distribution of the response times for Vegas exhibits a larger variation than that of Boston-BP, which in turn exhibits a larger variation than that of Boston-FEC. One way of measuring such variability is by computing the standard deviation of the response time distribution. For real-time applications, a small standard deviation is desirable as it would indicate a more "predictable" communication network.

Figure 6(c) shows the standard deviation of the response time as a function of switch buffer size, under 64 kB TCP window size, for packet sizes of 1,500 bytes and 9,180 bytes. In both cases, Boston-BP and Boston-FEC show smaller standard deviations—and thus reduced variability in response time—compared to Vegas. The standard deviation in response time increases for extremely small buffer sizes. This is due to the increased packet loss rates (more so for Vegas than for Boston-BP and Boston-FEC) as shown in figure 4(c).

Figure 6(d) shows the ratio between the standard deviation of Boston-BP and that of Vegas, as well as the ratio between the standard deviation of Boston-FEC and that of Vegas for various switch buffer sizes. In our experiments, Boston-BP showed an average of 32 percent lower standard deviation than that of Vegas under large packet size. Under small packet size, the average standard deviation for Boston-BP showed an improvement of 21 percent over Vegas. For both small and large packet sizes, Boston-FEC showed an additional improvement over Boston-BP. The improvement is more pronounced for small and medium-size buffers, which demonstrates the benefits (in terms of reducing jitter) of using Boston-FEC for a real-time application when network resources are scarce.

Since the response time (and thus the standard deviation of the response time) varies widely for various switch buffer sizes, a more consistent measure of variability—and thus (un)predictability–would be one that normalizes the standard deviation of the response time relative to its mean (using the Z-score theory [10]). Figures 6(e) show the behavior of such a *variability index*. It indicates that Boston-BP's variability index is 25 percent lower than that of Vegas for large packets, and 13 percent lower for small packets.

4 Summary

In this paper we unveiled the real-time characteristics of TCP Boston both analytically and via simulation. Our findings confirm the ability of TCP Boston in reducing response time through a minute increase in bandwidth requirements. This is valuable for real-time applications, especially when the increase in bandwidth requirements can be tolerated.

References

[1] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing Reference Locality in the WWW. In *Proceedings of IEEE PDIS'96: The International Conference in Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1996.

[1] The two deadline values (*i.e.*, 700 and 1,400 msec) have been selected to acquire reasonable rates of deadline-missing packets throughout the entire buffer ranges for both packets sizes presented in the plot.

[2] ANSI. AAL5 – A New High Speed Data Transfer AAL. In *ANSI T1S1.5 91-449*. November 1991.

[3] G. Armitage and K. Adams. Packet Reassembly During Cell Loss. *IEEE Network Mag.*, 7(5):26–34, September 1993.

[4] R. Atkinson. Default IP MTU for use over ATM AAL5. In *RFC 1626*. May 1994.

[5] A. Bestavros. SETH: A VLSI chip for the real-time information dispersal and retrieval for security and fault-tolerance. In *Proceedings of ICPP'90, The 1990 International Conference on Parallel Processing*, Chicago, Illinois, August 1990.

[6] A. Bestavros. An adaptive information dispersal algorithm for time-critical reliable communication. In I. Frisch, M. Malek, and S. Panwar, editors, *Network Management and Control, Volume II*. Plenum Publishing Corporation, New York, New York, 1994.

[7] A. Bestavros and G. Kim. TCP Boston: A Fragmentation-tolerant TCP Protocol for ATM Networks. In *Proceedings of Infocom'97: The IEEE International Conference on Computer Communication*, Kobe, Japan, April 1997.

[8] A. Bianco. Performance of the TCP Protocol over ATM Networks. In *Proceedingds of the 3rd International Conference on Computer Communications and Networks*, pages 170–177, San Francisco, CA, September 1994.

[9] E. Biersack. Performance Evaluation of Forward Error Correction in ATM Networks. *Comm. of ACM*, pages 248–257, August 1992.

[10] C. H. Brase and C. P. Brase. *Understandable Statistics*. D.C.Heath and Company, 1993.

[11] T. Chen and S. Liu. *ATM Switching System*. Artech House, Inc., 685 Canton St., Norwood, Ma 02062, 1995.

[12] D. E. Comer. *Internetworking with TCP/IP*, volume 1. Prentice Hall Inc., Englewood Cliffs, NJ, 1995.

[13] T. M. Corporation. Internet Resource Discovery Service Packets on NSFNET by packets. Available from `http://www.mids.com`, December 1994.

[14] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW Client-based Traces. Technical Report BUCS-TR-95-010, Boston University, Computer Science Department, April 1995.

[15] S. Floyd. Simulator Tests. Available from `ftp://ftp.ee.lbl.gov/papers/simtests.ps.Z`. ns is from `http://www-nrg.ee.lbl.gov/nrg.`, July 1995.

[16] A. Forum. *ATM User-Network Interface Specification*. Prentice Hall, Inc, Englewood Cliffs, New Jersey 07632, 1993.

[17] D. L. Gall. A video compression standard for multimedia applications. *Comm. of ACM*, 34(4):46–58, April 1991.

[18] M. Garrett and W. Willinger. Analysis, modeling and generation of self-similar VBR video traffic. In *Proceedings of ACM SIGCOMM'94*, London, UK, August 1994.

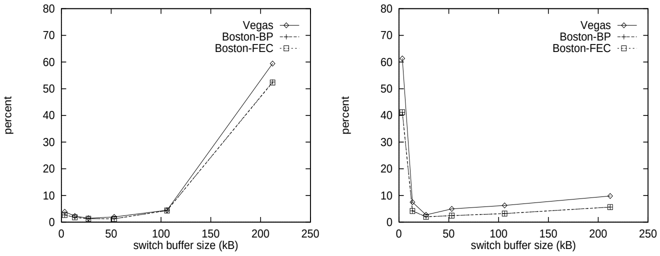
[19] M. Hassan. Impact of Cell Loss on the Efficiency of TCP/IP over ATM. In *Proceedings of the 3rd International Conference on Computer Communications and Networks*, pages 165–169, San Francisco, CA, September 1994.

[20] S. Mirchandani and R. Khanna, editors. *FDDI Technology and Applications*. John Wiley & Sons, Inc., 1993.

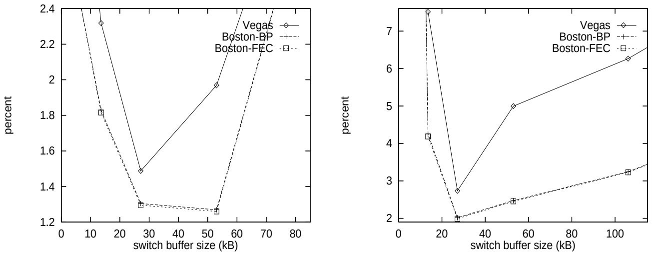
[21] M. O. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the Association for Computing Machinery*, 36(2):335–348, April 1989.

[22] K. Ramamritham. Real-time databases. *International journal of Distributed and Parallel Databases*, 1(2), 1993.

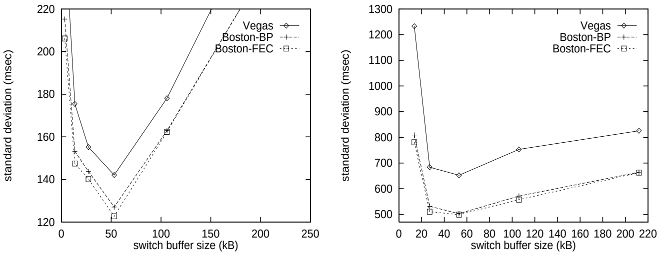
[23] A. Romanow and S. Floyd. Dynamics of TCP Traffic over ATM Networks. *IEEE Journal on Selected Areas in Communication*, 13(4):633–641, May 1995.



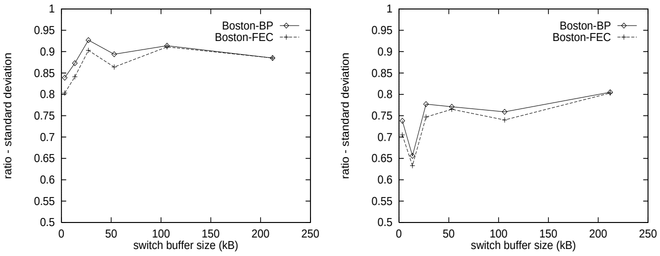
(a) Percent of missed deadlines



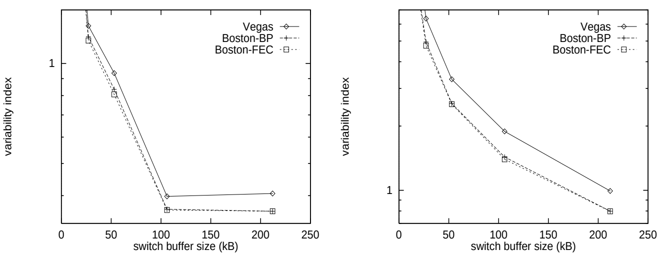
(b) Percent of missed deadlines: Partial view



(c) Standard Deviation of response time



(d) Ratio of Standard Deviation for response time



(e) Variability index of response time on a log scale

Experimental setting: 1,500-byte packets (left) and 9,180-byte packets (right) under 64 kB window size

Figure 6. Vegas vs Boston-BP vs Boston-FEC