Understanding End-to-End Performance: Testbed and Primary Results

Jia Wang, Yu Zhang, Srinivasan Keshav

Abstract—**As the Internet infrastructure evolves to include Quality of Service (QoS), a lot of work have been done on how to allocate network resources to satisfy the QoS requirements of IP flows. Less attention is being paid on mapping the network QoS specifications, e.g., network delay and loss rate, to the (perceived) end-to-end performance of user applications, e.g., the latency of retrieving a Web page. In this paper, we study the impact of the network QoS on the perceived end-to-end performance of user applications. We first propose a generic testbed using a combination of simulation and emulation technics. It can be used to evaluate the end-to-end performance of user applications in different network environments. Next, we use this testbed in studying the effect of network QoS metrics on the perceptual quality of various user applications. In this paper, we focus on estimating the latency of Web retrieval under given packet delay and loss rate and derive an accurate and efficient TCP short connection performance model.**

I. Introduction

The Internet is emerging as the single networking infrastructure that carries the data, audio, and video traffic. It has long been recognized that this convergence requires the Internet to provide corresponding QoS guarantees to user applications. With the emergence of many Internet QoS models [1] [2] [3], an interesting research issue arises, that is, how to map network QoS metrics to the performance observed by the user applications. Internet QoS models are typically defined in terms of bounds on network performance metrics such as bandwidth, delay, delay jitter and loss rate. However, the notion of service quality in these models, in terms of network performance metrics, is not the ultimate service quality delivered to end users, such as the retrieval latency of a Web page experienced by a user. We need to bridge the gap between network layer performance and the corresponding application layer performance.

We begin our study with designing a new generic experimental testbed. The testbed is a combination of emulation and simulation. The composite setting provides a simple and accurate experimental environment to study the behavior of the effect of network performance metrics on the applications. We study the end-to-end performance of various user applications on the testbed. Due to the space limitation, we focus on estimating latency of Web retrieval. We propose a new TCP short connection performance model which estimates the retrieval latency of a HTTP request given the requested file size and the network delay and loss characteristics *apriori*. In our study, the network performance is modeled by network delay and loss rate. We also assume that the losses are random and path symmetric.

Section II describes our testbed setting. Section III studies the Web performance under different network performance metrics and proposes a new model to estimate the latency of Web retrieval. Section IV addresses related work in the literature. We conclude our work in Section V.

Jia Wang is with AT&T Lab - Research, Florham Park, NJ, USA; Yu Zhang is with Cisco Systems, Inc., CA, USA; Srinivasan Keshav is with Ensim Corp., CA, USA. The work was done at Cornell University.

Fig. 1. The network model.

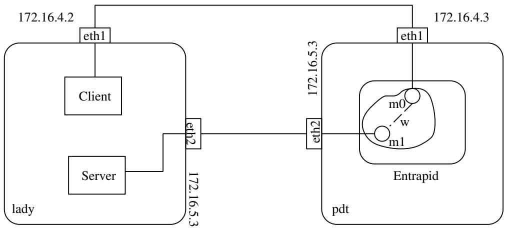

Fig. 2. The testbed setting: machine *lady* runs application entities and machine *pdt* runs the *Entrapid*.

II. Experimental testbed setting

Our experimental environment is modeled as two components: the *application entities* (e.g., Web browsers and Web servers) and the *network* connecting them (Figure 1). The rationale behind is that, from the point of view of the applications, the network between two application entities can be abstracted as a virtual link with its characteristics defined by the network performance metrics. Correspondingly, our testbed consists of a simulated network using the *Entrapid* simulator [8] and real implementation of applications connected to the simulated network using the *RealNet* technology in the *Entrapid*. This composite setting has two advantages: (i) *Simplicity.* We eliminate the need to modify, recompile, and relink applications with the network simulator; (ii) *Accuracy.* Since the implementation of applications is not modified, the behavior of the applications is exactly as it would be were it running on an actual network.

Figure 2 shows the testbed setting. We denote each Ethernet interface by its IP address. The testbed consists of two PCs (running Linux 2.0.34.) connected by two Ethernet cables with *Entrapid* simulator running on one and the application entities running on the other. We use *Entrapid* to simulate the network environment expected to examine. Inside the *Entrapid*, two virtual machines, m_0 and m_1, and a virtual link w connecting m_0 and m_1 are created to simulate the network. We bind each virtual machine to a network interface (i.e., m_0 to the interface 172.16.4.3 and m_1 to the interface 172.16.5.3). We also bind the application entities, which are running on the other PC, to specified network interfaces. For instance, we bind the application client to the interface 172.16.4.2 and bind the application server to the interface 172.16.5.2.

The routing tables at the network interfaces of these two machines are configured in such a way that all packets exchanged between the application entities are enforced to go through the virtual link w. For example, a client HTTP request will traverse the interface 172.16.4.2, interface 172.16.4.3, virtual machine m_0, link w, vitual machine m_1, interface 172.16.5.3, interface 172.16.5.4 and arrives at the server. The server's reply will traverse all the way back to the client. By tuning the delay and loss characteristics of the link w, we simulate different network environment of interest and study user applications performance.

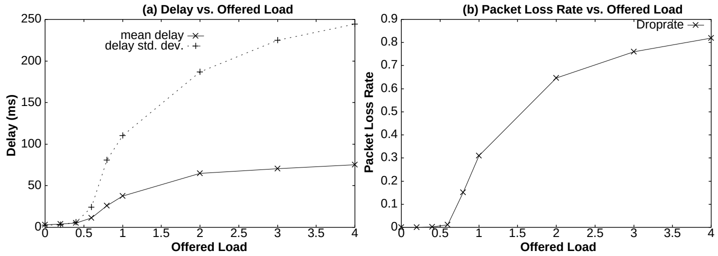

Fig. 3. The observed network performance metrics (mean packet size = 512 bytes): (a) mean and variance of packet delay vs. offered load, (b) packet loss rate vs. offered load.

III. Estimating Web performance

We study the Web performance under different network characteristics using the testbed described in Section II. The goal of our experiments is to study the behavior of HTTP flow in the face of various network delay and loss characteristics. We consider the high network offered load as a major contributor to the increasing network delay and loss rate and focus on the effect of the network offered load on the Web retrieval latency.

A. Methodology

We use a simple HTTP browser as the Web client and Apache 1.3.4 as the Web server. Both the HTTP requests and responses will go through the virtual link in the *Entrapid*, which simulates a network of specific delay and loss characteristics. The user perceived performance of Web browsing is mainly measured by the retrieval latency of a HTTP request, defined as the period between the time the request is issued and the time the entire response message is received. In our study, we use the transfer time of the underlying TCP[15] connection as an approximation of the retrieval latency. The observed network performance under various network offered load is shown in Figure 3.

Each experiment includes multiple trials under a specific network configuration and Web page size to achieve high stability (i.e., error $< 5\%$) of the results. In each trial, the client retrieves a file (without embedded objects) of a fixed size from the server. The Web client opens up a new TCP connection for the requested file. The actual transfer size on the TCP connection is approximately the file size. We deliberately set the file size to capture the typical transfer size of the underlying TCP connection for Web retrieval, in order to illustrate the typical retrieval latency experienced by the user. We choose the file sizes[1] to be 3KB and 10KB for HTTP/1.0 and 30KB for HTTP/1.1, respectively.

B. Experimental results

We vary the network offered load (normalized by the network bandwidth) in the range of $0\% \sim 100\%$ and measure the resulting retrieval latency. We observe that the effect of the network offered load on the retrieval latency of a file is a combination of that of the packet delay and the loss rate. We decouple the effects of the packet delay and the loss rate by examining their individual effects and the correlation between them.

Figure 4(a) shows that the retrieval latency heavily depends on the network conditions. As the network offered load increases, both the packet delay and the loss rate increase, which results in a rapid growth of the object retrieval latency. Figures 4(b) and (c) show the individual effects of the packet delay and the loss rate on the object retrieval latency, respectively. The latency increases approximately linearly with the increasing of the mean packet delay, but much more drastically with that of the loss rate. As the loss rate increases, the file transfer subjects to more severe TCP retransmission and congestion control to avoid instability. The increasing number of retransmissions and retransmission timer backoffs lead to a superlinear growth of the retrieval latency, which is the total time needed to deliver all the request and response packets.

In addition, we observe that the effect of the offered load on the retrieval latency is approximately a straightforward summation of the individual effects of the packet delay and the loss rate. In other words, for a certain network offered load ld, the resulting retrieval latency $r(ld)$ is approximately the sum of the latency $r_1(pd(ld))$ caused by the packet delay $pd(ld)$ corresponding to the offered load ld and the latency $r_2(lr(ld))$ caused by the loss rate $lr(ld)$ corresponding to the offered load ld. It implies that the effects of the packet delay and the loss rate can be decoupled and studied individually.

C. Modeling TCP performance

Wealth of evidence suggests that TCP connections for HTTP requests are short, often around 10KB and often suffer high packet loss rates in the neighborhood of 5% [5]. We focus on deriving a TCP short connection model[2] to provide a quantitative explanation of the experimental results shown above. Our model is motivated from [5].

C.1 Analytical model

The data transfer can be modeled as an initial connection establishment handshaking, followed by alternating phases of slow start and RTO runs, where an RTO run is a series of successive retransmission timeouts for one packet [5]. The transfer time is $t = t_{handshk} + t_{RTO} + t_{xfer}$, where $t_{handshk}$, t_{RTO}, t_{xfer} are the time spent on handshaking, timeouts, and slow start.

C.1.a Effective packet loss rate p. Assume a TCP receiver that implements delayed acknowledgment sends one ACK for roughly every b ($b = 2$) packets. A data packet appears to be lost in two cases: (i) the data packet gets lost (or overly delayed); (ii) the data packet is delivered successfully, but its ACK (sent as a separate packet or piggybacked) gets lost (or overly delayed). The sender cannot distinguish these two cases and both of them can potentially trigger a retransmission timeout. Suppose each data packet triggers an ACK packet, the *effective packet loss rate*[3] (i.e., the timeout probability) is $p = pl + pl * (1 - pl)$,

[1] In generic HTTP/1.0 [4], the client opens up a new TCP connection to retrieve each object embedded in a Web page, which results in the median and average transfer sizes to be $2 \sim 3$KB and $8 \sim 12$KB, respectively [5]. In HTTP/1.1, a persistent TCP connection can be used to retrieve several objects [7]. Pipelining allows the client to pipeline all the requests for the embedded objects in a page and the server to pipeline all the corresponding responses. The resulting average transfer size of a TCP connection, which can be roughly estimated as the average size of all the data on a web page, becomes $26 \sim 32$KB [12].

[2] Our development and evaluation of the TCP short connection model are primarily based on the TCP specification of Linux 2.0.34.

[3] Due to delayed acknowledgment, a data packet does not necessarily trigger an ACK, hence the computation of the exact effective packet loss rate is very complicated. However, we can use $p = pl + pl * (1 - pl)$ as a reasonable approximation.

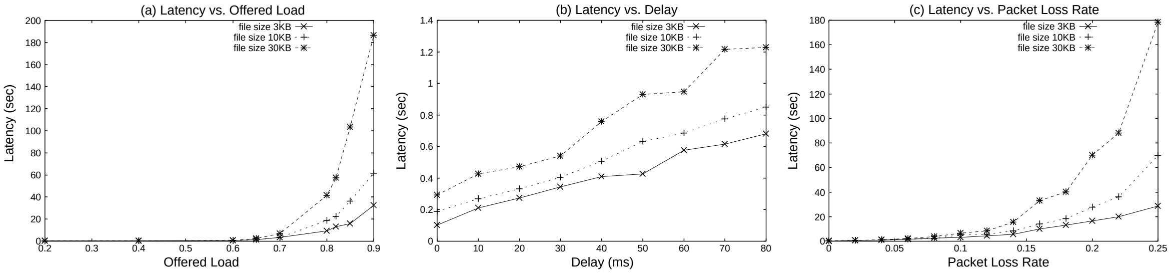

Fig. 4. The Web retrieval latency: (a) retrieval latency vs. offered load, (b) retrieval latency vs. packet delay, (c) retrieval latency vs. packet loss rate.

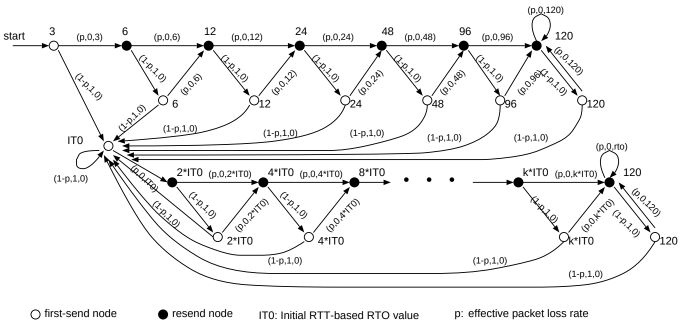

Fig. 5. The evolution of RTO value in the data transfer phase.

where pl is the packet loss rate. We use the effective packet loss rate p throughout the estimation in this paper. Let $data$ be the number of data segments to transfer, l be the number of losses during transferring this amount of data, w be the initial $cwnd$ for the data transfer phase, RTT be the round trip time, and T_0 be the average start RTO value of an RTO run, then the effective loss rate is $p = l/(data + l)$. Thus, $l = p * data/(1 - p)$.

C.1.b Estimating t_{RTO}. The t_{RTO} is the time spent on timeout during the data transfer phase. For the sake of simplicity, we ignore fast retransmission and fast recovery, and simplify delayed acknowledgment as the rule of generating one ACK per two segments. The probability that a particular loss will incur a retransmission timeout can be estimated as follows [5]. If $p = 0$, then $Q(p) = 0$; if $p > 0$, then $Q(p) = min(1, 3/\sqrt{\frac{8}{3bp}})$. Therefore, the number of retransmission timeouts experienced by a flow is $n = l \cdot Q(p)$. We take the loss rate p as the probability that RTO occurs for a send, so the number of RTOs in an RTO run is a geometric distribution with a mean of $1/(1 - p)$, and the number of RTO runs, denoted by u, is $u = \frac{n}{\frac{1}{1-p}} = l \cdot Q(p) \cdot (1 - p)$.

The evolution of RTO value (in Linux 2.0.34) is depicted as a state transaction diagram in Figure 5. Each node stands for a data sending (i.e., transmission or retransmission). The value associated with each node is the RTO value used to set the retransmission timer for that data sending. The start point corresponds to the first transmission of the first packet with the RTO timer set to 3 seconds. Associated with each edge is a tuple (*prob*, num_pkt, *time*), where *prob* is the probability of taking this edge, num_pkt is the number of packets delivered by taking this edge and *time* is the time added to the total timeout time by taking this edge. Each node with the RTO value rto can take one of two edges:

Timeout edge $(p, 0, rto)$. With probability p, the packet fails to be delivered and a timeout occurs. The rto will be added to the total timeout time. Upon a timeout, the RTO value doubles (i.e., retransmission timer exponential backoff). So the value associated with the node pointed to by this edge is $2 \cdot rto$.

Send-success edge (1-p, 1, 0). With probability $1 - p$, the packet is delivered successfully, and the total timeout time remains the same. Two cases need to be distinguished. (i) The packet delivery succeeds on its first attempt. Upon receiving an ACK for this packet, the sender performs RTT estimation and updates the RTO value. Therefore, the RTO value for the end node of this edge should be an RTT-based RTO value IT_0. (ii) The packet delivery succeeds after more than one attempts. According to Karn's algorithm [10], the ACK for this packet does not trigger RTT estimation and RTO update. The RTO value for the end node of this edge is still rto. We determine which case a node is in when it takes a *send-success* edge by examining the edge through which it is reached. If the edge is a *send-success* edge, the node is a first transmission of a packet (*first-send* node depicted as a circle in Figure 5) and it is in Case (i). If the edge is a *timeout* edge, the node is a retransmission (*resend* node depicted as a dot in Figure 5) and it is in Case (ii).

Since the maximum RTO value is clamped down at 120 seconds, the "doubling chain" of RTO value is finite. A typical RTO run starts from a *first-send* node, followed by one or more *timeout* edges and *resend* nodes, and ends with a *send-success* edge and a *first-send* node. T_0 is the RTO value associated with the start node in an RTO run. For simplicity, we assume that $srtt = RTT = 2 * delay$ and $mdev = \frac{srtt}{2}$. Since the minimum RTO value is specified to be 200ms in Linux, the RTT-based RTO value is a constant $max(srtt + 4 * mdev, 200ms) = max(3 * srtt, 200ms) = max(6 * delay, 200ms)$.

However, it is hard to get the analytical form of t_{RTO}. If we can compute the expected duration of a single RTO run t_u, then by using the expected number of RTO runs u, we can approximate t_{RTO} as $u \cdot t_u$. Let T_{0m} denote the first RTO value for an RTO run of data packet m, t_{um} denote the expected duration of this RTO run. Our further analysis on Figure 5 shows that $T_{0m} = f(T_{0(m-1)})$, $t_{um} = E[h(T_{0m})]$, and $t_u = \frac{1}{data} \sum_{m=1}^{data} t_{um}$, where f and h are non-linear functions. So $t_{RTO} = E[g(T_{01}, T_{02}, \cdots, T_{0,data})]$, where g is a complicated non-linear function, and $T_{01}, T_{02}, \cdots, T_{0,data}$ are random variables whose distributions are unknown.

In principle, once we have the evolution graph of the RTO value, given the number of data segments to send, denoted by $data$, we can compute t_{RTO} as follows. Let P denote the set of all paths with the sum of num_pkt of all edges along each path equal to $data$. Each path represents a possible way timeouts occur during the data transfer. So the expected timeout time for the data transfer is

$$t_{RTO} = \sum_{\text{path p} \in \text{P}} \text{the probability of taking } p * \text{the timeout time spent on } p$$

$$= \sum_{\text{path p} \in \text{P}} \left(\prod \textit{prob} \text{ of edges along } p\right) * \left(\sum \textit{time} \text{ of edges along } p\right)$$

In effect, this is equivalent to the simulation approach, which is usually used when we are interested in computing $\theta = E[g(X_1, X_2, \cdots, X_n)]$, where g is some specified function, but it is not possible to analytically compute θ. Using this approach, we generate sufficiently many combinations of $X_1, X_2, \cdots, X_n$, compute the corresponding $g(X_1, X_2, \cdots, X_n)$, and use the average of these function values as an estimate of θ [16]. In our case, we don't have analytical form of the function g, but we can compute the function value using Figure 5. By simulating a whole run of Figure 5 sufficiently many time (where a run starts with sending the first packet and ends when number of $data$ packets are sent successfully), and measuring the RTO time in each run, we effectively generate sufficiently many combinations of T_{01}, T_{02}, $\cdots$, $T_{0,data}$, and compute $g(T_{01}, T_{02}, \cdots, T_{0,data})$. We can use the average RTO time per run as a reasonable estimate for t_{RTO}. By using this simple and efficient approach, we are able to avoid the complexity of real TCP simulator, and work around the intricacy of deriving a pure analytical model as well.

C.1.c Estimating $t_{handshk}$. We take timeouts during handshaking into account and model handshaking time as a special case of data transfer phase. Figure 6 shows the 3-way handshaking on the TCP connection for a Web retrieval session. The case when no packet is lost during this phase is depicted in Figure 6(a). The Web client initiates the connection establishment by sending a SYN packet to the Web server. On receiving the SYN, the server sends its own SYN with an ACK for the client's SYN piggybacked (i.e., the SYN/ACK packet). When the SYN/ACK reaches the client, it triggers a separate ACK packet from the client to the server. At this point, the client enters into ESTABLISHED state. It sends out the first data packet (the HTTP request) to the server with the ACK for the server's SYN piggybacked. Only when the server receives the ACK for its SYN either piggybacked or sent as a separate packet, will it enter into ESTABLISHED state and start to send data packets. In Linux 2.0.34, the first SYN and SYN/ACK packets are both sent with the retransmission timer set to 3 seconds, while the ACK packet is sent without any retransmission timer set and sent only once.

Retransmission (and possible backoff) happens when there is packet loss. Whenever the SYN packet or the SYN/ACK packet gets lost, it will be retransmitted and the retransmission timer keeps backing off until it is finally delivered (Figure 6(b) and (c)). If the last ACK packet gets lost, the client counts on the first data packet sent to the server to piggyback the ACK. As usual, the data packet may be retransmitted several times before it reaches the server (Figure 6(d)). Thus, we estimate $t_{handshk}$ as $t_{handshk} = ths_{xfer} + ths_{RTO}$, where ths_{xfer} is the transfer time of the packets during handshaking, and $ths_{xfer} \approx 1.5 * RTT$ and ths_{RTO} is the time spent on timeout during this phase.

We can view the 3-way handshaking phase as the concatenation of two special data transfer phases whose t_{RTO}s can be efficiently estimated by simulation. The first phase is a reliable transfer of the client's SYN packet, including the (re)transmissions of SYN and SYN/ACK. Let ths_{RTO1} denote the timeout time in the first phase. It can be computed as t_{RTO} in a data transfer phase for one data packet with the effective loss rate $p = pl + pl * (1 - pl)$. The second phase can be one of two cases: (1) with probability $(1 - pl)$, it is a successful transmission of the ACK packet, with the timeout time equal to 0; (2) with probability pl, it is a loss of the ACK packet, followed by the (re)transmissions of the first data packet of the client until it successfully arrives at the server. Let ths_{RTO2} denote the timeout time in this case. It can be computed as t_{RTO} in a data transfer phase for one data packet with the effective loss rate $p = pl$ (since only the first data packet not its ACK need to be successfully delivered). Finally, the timeout time in the handshaking phase is $ths_{RTO} = ths_{RTO1} + pl * ths_{RTO2}$.

C.1.d Estimating t_{xfer}. In our model, t_{xfer} is estimated in a way similar to that proposed in [5] except the initial $cwnd$ of the data transfer phase. Since the slow start phase are alternating with RTO runs, the number of slow start phases is $v = u + 1$. The average data sent per phase is $e = \frac{data+l}{v}$. Consider the progress of TCP in slow start mode in terms of *rounds* whose durations are equal to one RTT. Let $cwnd_i$ be the $cwnd$ of the sender at the beginning of round i. $cwnd_i$ is a geometric series with ratio $r = 1 + \frac{1}{b}$.

In Linux 2.0.34, the initial $cwnd$ depends on the result of the slow start and congestion avoidance during handshaking. When the TCP connection is initialized, $cwnd$ is one MSS and $ssthresh$ is some very large number. With probability $(1 - p)^2$, no loss happens to SYN and SYN/ACK packets during handshaking, $cwnd$ becomes two MSS and $ssthresh$ remains the same for both the client and the server when they start sending data packets. Once either SYN or SYN/ACK ever gets lost, $cwnd$ becomes one MSS and $ssthresh$ becomes zero upon the retransmission timeout and $cwnd$ remains one MSS till the end of handshaking. Thus the initial $cwnd$ size w (measured by the unit of MSS) is $w = (1 - p)^2 \cdot 2 + (1 - (1 - p)^2) \cdot 1$, and t_{xfer} can be estimated as $t_{xfer} = v \cdot \log_r (e(r - 1)/w + 1) \cdot RTT$.

C.2 Performance

We examine accuracy the new model and compare with the model in [5] (referred as Model C) and measurement results. Due to the space limitation, we only provide an outline of the results, interested readers can refer to [17] for details.

Figure 7 shows the estimations of the transfer time of a fixed-size file for varying network offered load. The curve *measure* is the measured latency curve as in Figure 4. The curves *Model C low* and *Model C high* are both latency estimated using Model C with same $RTT = 2 * delay$, number of data packets $data = \frac{\text{file size + HTTP header size}}{1460}$, initial $cwnd$ $w = 1$, measured T_0, but different effective packet loss rates, i.e., $p = pl$ and $p = pl + pl * (1 - pl)$, respectively[4] . The curve *new model* is the latency estimated by our new model. We observe: (i) the absolute error of the estimated latency when the offered load $< 60\%$ is relatively small for both models; (ii) the new model is more accurate than Model C in all the cases; (iii) Model C underestimates the latency due to the underestimation of the effective loss rate; (iv) Model C with corrected effective loss rate overestimates due to the inaccurate modeling of the clamp-down timer

[4] *Model C low* actually refer to the model in [5]. We also plotted the results of Model C by using the correct effective loss rate model (referred as *Model C high*) to examine its effect on estimating HTTP retrieval latency.

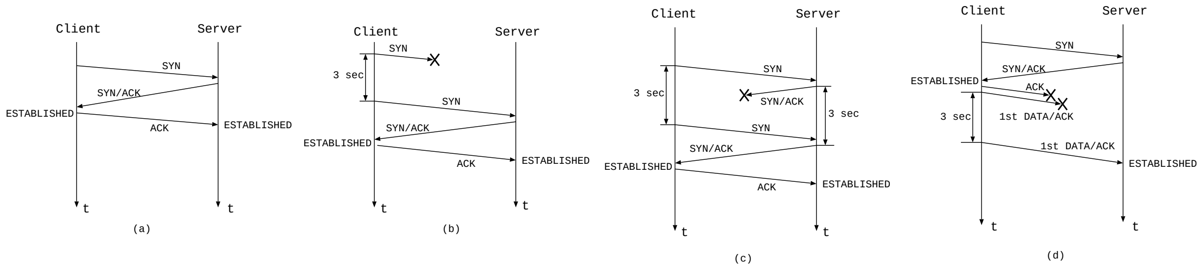

Fig. 6. Four cases of the 3-way handshaking: (a) no packet loss, (b) the SYN packet is lost, (c) the SYN/ACK packet is lost, (d) the last ACK is lost.

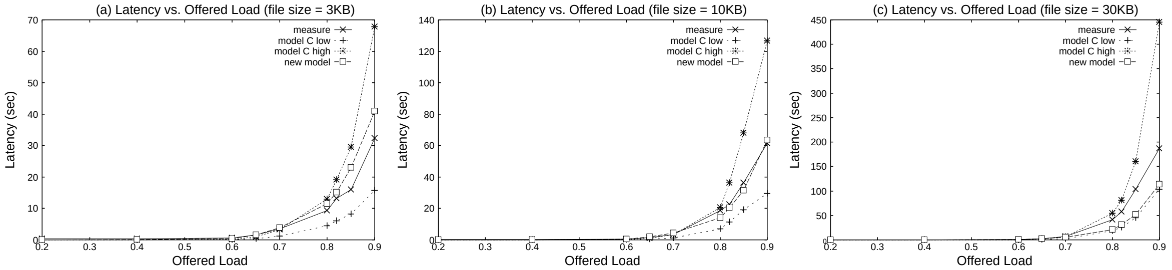

Fig. 7. Offered load vs. retrieval latency: (a) file size = 3KB, (b) file size = 10KB, (c) file size = 30KB.

[5]. We also examine the file transfer time when the loss rate = 0 and delay varies, and delay = 0 and loss rate varies. We observe that the new model over-performs Model C in all the cases.

In summary, we observe that the new model provides an accurate estimation of the retrieval latency and out-performs Model C [5] in all the cases. Comparing to the measurement results, when loss rate is high (i.e., $> 10\%$), the new model overestimates for small transfer sizes (i.e., $\sim$ 3KB) and underestimates for large transfer sizes (i.e., $\sim$ 30KB), but fits well with moderate transfer sizes (i.e., $\sim$ 10KB). We explain the reason in [17].

IV. Related work

Previous work on study network performance and its effect on applications relies on purly simulation (e.g., NS simulator [14]) or real measurements. Our testbed is a combination of simulation and emulation, which is novel in this field to our best knowledge. No similar approach has been proposed in previous work. On modeling TCP performance, most of the existing TCP performance models analyze the steady-state throughput of long bulk-transfer TCP connection with or without packet loss[6] [9] [11] [13]. The only model of TCP short connection performance is proposed in [5]. We have shown that our new model is more accurate than the one proposed in [5].

V. Conclusions

In this paper, we designed a novel composite testbed which is a combination of simulation and emulation. The former gives us manageability and efficiency by allowing us to impose changing network characteristics of various network environments, such as LAN, WAN, and wireless networks directly and easily, without sacrificing accuracy. The latter gives us great flexibility by allowing us to hook various kinds of applications to the simulated network as easily as in a real network environment.

We use this testbed to study the end-to-end performance of various user applications. We focus on estimating the Web retrieval latency. We show that the retrieval latency increases linearly with the packet delay but nonlinearly with the packet loss rate. To provide a quantitative explanation, and to estimate the latency of Web retrieval given the requested file size, we develop a TCP short connection performance model which estimates the transfer time of the requested file size on the underlying TCP connection. Our experimental results show that the new model is more accurate than the existing model [5].

References

[1] Y. Bernet et al., A framework for differentiated services, Internet Draft *<draft-ietf-diffserv-framework-02.txt>*, Feb. 1999.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, An architecture for differentiated services, Internet RFC 2475, Dec. 1998.

[3] R. Braden, D. Clark and S. Shenker, Integrated services in the Internet architecture: an overview, Internet RFC 1633, June 1994.

[4] T. Berners-Lee, R. Fielding, and H. Frystyk, Hypertext transfer protocol - HTTP/1.0, Internet RFC 1945, May 1996.

[5] N. Cardwell, S. Savage, and T. Anderson, Modeling TCP Latency, Proc. of Infocom'00, 2000.

[6] S. Floyd, Connections with multiple congested gateways in packet-switched networks, part 1: one-way traffic, ACM Computer Communications Review, 21(5), Oct. 1991.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, Hypertext transfer protocol - HTTP/1.1, Internet RFC 2068, Jan. 1997.

[8] X. Huang, R. Sharma, and S. Keshav, The ENTRAPID protocol development environment, Proc. of Infocom'99, Mar. 1999.

[9] A. Kumar, Comparative performance analysis of versions of TCP in a local network with a lossy link, IEEE/ACM Trans. on Networking, 6(4), Aug. 1998.

[10] P. Karn and C. Partridge, Improving round-trip time estimates in reliable transport protocols, Proc. of ACM Sigcomm'87, pp. 2-7, August 1987.

[11] T. V. Lakshman and U. Madhow, The performance of TCP/IP for networks with high bandwidth-delay products and random loss, IEEE/ACM Trans. on Networking, June 1997.

[12] B. A. Mah, An empirical model of HTTP network traffic, Proc. of Infocom'97, Apr. 1997.

[13] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, The macroscopic behavior of the TCP congestion avoidance algorithm, ACM Computer Communications Review, 27(3):67-82, July 1997.

[14] UCB/LBNL/VINT Network Simulator - ns (version 2), *http://www.isi.edu/nsnam/ns/*.

[15] J. Postel, Transmission control protocol, Internet RFC 793, Sept. 1981.

[16] S. M. Ross, Introduction to Probability Models (6th Edition), pp598-599, Academic Press, 1997.

[17] Y. Zhang, J. Wang, and S. Keshav, The implication of network performance on service quality, Technical Report TR99-1754, Department of Computer Science, Cornell University, July 1999.