A Case for Context-Aware TCP/IP

Carey Williamson Qian Wu
Department of Computer Science
University of Calgary

Abstract—

This paper discusses the design and evaluation of CATNIP, a Context-Aware Transport/Network Internet Protocol for the Web. This integrated protocol uses application-layer knowledge (i.e., Web document size) to provide explicit context information to the TCP and IP protocols. While this approach violates the traditional layered Internet protocol architecture, it enables informed decision-making, both at network endpoints and at network routers, regarding flow control, congestion control, and packet discard decisions.

We evaluate the performance of the context-aware TCP/IP approach first using ns-2 network simulation, and then using WAN emulation to test a prototype implementation of CATNIP in the Linux kernel of an Apache Web server. The advantages of the CATNIP approach are particularly evident in a congested Internet with 1-10% packet loss. Simulation results indicate a 10-20% reduction in TCP packet loss using simple endpoint control mechanisms, with no adverse impact on Web page retrieval times. More importantly, using CATNIP context information at IP routers can reduce mean Web page retrieval times by 20-80%, and the standard deviation by 60-90%. The CATNIP algorithm can also interoperate with Random Early Detection (RED) for active queue management.

Keywords: Internet protocols, TCP/IP, Web performance, network simulation, network emulation

I. Introduction

There has been a long-standing debate in the Internet research community about "layered" protocol stacks, such as the OSI and TCP/IP protocol stacks. The common mantra is: "Layered design is good; layered implementation is bad".

Layered designs are "good" because they provide a unifying framework for the specification and discussion of protocols. Layered designs can lead to modular protocols, each with clearly-defined functionality and well-defined interfaces to other protocol layers. The layered design approach provides as much independence as possible between layers, enabling "plug and play" interoperability with different protocols, if desired.

Layered implementations are "bad" because they can compromise performance. For example, a naive implementation of a strictly layered protocol stack can result in excessive copying of packet data from layer to layer, adding significant overhead to protocol processing. More importantly, strictly layered implementations can hide from higher-layer protocols information that is important to know (e.g., channel error rate, maximum packet size) for optimizing communication. For this reason, implementations of the TCP/IP protocol stack often include inter-layer optimizations. For example, the TCP Maximum Segment Size (MSS) is computed from the Maximum Transmission Unit (MTU) size at the network layer. As another example, TCP uses a PUSH bit in the packet header to expedite delivery of application-layer data in certain situations.

In this paper, we consider the specific problem of Web data transfer using the TCP/IP protocol stack, and show how end-to-end performance can deteriorate on a congested Internet. We then show that end-to-end performance can be improved by providing a minimal amount of context information from the application layer to the TCP/IP protocol stack about the Web document transfer. We use this example to support the case for a general "context-aware" TCP/IP.

We choose the Web/TCP/IP example because Web data transfer is a particularly relevant example from today's Internet, and one in which the mismatches between protocols (HTTP and TCP) are particularly acute. Further discussion on these protocol interaction problems are deferred to Section II.

The question that we focus on in this paper is: can we make the TCP/IP protocols "smarter" about the specific job (e.g., Web document transfer) that they are trying to do? As a possible answer, we propose a "goal-oriented" transport-layer protocol called the Context-Aware Transport/Network Internet Protocol (CATNIP). CATNIP is based on the TCP protocol, but changes the sender's TCP/IP protocol stack to get information, such as Web document size, from the application layer. With this information, CATNIP sources can make informed decisions about flow and congestion control, so as to reduce traffic burstiness, avoid packet losses, and improve Web document transfer times. Similarly, CATNIP routers can make informed packet discard decisions when Internet congestion occurs.

The specific research questions addressed by this work are:

- To what extent can a TCP source influence Web document transfer time, using only application-layer context information, and no additional support from the IP network layer?
- To what extent can an IP router affect Web document transfer time with its packet discard decision-making, using Web/TCP context information?
- What are the performance advantages, if any, of a Context-Aware Transport/Network Internet Protocol (CATNIP) for Web document transfer?

The CATNIP approach is evaluated using both network simulation and a prototype implementation in the Linux kernel. Our simulation results demonstrate significant advantages for the context-aware TCP/IP approach: endpoint control algorithms at the TCP sources can reduce overall packet loss by 10-20%, and exploiting CATNIP context information at routers can reduce mean transfer times by 17-82% in our experiments. A prototype implementation of CATNIP in the Linux kernel demonstrates the feasibility of the approach, and preliminary network emulation experiments with an Apache Web server indicate that up to 40% reductions in Web page transfer times are possible. The performance advantages of CATNIP are most pronounced on a congested Internet with 1-10% overall packet loss. In general, the CATNIP approach reduces both the mean and variance of Web page retrieval times.

The remainder of this paper is organized as follows. Sec-

tion II provides some background information on TCP and Web performance, and a brief discussion of related work. Section III discusses the design of CATNIP, a Context-Aware Transport/Network Internet Protocol for Web document transfer. Section IV presents the experimental methodology for the simulation evaluation of the CATNIP protocol. Section V presents the results from our simulation experiments. Finally, Section VI concludes the paper, and describes ongoing work.

II. Background and Related Work

A. Background

The Web relies primarily on three communication protocols: IP, TCP, and HTTP. The Internet Protocol (IP) is a connectionless network-layer protocol that provides global addressing and routing on the Internet. The Transmission Control Protocol (TCP) is a connection-oriented transport-layer protocol that provides end-to-end data delivery across the Internet. Among its many[1] functions, TCP has flow control, congestion control, and error recovery mechanisms to provide reliable data transmission between sources and destinations. The robustness of TCP allows it to operate in many network environments. Finally, the Hyper-Text Transfer Protocol (HTTP) is a request-response application-layer protocol layered on top of TCP. HTTP is used to transfer Web documents between Web servers and Web clients (browsers). Currently, HTTP/1.0 and HTTP/1.1 [21] are widely used on the Internet.

Several interesting protocol interactions occur when TCP is used to transfer Web documents. For example, TCP's flow and congestion control algorithms are (arguably) designed to optimize throughput for long-lived bulk data transfers. The TCP *slow-start* phase is used initially to probe the network capacity, and steady-state is typically reached in the *congestion avoidance* phase. For small documents, however, the steady-state congestion avoidance phase is not always reached before the transfer terminates. Unfortunately, the extra round-trip times incurred during TCP slow-start add to the transfer times for Web users.

The TCP protocol is unaware of this dichotomy between throughput and latency. In addition, the short and bursty nature of most Web document transfers means that the cost of packet loss and retransmission is high, in terms of transfer latency. Often a TCP timeout is required to recover from a packet loss, adding significant delay to the document transfer time.

B. Performance Problems

Two TCP-related network performance problems are the primary focus in this paper:

- The window-based flow control mechanism often produces bursts (clumps) of packet transmissions. The bursty packet arrival process can make IP routers more susceptible to packet losses. Depending on the number of lost packets and the loss pattern, the performance of a TCP connection can degrade significantly. For example, Barford and Crovella [7] report that file transfers with at least one packet loss take 1.3 to 7.8 times longer than transfers that do not experience a packet loss. Clearly, reducing packet loss is desirable.

[1]The reader unfamiliar with TCP is encouraged to consult Appendix A for a concise TCP tutorial.

- Packet losses can be relatively costly for small document transfers. TCP has two schemes to recover from packet losses: timeout and fast retransmit. The fast retransmit strategy is triggered when a TCP sender receives three duplicate ACKs, and it is more efficient than the timeout strategy. However, for small documents (common in the Web [4], [14], [27]), the congestion window sizes for these TCP connections may be too small to trigger fast retransmit if a packet loss occurs. Thus the less efficient timeout strategy for retransmission is often required. This can result in poor performance for Web data transfers using TCP, in terms of user-perceived response time.

C. Related Work

Several researchers have addressed performance issues related to TCP and the Web.

Padmanabhan and Katz [23] proposed a "fast start" mechanism for TCP that increases the initial TCP window size (based on cached connection state information [6]), combined with a "low" packet priority for these "extra" packets at TCP connection startup. The goal of this approach is to decrease the number of round trip times incurred, particularly for short-lived Web document transfers. The fast start mechanism requires Internet routers to consider packet priority, discarding the extra fast start packets in the event of Internet congestion.

Several other researchers have proposed rate-based pacing of packets as a way to reduce the burstiness of TCP transmissions [1], [19], [29]. Reducing the burstiness can reduce the average level of packet loss in the Internet, and can also reduce the likelihood of multiple lost packets from the same TCP congestion window, which can degrade TCP performance significantly. Improved mechanisms for TCP connection startup and recovery from multiple TCP packet losses have also been proposed [2], [3], [5], [9], [13], [17], [30]. We believe that our context-aware TCP/IP approach complements many of these prior approaches, including connection state caching, rate-based pacing, SACK TCP [13], Ensemble-TCP [12], and RED [16].

Barford and Crovella [8] discuss the "critical path" analysis of TCP connections, in an attempt to pinpoint the causes of the delays incurred during Web document transfers, However, this analysis is primarily from a network traffic analysis viewpoint, rather than a TCP control algorithm viewpoint. Our work builds upon their work in this regard.

Finally, the issue of "context-awareness" for TCP has been considered, but primarily for TCP in relationship to non-traditional physical layer network environments, such as wireless networks and *ad hoc* networks. Conventional TCP implementations consider packet loss as an implicit signal of network congestion, and use backoff mechanisms to reduce packet load offered to the network. This approach works reasonably well for the wired Internet, since losses of packets due to congestion dominate packet losses due to transmission errors. In wireless networks, this situation is reversed: losses due to transmission errors dominate congestion losses. Researchers have added more aggressive mechanisms to TCP to improve its performance in wireless environments. In multi-hop *ad hoc* networking environments, explicit feedback (EF) mechanisms have been proposed to notify TCP of transient routing failures, and the use of rate-based flow control suggested to expedite TCP data transfer.

Context-aware TCP/IP is a general concept, which can be applied to many different application-layer protocols, or indeed to multiple layers of the protocol stack (higher or lower). In this regard, our work is reminiscent of Application-Layer Framing (ALF) [11]. However, the focus in this paper is primarily on the Web as an illustrative example for the use of context-aware TCP/IP. There are two reasons for this choice of example. First, Web traffic constitutes a significant fraction of the packet load on today's Internet [27]. Second, the Web application is one that is particularly sensitive to the dynamics of TCP.

III. CATNIP: A Context-Aware Transport/Network Internet Protocol

This section describes the design of our context-aware TCP/IP protocol. The section begins with some background motivation for the design features of CATNIP, and then a discussion of algorithmic features at both the TCP and IP layers.

A. Motivation

Our work is motivated by the following three observations:

- *Not all packet losses are created equal.*

TCP has two mechanisms to recover from lost data packets, namely timeout and fast retransmit. Surprisingly, most network performance studies in the literature focus on the average TCP packet loss rate, and not on *which* TCP packets are lost. In a given TCP data transfer, the losses of some packets (e.g., the first or last packet of a transfer) can only be recovered using a coarse timeout. Other lost packets may be recovered by timeout, by fast retransmit, or by SACK, depending on the number of packet losses, their relative location in the flow control window, and the congestion window size at the time of the loss.

Our observation is that *which* packet is lost matters a lot, in terms of the impact on the data transfer time for the TCP source. This observation is illustrated graphically in Figure 1, for a (simulated) transfer of an (arbitrarily chosen) 14-kilobyte document in the ns-2 simulator [28]. This experiment uses a simple two-node client-server network model, with a 1 Mbps link, a 100 millisecond propagation delay, and a TCP packet size of 512 bytes. On this graph, the black dots show the transfer time (on the vertical axis) that would result if a *single* TCP data packet loss occurred during the transfer, for the packet with the specific sequence number indicated along the horizontal axis (from 1 to 28). Loss of an ACK packet is less of a problem for TCP, since subsequent ACKs are cumulative. The horizontal dashed line shows the transfer time (1.1 seconds) that would result if no packet losses occurred.

Clearly, the loss of a single TCP data packet during the transfer can have a dramatic impact on the transfer time. The loss of packet 1, 2, or 3 results in a TCP timeout at the source, with the timeout value dependent upon the round-trip time (RTT) estimate available at the source at the time of the loss. Loss of the first[2] packet (packet 1) is the most "expensive", since the default TCP timeout value (6 seconds in the ns-2 simulator) is used. Similarly, loss of any of the last three packets of the transfer (packet 26, 27, or 28) results in a timeout, since there are not enough duplicate ACKs to trigger fast retransmit. Fortunately, there is a tighter retransmit timeout (RTO) value at this point (because of a better estimate of the RTT and its variance), and so the impact on the total transfer time is less pronounced than for the early packets in the transfer. Losses of any of the intermediate packets of the transfer would result in fast retransmit to recover the missing packet (assuming normal TCP slow-start *cwnd* evolution), with minimal adverse impact on the total transfer time, as shown in Figure 1.

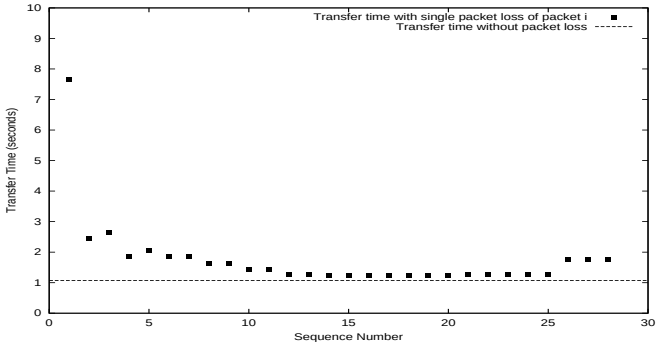


Fig. 1. Transfer Times for a Simulated 14-kilobyte Web Document Transfer under Different TCP Single Packet Loss Scenarios

To summarize, the "packet loss profile" in Figure 1 shows that the transfer time for a Web document using TCP is very sensitive to the loss of a packet early or late in the transfer. In other words, some packets are more important than others in terms of their possible impact on TCP data transfer time. A TCP source that can reduce the risk of packet loss in these phases of the transfer, or at least shift packet losses away from "important" packets, can improve Web document transfer performance.

- *The TCP source has limited control over packet loss effects.*

The TCP slow-start and congestion avoidance algorithms provide a means for a TCP source to estimate and adjust to the bottleneck network bandwidth along a network path. By its very nature, the TCP control algorithm "pushes" until a packet loss occurs, and then adjusts to achieve a steady-state throughput that approaches the available bandwidth.

Unfortunately, the packet loss indicating network congestion can have a dramatically different impact on each TCP source affected by the loss epoch, depending on each source's congestion window size at the time of the loss. While the TCP algorithm tries to control the number of packet losses, there is no consideration of which packets to discard, since an IP router has no state information about a TCP flow (e.g., number of packets, current packet number, current congestion window size).

Adding a means to convey application-layer context information (e.g., Web document size) to the TCP and IP layers may provide sufficient information for informed decision-making at the TCP and IP protocol layers. This context information could take the form of a simple "priority" bit in TCP/IP packet headers to identify crucial packets, or explicit sequence number information (e.g., packet i of N) in every packet.

- *An IP router has significant control over packet loss effects.*

A router that discards packets that are early or late in a specific Web document transfer will have maximal adverse impact on the Web document transfer latency. A router that avoids dis-

[2]Our simulation did not use ns-2's three-way handshake for TCP connection setup, so the first packet referred to here is a data packet, not a SYN packet. A similar observation applies for SYN packets for real TCP connections.

carding these packets will have minimal adverse impact on the Web document transfer latency.

B. Adding Context-Awareness to TCP

The performance issues identified previously motivate an enhancement to TCP that provides explicit context information about the data being transferred. Our experiments consider several different "context-aware" algorithms that could be used at the sender-side for a TCP source, beyond a baseline Reno TCP configuration:

- **Rate-Based Pacing of the Last Window (RBPLW)**: A TCP source that is explicitly aware of the number of packets remaining in its last window of data may choose to space (in time) the transmissions of these last few packets over a short interval so as to reduce the risk of packet loss. This use of Rate-Based Pacing (RBP) is different from that in the literature [1], [19], [29], where RBP has been considered for connection startup, for recovery after a packet loss, or for complete connection lifetimes. Instead, we are proposing RBP for the final few packets of a transfer, since the increase in transfer time is minimal, and the risk of packet loss is (hopefully) reduced. Note that this approach by itself does *not* require any changes to TCP/IP packet headers, nor does it require special support at the IP routers. It is an endpoint control algorithm only, which makes the TCP source more cautious.
- **Early Congestion Avoidance (ECA)**: A TCP source that is explicitly aware of the number of packets remaining in its Web document transfer may voluntarily (and prematurely) switch from the TCP slow-start algorithm to the TCP congestion avoidance algorithm. The TCP sender does this if it deems that linear growth of the congestion window will reduce the risk of packet loss, while having minimal adverse impact on expected document transfer time. For example, the additional round-trip time(s) incurred to complete the data transfer may be tolerable, compared to the risk of a coarse timeout. Note that this approach by itself does *not* require any changes to TCP/IP packets, nor does it require special router support within the network. This algorithm makes the TCP source less aggressive.
- **Selective Packet Marking (SPM)**: In a context-aware TCP protocol, the TCP source knows exactly how many packets there are in the Web document transfer. Among these packets, the first three and the last three are the most crucial in terms of their possible impact on the Web document transfer time in the event of a packet loss, as are any packets sent when the congestion window is small (i.e., less than 4 packets). A context-aware TCP source can selectively mark these crucial packets (about 35-40% of the packets for a typical 8 KB Web document transfer [4], [14], assuming 512-byte packets) "high" priority. All other packets of the transfer are marked with default "low" priority. Note that these priorities refer to *packet loss priority* only, not packet scheduling priority. The SPM approach requires a priority bit in the TCP/IP packet headers, and special packet handling at the routers based on this information. SPM by itself does not alter any other aspects of TCP's flow control, congestion control, and packet departure process.

Figure 2 illustrates these TCP variants. Figure 2(a) shows an example of a simulated Reno TCP transfer of a 7-kilobyte Web document, using ns-2. The graph illustrates the TCP slow-start behaviour, with exponential growth (e.g., 1, 2, 4, and 8 packets) of the TCP congestion window. Note that the last burst contains only 7 (rather than 8) packets, since the transfer only involves 14 packets (512 bytes each) in this case. Figure 2(b) shows Rate-Based Pacing of the Last Window (RBPLW). The TCP source voluntarily paces the last window's worth of packets, to reduce burstiness. Figure 2(c) shows Early Congestion Avoidance (ECA). The TCP source voluntarily restricts the congestion window to linear growth in the congestion avoidance phase. The result is a burst of 4 packets followed by a burst of 3 packets, rather than a single burst of 7 packets. The rationale for this approach is to avoid packet losses that may occur due to large bursts of back-to-back packets. Figure 2(d) shows RBPLW combined with ECA. In this case, only the last window of 3 packets is temporally spaced.

These context-aware TCP algorithms are evaluated in the experiments in Section V. In these experiments, different versions of the context-aware TCP protocol use the foregoing three mechanisms, either in isolation or in combination.

C. Adding Context-Awareness to IP

For the routers in our experiments, we consider five algorithms that vary in the amount of context information available and how it is used.

The algorithms are:

- **Drop Tail**:

The classic DropTail router uses a First-In-First-Out queue. Arriving packets enter the tail of the queue, and proceed onward for FIFO service. Arriving packets that encounter a full queue are dropped (discarded). This algorithm has no context information, and serves as a simple baseline for comparison.

- **Random Early Detection (RED)**:

The RED algorithm [16] seeks to maintain queue occupancy at or below a specified threshold level, and uses a probabilistic approach to discard packets from flows to limit excessive queue growth. The parameters of this algorithm determine the queue threshold at which packet discard begins, and how to compute the discard probabilities as a function of queue length. This approach can discard packets before the queue is full, and is thus classified as an active queue management approach. In our simulations, the RED approach has no Web/TCP/IP context information. It is used as a baseline for comparison, since this approach has been proposed for use on the Internet.

- **CATNIP-Good**:

The CATNIP-Good algorithm exploits the packet loss priority information provided by TCP sources using Selective Packet Marking (SPM). This algorithm modifies the FIFO structure of the DropTail router, by exploiting context information carried in the form of a priority bit in the TCP/IP packet header. In the normal case, arriving packets enter the tail of the queue, and proceed forward for FIFO service. A low-priority arriving packet that encounters a full queue is dropped, as in a DropTail router. However, a high-priority arriving packet that encounters a full queue may or may not be dropped, depending on the state of the queue. In particular, if there is at least one low-priority packet within the queue, one such low-priority packet (e.g., chosen at random) is removed from the queue, making room for the high-priority packet to be added at the tail of the queue. All

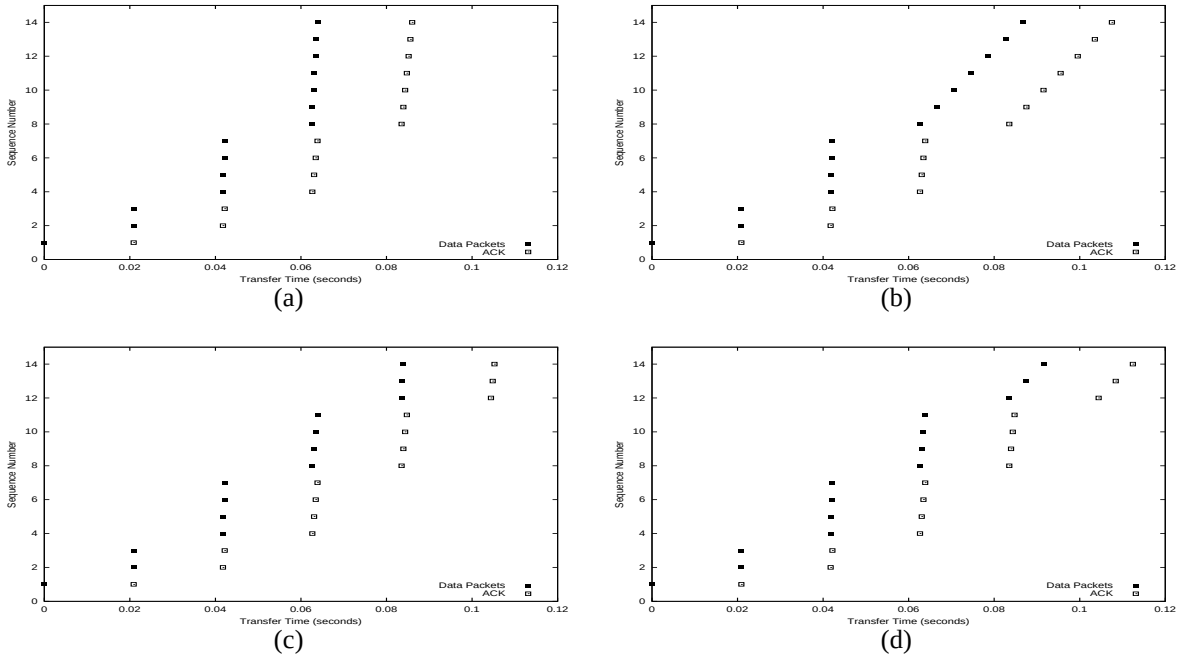


Fig. 2. Graphical Illustration of TCP Behaviours for a Simulated 14-packet Web Document Transfer: (a) Reno TCP; (b) Rate-Based Pacing of the Last Window (RBPLW); (c) Early Congestion Avoidance (ECA); (d) ECA+RBPLW

queued packets maintain their relative order. If the full queue contains no low-priority packets, then the arriving packet is always dropped.

- **CATNIP-Bad**:

Like "The Force" in the movie Star Wars, omniscient power can be used either for good or for evil. A CATNIP-Bad router uses the same "packet priority" context information that is available to a CATNIP-Good router, but in the opposite way. That is, the CATNIP-Bad router tries to discard high-priority TCP packets, rather than low-priority TCP packets, when the queue is full. It thus has maximal adverse effect on TCP sources. This algorithm is used as a worst-case comparison point to understand the impact of router queue management algorithm on Web/TCP performance.

- **CATNIP-RED**:

This queue management algorithm combines the RED approach and CATNIP-Good. That is, it uses the RED notion of early packet discarding to maintain queue occupancy below a threshold, but considers packet loss priority context information in its discard decisions. Whenever a packet must be discarded by the RED algorithm, an attempt is made to discard a low-priority packet rather than a high-priority packet. This algorithm is used to assess the interoperability between RED and CATNIP.

IV. Experimental Methodology

We initially evaluate the context-aware TCP/IP approach using simulation, with the ns-2 network simulator [10], [28]. The ns-2 simulator provides a detailed packet-level network simulation environment, complete with HTTP, TCP, and IP protocol models. For simplicity, we use the TCP Reno model and the DropTail router as the baseline for comparison in our experiments. We add various degrees of the CATNIP functionality to the TCP and IP models, and evaluate performance impacts in progressive steps. The following subsections provide more background on the setup for the simulation experiments, including simulated network topology, Web workload model, experimental design, performance metrics, and simulation validation. Simulation results are used to guide the implementation and evaluation of CATNIP in the Linux kernel, discussed in Section V-G.

A. Network Model and Assumptions

Figure 3 shows the network topology[3] used in the simulation experiments. There are 100 Web clients connected to a router (RouterC) via dedicated 10 Mbps links. There are 10 Web servers connected to another router (RouterS) via dedicated 10 Mbps links. All Web document transfers flow from the Web servers to the clients. There is a 1.5 Mbps bottleneck link between the two routers, shared by all traffic flows. All router queues are large enough to hold 50 packets, except for the queue at the bottleneck link, which can hold at most 10 packets. The propagation delays on the links are as indicated in Figure 3.

Each Web server receives requests from a distinct set of 10 Web clients. Clients are homogeneous. That is, each client uses the same version of TCP.

Clients generate requests for each Web document in the simulated workload, modeling the HTTP/1.0 protocol. A maximum of four concurrent (parallel) TCP connections are allowed between a given client-server pair. In most experiments, HTTP/1.1 is not modeled, though one experiment with HTTP/1.1 is discussed in Section V-F.

[3]This network model is similar to that used by Padmanabhan and Katz [23]. The primary difference is the shorter propagation delay for the bottleneck link in our network (5 milliseconds, rather than 50 or 200 milliseconds), to lessen the influence of RTT effects.

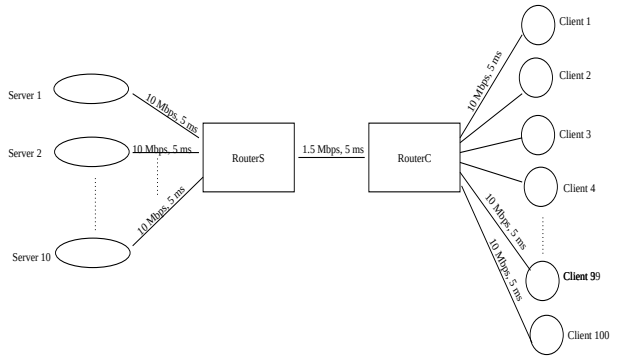


Fig. 3. Network Topology for Simulation Evaluation of Context-Aware TCP/IP

B. Web Workload Model

The workload for the simulation experiments is generated using the ns-2 Web workload models [20]. These models use empirically-observed distributions to determine the size (in bytes) of the HTML content of each Web page, the number of embedded images in each Web page, and the size (in bytes) of each of the embedded images. Each Web page is generated independently from these empirical distributions, based on a pseudo-random number sequence.

The precise workload used in our simulation experiments consists of 10 such randomly generated Web pages, whose structure and sizes are illustrated schematically in Figure 4. For each illustrated page, the rectangle in the top left corner represents the size of the base HTML file, and the other rectangles represent embedded images and their relative sizes (though not exactly to scale). For example, Page 1 in the generated workload consists of a 2,589-byte HTML file, and two embedded images (548 bytes, and 1,808 bytes, respectively). Each simulated Web page[4] has at least one embedded image. Page 5 has 16 embedded images, of widely varying sizes.

In the simulations, each user accesses these 10 Web pages, in random order. The think times between page downloads for each user are drawn independently from the empirically-derived think time models in the ns-2 simulator [20].

The entire workload consists of 1000 Web page downloads, which includes 4500 TCP connections, and about 46,000 TCP data packets. In the baseline configuration, a typical client in the simulations downloads the entire 216-kilobyte Web page workload in about 5 minutes of simulated time (including think times between pages), for an average per-user throughput of about 6 kilobits per second. Thus the average aggregate throughput for 100 clients is approximately 0.6 Mbps, though the peak throughput demands can exceed the capacity of the bottleneck link.

C. Experimental Design

There are two primary factors in our simulation experiments: TCP version, and the queue management algorithm at IP routers. A one-factor-at-a-time experiment is conducted using these factors. A summary of the experimental design appears in Table I.

[4]Note that the mean (21 KB) and median (9 KB) Web page sizes in our workload are much smaller than the 30, 50, and 175 KB transfers considered in the TCP "fast start" experiments by Padmanabhan and Katz [23] to highlight the advantages of CATNIP even for modest transfer sizes.

TABLE I
EXPERIMENTAL FACTORS AND LEVELS FOR EVALUATING CONTEXT-AWARE TCP/IP

Factor	Levels
TCP	Reno, RBPLW, ECA, ECA+RBPLW, SPM
IP	DropTail, RED, CATNIP-Good, CATNIP-Bad, CATNIP-RED

We consider five TCP versions. Reno TCP has no application-layer context information, while the other four TCP versions use different heuristics to exploit context information.

At the routers, we consider five queue management algorithms. These algorithms range from DropTail, which has no context information, to CATNIP-Good, which has complete context information.

D. Performance Metrics

Two performance metrics are considered in the experiments: the transfer time for each complete Web page, and the average packet loss in the simulated network. The aggregate transfer time for a Web page represents the latest completion time for the transfers of the individual components of the Web page, using the indicated version of the TCP protocol. For transfer times, we consider the mean and standard deviation of transfer times as the measures of interest, since the variability of transfer times can be significant in the presence of packet losses.

E. Validation

Validation of our context-aware TCP models was conducted using small network scenarios, small numbers of sources, and TCP sequence number plots of selected source behaviours (see Figure 2, for example). Detailed simulation output includes the start time, end time, transfer size, and packet loss occurrences for each individual TCP connection and for each Web page, by client. Analysis of these traces indicates that the TCP and IP models are working as intended.

V. SIMULATION AND EXPERIMENTAL RESULTS

This section presents the results from our experiments. Simulation results are presented in Sections V-A to V-F. Experimental results with our prototype implementation are discussed in Section V-G. Section V-H summarizes the results.

A. Simulation Results for DropTail Routers

The first set of experiments assumes DropTail routers in the simulated network topology. The experiments focus on what influence the TCP endpoints can have on Web page transfer times, using the TCP heuristics described in Section III-B. However, no Selective Packet Marking (SPM) is used in these experiments, since DropTail routers ignore packet priority.

Figure 5 shows the simulation results from the DropTail experiments, for the 100 simulated clients. Figure 5(a) shows the mean transfer times for each of the 10 Web pages in the simulated workload, while Figure 5(b) shows the standard deviations of the transfer times. On each graph, the 10 Web pages are represented across the horizontal axis, with transfer time on the vertical axis. For each Web page, there are four columns in

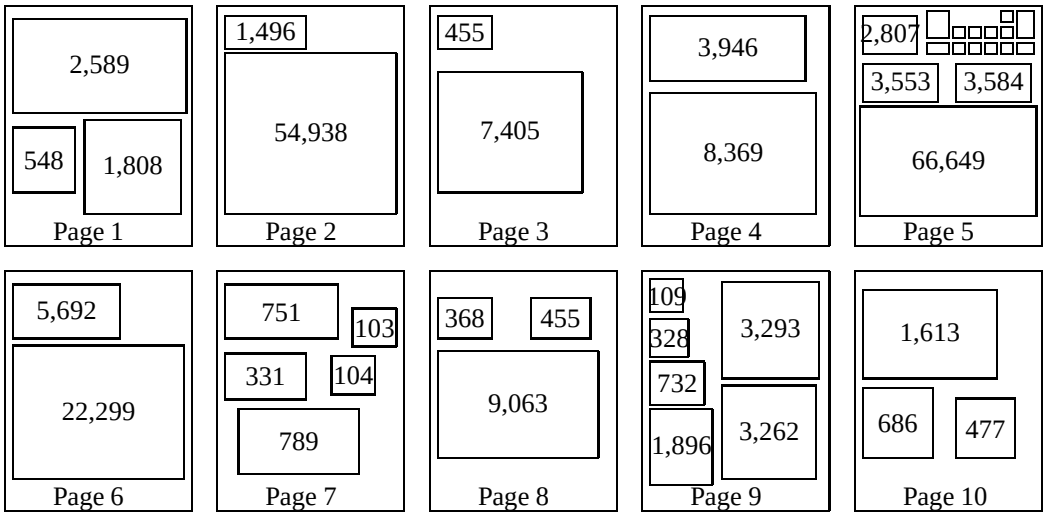


Fig. 4. Schematic Illustration of the Simulated Web Workload

the bar chart, one representing each TCP algorithm considered in the experiment. The left-to-right ordering of algorithms is Reno, Reno+RBPLW, ECA, and ECA+RBPLW.

The simulation results in Figure 5 show that each of the 10 simulated Web pages in the workload has its own distinct performance characteristics. This observation is not surprising, given that the 10 Web pages are diverse in size, structure, and number of TCP connections required. The performance results for the 10 Web pages should be viewed as representative examples of transfer times for typical Web pages. The trend in behaviour across this set of Web pages will be the main focus of our analyses and observations. For reference purposes, Table II and Table III provide detailed simulation results for two of the TCP algorithms (Reno and ECA, respectively) illustrated in Figure 5. Tabular results for the other algorithms are omitted, for space reasons.

Figure 5(a) shows that (unfortunately) the TCP endpoint control algorithms by themselves have little advantage to offer. For example, the Rate-Based Pacing of the Last Window (RBPLW) heuristic seems to improve the performance of TCP Reno in most cases (quite noticably for Web pages 1, 8, and 9, while having little or no positive effect on the other Web pages considered). The Early Congestion Avoidance (ECA) heuristic sometimes improves performance compared to Reno (e.g., pages 1 and 9), but is sometimes worse than Reno (e.g., pages 3 and 4). The RBPLW feature helps a bit when combined with ECA, though it has an adverse effect on several of the Web pages considered (e.g., pages 4, 7, and 9). The standard deviations of transfer times in Figure 5(b) are large, meaning that performance differences amongst TCP algorithms are not statistically significant. Simply stated, the results here are inconclusive.

Table IV provides a summary of the packet loss behaviours for the TCP algorithms with DropTail routers. The ECA and RBPLW heuristics each reduce packet loss. Adding RBPLW to Reno reduces packet loss from 3.46% to 3.10%, a reduction of 10%. Adding ECA to Reno reduces packet loss from 3.46% to 2.87%, a reduction of 17%. Adding RBPLW to ECA seems to mitigate the advantages of ECA.

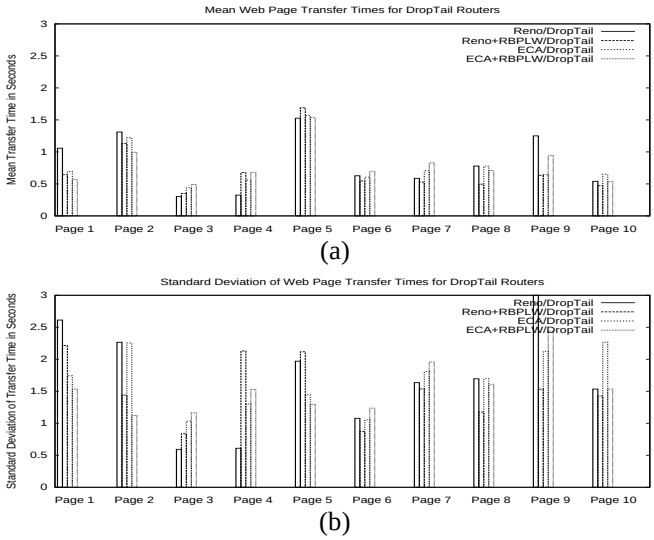


Fig. 5. Simulation Results for DropTail Routers

TABLE II
DETAILED SIMULATION RESULTS FOR RENO/DROPTAIL

Web Page	Total Conns	Total Bytes	Web Page Transfer Time				
			Min	Median	Max	Mean	SDev
1	3	4,945	0.134	0.134	18.131	1.058	2.613
2	2	56,434	0.450	0.771	18.630	1.311	2.266
3	2	7,860	0.184	0.186	6.036	0.302	0.593
4	2	12,315	0.202	0.215	6.239	0.325	0.610
5	17	87,451	0.844	1.049	18.098	1.527	1.970
6	2	27,991	0.276	0.328	6.939	0.627	1.078
7	5	2,078	0.103	0.103	6.127	0.587	1.634
8	3	9,886	0.205	0.213	6.239	0.779	1.695
9	6	9,620	0.190	0.195	18.353	1.251	3.414
10	3	2,776	0.103	0.108	6.147	0.541	1.534

TABLE III
DETAILED SIMULATION RESULTS FOR ECA/DROPTAIL

Web Page	Total Conns	Total Bytes	Web Page Transfer Time				
			Min	Median	Max	Mean	SDev
1	3	4,945	0.134	0.134	6.472	0.694	1.742
2	2	56,434	0.450	0.573	18.835	1.223	2.255
3	2	7,860	0.205	0.205	6.446	0.439	1.032
4	2	12,315	0.210	0.213	6.236	0.561	1.301
5	17	87,451	0.930	1.051	7.636	1.569	1.445
6	2	27,991	0.277	0.299	6.432	0.605	1.053
7	5	2,078	0.103	0.103	6.141	0.707	1.806
8	3	9,886	0.213	0.214	6.255	0.777	1.698
9	6	9,620	0.184	0.185	18.165	0.645	2.125
10	3	2,776	0.103	0.103	18.101	0.652	2.267

TABLE IV
SUMMARY OF PACKET LOSS RESULTS FOR DROPTAIL ROUTERS

TCP/IP Algorithm	All Packets		
	Sent	Drops	Loss
Reno/DropTail	47,918	1,656	3.46%
Reno+RBPLW/DropTail	47,632	1,477	3.10%
ECA/DropTail	47,429	1,359	2.87%
ECA+RBPLW/DropTail	47,641	1,466	3.08%

B. Simulation Results for CATNIP-Good Routers

The next set of experiments consider the addition of "context-awareness" at the IP routers in the simulated network. In particular, the routers use one of the versions of the CATNIP (Context-Aware Transport/Network Internet Protocol) algorithm. All TCP sources use the Selective Packet Marking (SPM) mechanism to indicate the packet loss priority, and the routers are explicitly aware of these priorities in the queue management.

Figure 6 shows the simulation results for the CATNIP-Good algorithm at the routers. As was done for the DropTail experiments, the graphical results present means (Figure 6(a)) and standard deviations (Figure 6(b)) of transfer times for each of the 10 Web pages, computed across the 100 clients in the simulation. On the graphs, each cluster of four columns represents the four TCP source algorithms considered, as indicated in the graph key. The extra column on the left (above the 'P' in 'Page') shows the baseline performance for the Reno/DropTail configuration, for which SPM is ignored. For reference purposes, Table V and Table Vi provide detailed simulation results for the Reno and ECA algorithms in the CATNIP-Good scenario.

Figure 6(a) shows the mean transfer time results for this experiment. The primary observation is that adding context-awareness at the IP routers (with the CATNIP-Good algorithm) improves the mean Web page transfer time for most of the Web pages in the workload. For some Web pages, the mean transfer time is reduced by a factor of 2 or more (e.g., pages 1, 7, 8, 9, and 10). Considering Reno/CATNIP-Good versus the baseline Reno/DropTail results, the improvements in mean transfer time range from 17% for Web page 5 to 82% for Web page 9. Most pages have improvements ranging from 24% to 70%. Five pages have an improvement of 65% (i.e., a factor of 3 reduction) or more. The relative impacts of the different TCP source variants are modest, compared to the impact of the CATNIP-Good algorithm. In general, the ECA algorithm performs similarly to Reno in most cases, though in some cases it is slightly worse. There is no advantage for the RBPLW algorithm in this scenario.

TABLE V
DETAILED SIMULATION RESULTS FOR RENO/CATNIP-GOOD

Web Page	Total Conns	Total Bytes	Web Page Transfer Time				
			Min	Median	Max	Mean	SDev
1	3	4,945	0.134	0.138	6.136	0.275	0.842
2	2	56,434	0.450	0.792	2.245	0.866	0.463
3	2	7,860	0.184	0.184	0.566	0.230	0.096
4	2	12,315	0.202	0.209	0.630	0.266	0.110
5	17	87,451	0.702	1.082	2.733	1.274	0.430
6	2	27,991	0.276	0.312	1.465	0.446	0.217
7	5	2,078	0.103	0.103	6.100	0.173	0.599
8	3	9,886	0.205	0.208	0.584	0.254	0.093
9	6	9,620	0.190	0.197	0.445	0.230	0.070
10	3	2,776	0.103	0.103	6.064	0.174	0.595

The mean transfer times decrease largely because the "outlier" transfer times are reduced or eliminated with CATNIP-Good. That is, the context information at the router reduces the number of "painful" packet losses for TCP sources. This improvement is evident in Tables V and VI compared to Tables II and III. Note the lower maximum[5] transfer times, the lower mean transfer times, and the lower standard deviations.

This performance improvement is indicated more clearly in Figure 6(b), which shows the standard deviations of the transfer times for each Web page, across the 100 clients. Compared to the Reno/DropTail scenario, there is a dramatic reduction in the variability of Web page transfer time for each Web page considered, when the routers use the CATNIP-Good algorithm for queue management. For example, the standard deviations for Reno/CATNIP-Good are 60% (page 10) to 95% (pages 8 and 9) lower than for Reno/DropTail. For several Web pages, the variance of the transfer time approaches zero (e.g., page 3 in Table V and Table VI.

Table VII summarizes the packet loss results for CATNIP-Good routers. The results show that about 40% of the packets carried on the network are marked high priority, and that the losses (as expected) are heavily biased toward the low priority packets. Surprisingly, the average packet loss rates with CATNIP-Good are *higher* than for the DropTail scenario, by about 10% (e.g., 3.74% versus 3.46% for Reno). The reason for this is that TCP sources experience fewer coarse timeouts, and fewer resets (i.e., $cwnd = MSS$) of the congestion window, when CATNIP-Good routers are used. As a result, TCP sources tend to remain more aggressive, and have (on average) a larger congestion window size. The counter-intuitive relationship between packet loss rate and mean Web page transfer time serves to reinforce our point that not all packet losses are equal.

As noted previously, the ECA and RBPLW heuristics each reduce packet loss. However, the most important observation is that packet losses are shifted away from the high priority TCP packets, using the CATNIP-Good algorithm (see Table VII).

[5]The worst-case transfer times remaining are in the range of 6 seconds, most likely reflecting a timeout caused by the loss of the very first packet of a transfer. There is little that a TCP source can do about this case.

TABLE VII
SUMMARY OF PACKET LOSS RESULTS FOR CATNIP-GOOD ROUTERS

TCP/IP Algorithm	All Packets			High Priority			Low Priority		
	Sent	Drops	Loss	Sent	Drops	Loss	Sent	Drops	Loss
Reno/CATNIP-Good	48,126	1,799	3.74%	19,126	93	0.49%	29,000	1,706	5.88%
Reno+RBPLW/CATNIP-Good	47,966	1,751	3.65%	19,228	81	0.42%	28,738	1,670	5.81%
ECA/CATNIP-Good	47,791	1,505	3.15%	18,836	51	0.27%	28,955	1,454	5.02%
ECA+RBPLW/CATNIP-Good	48,087	1,836	3.82%	19,876	96	0.48%	28,211	1,743	6.18%
Reno/DropTail	47,918	1,656	3.46%	19,008	514	2.70%	28,910	1,142	3.95%

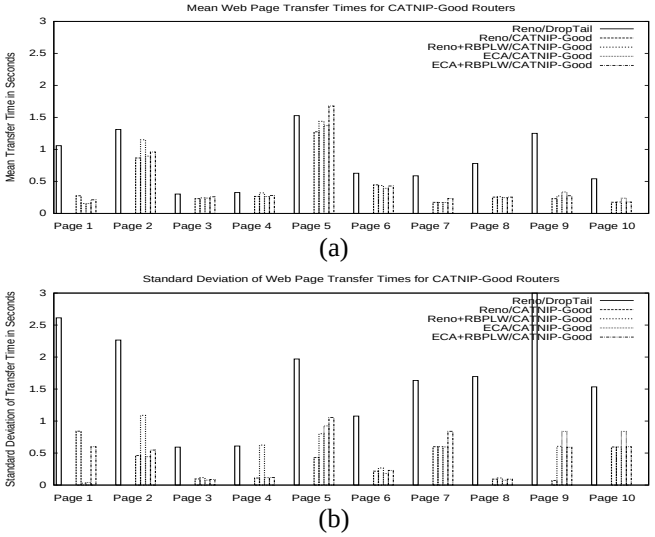


Fig. 6. Simulation Results for CATNIP-Good Routers

TABLE VI
DETAILED SIMULATION RESULTS FOR ECA/CATNIP-GOOD

Web Page	Total Conns	Total Bytes	Web Page Transfer Time				
			Min	Median	Max	Mean	SDev
1	3	4,945	0.134	0.135	0.390	0.153	0.040
2	2	56,434	0.450	0.811	2.440	0.895	0.443
3	2	7,860	0.205	0.206	0.580	0.239	0.077
4	2	12,315	0.210	0.214	0.828	0.264	0.116
5	17	87,451	0.935	1.072	6.225	1.371	0.924
6	2	27,991	0.277	0.301	0.929	0.392	0.181
7	5	2,078	0.103	0.103	6.073	0.170	0.596
8	3	9,886	0.213	0.215	0.609	0.246	0.076
9	6	9,620	0.184	0.190	6.167	0.334	0.839
10	3	2,776	0.103	0.103	6.088	0.238	0.839

C. Simulation Results for CATNIP-Bad Routers

The next experiment considers the CATNIP-Bad algorithm at the routers. The IP routers have explicit TCP/Web context information provided to them by Selective Packet Marking (SPM) at the TCP sources. However, the routers intentionally violate the intended semantics of the packet loss priority indication, by throwing away high priority packets when congested. The purpose of this experiment is to understand the "worst-case" influence of routers on Web transfer performance.

Tables VIII and IX provide detailed simulation results for the Reno and ECA algorithms in the CATNIP-Bad scenario. (The graphical results are omitted for space reasons.) Many of the Web pages have worst-case download times of 18 seconds or more. As expected, the primary influence of CATNIP-Bad is to throw away the "wrong packet" at the "wrong time", when network congestion occurs, making matters worse.

Table X summarizes the packet loss results for the various TCP algorithms considered, for CATNIP-Bad routers. The results show that packet losses are shifted to the high priority TCP packets, leading to the poor Web data transfer performance mentioned previously.

D. Simulation Results for RED and CATNIP-RED Routers

The next set of experiments considers RED and CATNIP-RED algorithms at the simulated routers. The purpose of this experiment is to assess the impact of the RED algorithm, which is arguably "smarter" than DropTail, but not as smart as CATNIP-Good. The CATNIP-RED approach combines the features of RED and CATNIP-Good. In our experiments, the RED algorithm uses a minimum queue threshold of 20%, a maximum queue threshold of 60%, and a queue weight of 0.01.

The simulation results for these scenarios appear in Figure 7. These results are presented using the same format as the previous results, showing mean (Figure 7(a)), and standard deviation (Figure 7(b)) of Web page transfer times for the 100 clients. One difference here is the four algorithms shown in the graphs: these are Reno/RED, Reno/CATNIP-RED, ECA/RED, and ECA/CATNIP-RED, in that order. The Reno/DropTail results are again included for comparison purposes (the leftmost column for each Web page considered). No RBPLW results are included for these experiments, since RBPLW seems to offer no performance improvement.

The results in Figure 7 show that mean and standard deviation of Web page transfer times are affected. The mean transfer times tend to increase slightly with RED (pages 3, 4, 5, 6, 8, and 10), because the overall level of packet loss (3.8-4.1%) is slightly higher (since RED can discard packets before the router queue is full). The effect of CATNIP-RED is far greater than the effect of ECA; in fact, Reno and ECA perform similarly in almost all cases. The results also show that the CATNIP-RED algorithm has a consistently positive impact for all the Web pages in the workload. These results indicate that RED and CATNIP-Good can interoperate in a complementary fashion.

E. Simulation Results for Fairness

An obvious concern that arises is the "TCP-friendliness" of CATNIP if it were to share a network link with non-CATNIP TCP sources. To investigate this fairness issue, a simple simula-

TABLE X
SUMMARY OF PACKET LOSS RESULTS FOR CATNIP-BAD ROUTERS

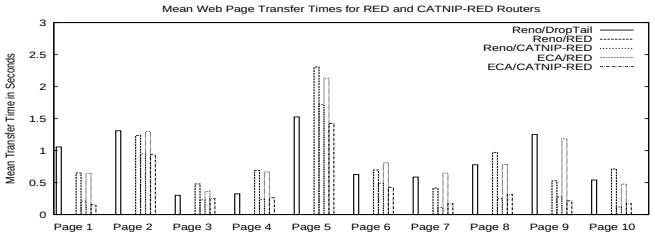
TCP/IP Algorithm	All Packets			High Priority			Low Priority		
	Sent	Drops	Loss	Sent	Drops	Loss	Sent	Drops	Loss
Reno/CATNIP-Bad	47,690	1,847	3.87%	19,317	1,320	6.83%	28,373	527	1.86%
ECA/CATNIP-Bad	47,250	1,501	3.18%	18,842	1,165	6.18%	28,408	336	1.18%
Reno/DropTail	47,918	1,656	3.46%	19,008	514	2.70%	28,910	1,142	3.95%

TABLE VIII
DETAILED SIMULATION RESULTS FOR RENO/CATNIP-BAD

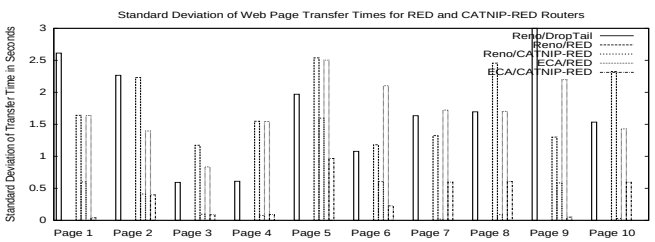
Web Page	Total Conns	Total Bytes	Web Page Transfer Time				
			Min	Median	Max	Mean	SDev
1	3	4,945	0.134	0.134	18.150	1.019	2.570
2	2	56,434	0.450	0.475	18.077	1.445	2.864
3	2	7,860	0.184	0.187	42.037	1.918	5.192
4	2	12,315	0.202	0.209	42.171	1.497	5.046
5	17	87,451	0.663	1.049	43.014	4.410	7.095
6	2	27,991	0.276	0.291	42.419	1.859	6.088
7	5	2,078	0.103	0.103	18.131	0.835	2.461
8	3	9,886	0.205	0.209	6.282	0.894	1.870
9	6	9,620	0.190	0.195	18.168	1.083	2.548
10	3	2,776	0.103	0.103	6.170	0.601	1.633

TABLE IX
DETAILED SIMULATION RESULTS FOR ECA/CATNIP-BAD

Web Page	Total Conns	Total Bytes	Web Page Transfer Time				
			Min	Median	Max	Mean	SDev
1	3	4,945	0.134	0.134	6.488	0.697	1.738
2	2	56,434	0.450	0.474	18.567	1.162	2.235
3	2	7,860	0.205	0.205	18.210	1.315	3.102
4	2	12,315	0.210	0.212	18.476	0.815	2.292
5	17	87,451	0.701	0.967	18.110	2.523	2.914
6	2	27,991	0.277	0.289	6.570	0.662	1.325
7	5	2,078	0.103	0.103	42.098	2.387	5.704
8	3	9,886	0.213	0.220	6.944	1.030	2.025
9	6	9,620	0.184	0.186	18.187	1.604	3.595
10	3	2,776	0.103	0.103	18.085	1.020	2.992



(a)



(b)

Fig. 7. Simulation Results for RED and CATNIP-RED Routers

tion experiment was conducted with 50 ECA sources sharing the network simultaneously with 50 Reno sources. While the ECA sources had lower average packet loss (3.05% versus 3.68% for DropTail routers, and 3.23% versus 3.95% for CATNIP-Good routers), there was no performance advantage or disadvantage for ECA in terms of mean Web page retrieval time.

F. Simulation Results for HTTP/1.1

To estimate the performance advantages of our context-aware TCP/IP protocol in the presence of HTTP/1.1, we conduct a simple simulation experiment with an approximate model of HTTP/1.1. It is the "persistent connection" feature of HTTP/1.1 that is of interest here, since it allows a single TCP connection to be maintained and used throughout the duration of a user session containing multiple HTTP GET requests.

Our approximate model of HTTP/1.1 reuses the Web page workload from the previous simulation experiments, with the additional assumption that each Web page is transferred *in its entirety* using a *single* TCP connection. The aggregate byte transfer volume of this connection is simply the sum of the sizes of the objects within each page. This approximate model assumes that all HTTP GET requests are pipelined on the same TCP connection, and that each request has zero processing time. As a result, the TCP transfer maintains its connection state throughout the aggregate transfer. This model presents an optimistic view of HTTP/1.1 transfer performance.

The main design question is whether the Selective Packet Marking (SPM) feature of our context-aware TCP/IP protocol should be applied at the application-layer document boundaries (i.e., selective marking is done for the first few packets and the last few packets of each transferred document) or based only on TCP state (i.e., selective marking is done for the first few packets and the last few packets of the TCP connection, plus any packets sent when the TCP congestion window is small). We make the latter choice, since it seems more natural. This choice also provides a pessimistic assessment of the potential benefits of SPM (since a smaller proportion of the TCP packets are marked).

The simulation results from our HTTP/1.1 experiments show that Reno/CATNIP-Good offers 7-76% improvement in mean Web page transfer times compared to Reno/DropTail, and 58-98% reduction in standard deviations. In these experiments, only 20% of the total packets were marked high priority, and only 2.59% of the total packets were lost. We thus conclude that the CATNIP approach offers benefits even with HTTP/1.1.

G. Experimental Implementation and Evaluation

To explore the practical implementation issues associated with context-aware TCP/IP, we undertook a "proof of con-

cept" prototype implementation in the Linux 2.4.16 kernel. Our implementation effort focussed only on the SPM feature of context-aware TCP, since the simulation experiments show that it is the feature that enables the greatest performance advantages. We use the (reserved) high-order bit of the TCP flags field to convey packet priority context information.

Implementing SPM in the Linux TCP/IP protocol stack required adding a module that conveys application-layer document size information to the TCP socket layer.

The kernel modules affected by our modifications are `tcp.h`, `tcp.c`, `tcp_output.c`, and `netsyms.c`. We verified the correct operation of our packet marking TCP code using `tcpdump` traces from our experimental environment.

Due to the imminent paper deadline, we have only a *partial* implementation of SPM working at this time. This version provides selective packet marking of the first few TCP packets of a connection, and all packets sent with a small congestion window. Marking of the last few packets of a transfer is only possible if the TCP sequence number information relative to the total transfer size is known. The full implementation requires two additional state variables in the socket data structure: an *initial sequence number*, and a *document size*. This kernel is being built and tested right now.

We evaluated our prototype implementation in a WAN emulation environment using the Internet Protocol Traffic and Network Emulator (IP-TNE) [24]. A desktop PC (Sony Vaio, 1 GHz AMD processor, 128 MB SDRAM) was used to run the Apache Web server software (version 1.3.19-5) on top of our modified Linux 2.4.16 kernel. The client workload was generated using 100 emulated clients in the IP-TNE, traversing an emulated network topology similar to that in Figure 3 (with the exception that only 1 Web server is used, not 10). Each client generates requests for the 10 Web pages (see Figure 4), which are physically stored on the Web server. User think times were set to zero for these preliminary experiments, and concurrent TCP connections at the emulated clients were disabled. Only HTTP/1.0 is modeled, since the HTTP/1.1 model in IP-TNE is still under development (completion anticipated early in March 2002).

The primary factor in our experiments is the buffer size at the emulated IP router attached to the 1.5 Mpbs bottleneck link. The buffer size affects the overall level of packet loss in the emulated network. We consider buffer size values from 64 KB to 512 KB, which span the range of interest for relevant levels of TCP packet loss (0% for 512 KB, and about 10% for 64 KB).

The results from our WAN emulation experiments are summarized in Table XI. Each numerical entry in the table shows the average relative download time for a Web page, where the relative download time represents the ratio of the download time with the CATNIP approach (at the Web server and in the emulated IP router) to the download time with the default Linux TCP implementation (and FIFO routers in IP-TNE). Relative download times are used to normalize the results across the diverse Web page workload used, and to fit all results into one table (for space reasons). Values smaller than 1.0 indicate a performance advantage for the CATNIP approach.

The preliminary results in Table XI show that our *partial* implementation of the CATNIP approach provides performance advantage for most (but not all) of the Web document downloads in this simple experiment. The performance advantages of CATNIP are most evident at modest levels of packet loss, and diminish (as expected) as the overall level of packet loss in the network decreases. Performance advantages will improve when the full SPM functionality is added.

TABLE XI
RELATIVE DOWNLOAD TIMES FOR WAN EMULATION EXPERIMENTS WITH CATNIP-GOOD

Web Page	Total Conns	Total Bytes	Router Buffer Size			
			64 KB	128 KB	256 KB	512 KB
1	3	4,945	0.95	0.88	0.95	0.96
2	2	56,434	0.79	1.13	1.10	1.14
3	2	7,860	0.88	0.67	0.95	0.98
4	2	12,315	0.64	0.69	0.97	0.98
5	17	87,451	0.93	0.96	1.00	1.02
6	2	27,991	1.01	1.05	0.93	0.97
7	5	2,078	1.02	0.89	0.96	0.96
8	3	9,886	1.38	1.04	0.99	0.97
9	6	9,620	0.82	0.77	0.96	0.96
10	3	2,776	0.85	0.86	0.96	0.98
Mean			0.93	0.89	0.98	0.99
StdDev			0.18	0.15	0.05	0.05
TCP Pkt Loss			9.66%	4.74%	0.22%	0.00%

H. Summary of Results

Application-layer context information about Web document transfer size seems to provide valuable context information for the TCP/IP protocols. The most dramatic performance improvements occur when priority information is available in the packet discard decision-making process. The use of the CATNIP-Good algorithm can provide significant reduction in the mean and variance of Web page transfer times. These reductions range from 20% to 80% for the Web page workload considered in our simulation study. Conversely, the CATNIP-Bad algorithm can increase mean and variance of Web transfer times significantly.

The TCP source heuristics considered in this paper offer rather modest performance improvements in Web document transfer time. Their improvements are more pronounced for simple DropTail routers, and are negligible for CATNIP routers. In general, the Early Congestion Avoidance heuristic reduces overall packet loss, but its impact on Web transfer time is small. The heuristic for Rate-Based Pacing of the Last Window of packets seems to offer inconsistent performance results, particularly when combined with other algorithms, and is of little value.

VI. CONCLUSIONS

This paper has presented the design and evaluation of CATNIP, a context-aware Internet protocol that integrates TCP-layer and IP-layer functionality through the use of application-layer knowledge about Web document transfers. Our experiments evaluate a range of mechanisms that can be used at the TCP source to exploit application-layer context information in flow and congestion control decision-making. Experiments also consider a variety of queue management algorithms at IP routers for exploiting context information.

Simulation results show that a 10-20% reduction in TCP packet loss is possible using simple endpoint control mechanisms alone, without any adverse impact on Web page retrieval

times. More importantly, queue management algorithms at the router have a dramatic impact on the mean and standard deviation of Web page retrieval times. The best results are achieved with CATNIP context information at the routers: 20-80% reductions in the mean Web page retrieval times, and 60-90% reductions in the standard deviations. Furthermore, the CATNIP algorithm can interoperate with RED or its variants for active queue management at IP routers. Preliminary experiments with a prototype implementation of the CATNIP approach in the Linux kernel of an Apache Web server demonstrate performance advantages of up to 40% for Web page retrieval time.

The main "take home" messages in our paper are:

- Not all packet losses are created equal.
- A TCP source alone has very limited control over end-to-end Web data transfer performance, even when application-layer context information is available.
- The IP packet forwarding layer has a significant influence on end-to-end Web data transfer performance, particularly when application-layer context information is available.
- A simple change to the TCP/IP protocol stack implementation can provide the context information required at the TCP/IP protocol layers.
- Changes to the queue management algorithms at routers can provide significant performance advantages for the context-aware TCP/IP approach.

We use these observations to argue the desirability of a layer-violating context-aware approach to TCP/IP, at least for Web document transfer on the Internet. Other uses for the context-aware approach are under consideration.

Ongoing work involves a fuller evaluation of context-aware TCP/IP protocols in our wide-area IP network emulation environment, and an investigation of the many practical issues facing the development and deployment of context-aware TCP/IP protocols on the Internet (e.g., socket API, TCP/IP headers, incremental deployment, fairness, interoperability, and HTTP/1.1). These issues clearly require further investigation.

ACKNOWLEDGEMENTS

Financial support for this research was provided by iCORE (Informatics Circle of Research Excellence) in the Province of Alberta, and by the Natural Sciences and Engineering Research Council of Canada, through NSERC Research Grant OGP0121969. The authors are grateful to Martin Arlitt and several anonymous reviewers for providing constructive feedback on an earlier version of this paper.

The network emulation experiments described in this paper would not have been possible without the Internet Protocol Traffic and Network Emulator (IP-TNE) developed by the TeleSim research group at the University of Calgary. In particular, Roger Curry extended the buffer management implementation in IP-TNE to support packet priorities and active queue management, as needed by CATNIP. Rob Simmonds and Tianbo Kuang both helped with our Linux implementation of CATNIP TCP.

APPENDIX A: TCP OVERVIEW

TCP is a connection-oriented, end-to-end reliable-byte-stream transport-layer protocol [25], [26]. It is widely used on the Internet and in the Web.

The fundamental unit of data transfer in TCP is a byte (i.e., for sequence numbering, flow control, and error control purposes). However, TCP implementations generally work with a larger logical unit size called a *segment* when transmitting packets across an IP internetwork. The Maximum Segment Size (MSS) is a settable parameter for a TCP transfer. The choice of the MSS typically depends on the Maximum Transfer Unit (MTU) size supported by the underlying network layer. In most instances, each TCP segment is carried in one IP packet; hence we use the terms segment and packet interchangeably throughout the paper. The task of TCP is to divide the application-layer data into one or more segments, transmit them across the network, and deliver them reliably (and in order) to the receiving TCP. Each segment carries an explicit sequence number, for the purposes of ordering and reliability.

There are several mechanisms in TCP to ensure reliable packet delivery. For example, when a sender transmits a segment, it starts a timer. If this segment is received successfully, then the receiver sends back an acknowledgement (ACK). Technically, the receiver is not required to generate the ACK immediately. Rather, the receiver can delay the ACK (up to 200 milliseconds), in the hope that the ACK can be piggybacked on an outgoing data packet, or that multiple incoming packets (closely spaced in time) can be acknowledged with a single (cumulative) outgoing ACK. In either case, an ACK indicates the next expected TCP sequence number. The sender uses the ACK for flow control and error control purposes, as well as to estimate the round-trip time (RTT) to the destination. If the timer expires before an ACK is received, then the sender retransmits the outstanding segment. Another commonly used strategy is fast retransmit [15], which uses duplicate ACKs to trigger the retransmission of a missing segment, typically well before the retransmission timer expires. This approach works well in recovering from single packet losses [13].

TCP uses a sliding window flow control mechanism that limits the maximum number of bytes that can be outstanding (i.e., not yet acknowledged) between a sender and a receiver at any time. A sender is allowed to transmit the segments in a window as quickly as it wishes (assuming data is available to transmit). As ACKs are received, the flow control window advances, and new segments can be transmitted.

A congestion control mechanism was added to TCP in 1988, based on algorithms proposed by Jacobson [18]. These algorithms use adaptive window-based flow control to achieve congestion control, since the IP network layer in the Internet does not provide congestion control.

In TCP congestion control, the flow control window size is adjusted dynamically based on two TCP state variables: the congestion window (*cwnd*), and the slow-start threshold (*ssthresh*). The initial value of *cwnd* is one segment, and *cwnd* is increased as successful ACKs are received. The increase is exponential in the slow-start phase (i.e., doubling *cwnd* every RTT, until *ssthresh* is reached), and linear in the congestion avoidance phase (i.e., increasing *cwnd* by one segment for every complete window's worth of data exchanged) [18].

TCP uses packet loss (due to buffer overflow at a router) as an implicit signal of network congestion. Each time a packet loss is detected, TCP updates its estimate of the slow-start threshold

(e.g., $ssthresh = cwnd/2$), reduces its congestion window size (e.g., $cwnd = MSS$), and re-enters the slow-start phase. The *fast recovery* [15] mechanism reduces the congestion window size by half (e.g., $cwnd = cwnd/2$) following a fast retransmit, rather than reducing it to one segment.

The foregoing algorithms are part of most TCP implementations, including Reno TCP and New Reno TCP that are widely used on the Internet today [3], [15], [22].

REFERENCES

[1] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the Performance of TCP Pacing", *Proceedings of IEEE INFOCOM*, Tel Aviv, Israel, March 2000.

[2] M. Allman, C. Hayes, and S. Ostermann, "An Evaluation of TCP with Larger Initial Windows", *ACM Computer Communication Review*, Vol. 28, No. 3, pp. 41-52, July 1998.

[3] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[4] M. Arlitt and C. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications", *IEEE/ACM Transactions on Networking*, Vol. 5, No. 5, pp. 631-645, October 1997.

[5] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Solutions", *Proceedings of IEEE INFOCOM*, San Francisco, CA, March 1998.

[6] H. Balakrishnan, S. Seshan, M. Stemm, and R. Katz, "Analyzing Stability in Wide-Area Network Performance", *Proceedings of ACM SIGMETRICS*, Seattle, WA, pp. 2-12, June 1997.

[7] P. Barford and M. Crovella, "Measuring Web Performance in the Wide Area", *ACM Performance Evaluation Review*, Vol. 27, No. 2, pp. 35-46, September 1999.

[8] P. Barford and M. Crovella, "Critical Path Analysis of TCP Transactions", *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, pp. 127-138, September 2000.

[9] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance", *Proceedings of ACM SIGCOMM*, San Francisco, CA, pp. 24-35, September 1993.

[10] L. Breslau *et al.*, "Advances in Network Simulation", *IEEE Computer*, Vol. 33, No. 5, pp. 59-67, May 2000.

[11] D. Clark and D. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", *Proceedings of ACM SIGCOMM*, September 1990.

[12] L. Eggert, J. Heidemann, and J. Touch, "Effects of Ensemble-TCP", *ACM Computer Communication Review*, Vol. 30, No. 1, pp. 15-29, January 2000.

[13] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", *ACM Computer Communication Review*, Vol. 26, No. 3, pp. 5-21, July 1996.

[14] A. Feldmann, J. Rexford, and R. Caceres, "Efficient Policies for Carrying Web Traffic Over Flow-Switched Networks", *ACM/IEEE Transactions on Networking*, Vol. 6, No. 6, pp. 673-685, December 1998.

[15] S. Floyd, "A Report on Recent Developments in TCP Congestion Control", *IEEE Communications*, Vol. 39, No. 4, pp. 84-90, April 2001.

[16] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE Transactions on Networking*, Vol. 1, No. 4, pp. 397-413, August 1993.

[17] J. Hoe, "Improving the Start-up Behavior of a Congestion Scheme for TCP", *Proceedings of ACM SIGCOMM*, Stanford, CA, pp. 270-280, August 1996.

[18] V. Jacobson, "Congestion Avoidance and Control", *Proceedings of ACM SIGCOMM*, Stanford, CA, pp. 314-329, August 1988.

[19] J. Ke, "Towards a Rate-Based TCP Protocol for the Web", *Proceedings of MASCOTS'2000*, San Francisco, CA, pp. 36-45, October 2000.

[20] B. Mah, "An Empirical Model of HTTP Network Traffic" *Proceedings of IEEE INFOCOM*, April 1997.

[21] J. Mogul, "The Case for Persistent Connection HTTP", *Proceedings of ACM SIGCOMM*, Boston, MA, August 1995.

[22] J. Padhye and S. Floyd, "On Inferring TCP Behavior" *Proceedings of ACM SIGCOMM*, San Diego, CA, pp. 287-298, August 2001.

[23] V. Padmanabhan and R. Katz, "TCP Fast Start: A Technique for Speeding up Web Transfers", *Proceedings of IEEE GLOBECOM'98 Internet Mini-Conference*, Sydney, Australia, pp. 41-46, November 1998.

[24] R. Simmonds, R. Bradford, and B. Unger, "Applying Parallel Discrete Event Simulation to Network Emulation", *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS)*, Bologna, Italy, pp. 15-22, May 2000.

[25] W. Stevens. *TCP/IP Illustrated, Volume 1*, Addison-Wesley, New York, 1994.

[26] A. Tanenbaum, *Computer Networks*, Third Edition, Addison-Wesley, 1996.

[27] K. Thompson, G. Miller, and R. Wilder, "Wide-area Internet Traffic Patterns and Characteristics", *IEEE Network*, Vol. 11, No. 6, pp. 10-23, November/December 1997.

[28] UCB/LBNL/VINT Network Simulator ns2.1, http://www-mash.cs.berkeley.edu/ns.

[29] V. Visweswaraiah and J. Heidemann, "Improving Restart of Idle Connections", Technical Report 97-661, University of Southern California, November 1997.

[30] H. Wang, "A New Scheme for TCP Congestion Control: Smooth-Start and Dynamic Recovery", *Proceedings of MASCOTS'98*, pp. 69-75, Montreal, Canada, July 1998.