IEICE TRANS. COMMUN., VOL.E89–B, NO.9 SEPTEMBER 2006

PAPER Special Section on Networking Technologies for Overlay Networks

TCP-Westwood Low-Priority for Overlay QoS Mechanism

Hideyuki SHIMONISHI[†a)], **Takayuki HAMA**[†], ***Members***, **M.Y. SANADIDI**[††], **Mario GERLA**[††], ***Nonmembers***, ***and*** **Tutomu MURASE**[†], ***Member***

SUMMARY An overlay traffic control is a way to provide flexible and deployable QoS mechanisms over existing networks, such as the Internet. While most of QoS mechanisms proposed so far require router supports, overlay QoS mechanisms rely on traffic control at transport layer without modifying existing routers in the network. Thus, traffic control algorithms, which are implemented at traffic sources or PEPs (Performance Enhancement Proxies), play a key role in an overlay QoS mechanism. In this paper, we propose an end-to-end prioritization scheme using TCP-Westwood Low-Priority (TCPW-LP), a low-priority traffic control scheme that maximizes the utilization of residual capacity without intrusion on coexisting foreground flows. Simulation and Internet measurement results show that TCPW-LP appropriately provides end-to-end low-priority service without any router supports. Under a wide range of buffer capacity and link error losses, TCPW-LP appropriately defers to foreground flows and better utilizes the residual capacity than other proposed priority schemes or even TCP Reno.

key words: *overlay, QoS, priority, TCP, TCP-westwood*

1. Introduction

The diverse service requirements of emerging Internet applications justify the need for differentiated service quality, even within best effort classes supported mainly by TCP. For example, web-browsing or Internet chatting require quick transfers to satisfy response time requirements; whereas content pre-fetching/distribution in Content Distribution Networks (CDNs) involve relatively large bulk transfers, but no strict completion time requirements.

These coexisting application classes with their diverse performance requirements can be well supported by a "foreground/background" priority scheme. One web example is content pre-fetching where a web cache attempts to improve its hit rate by pre-fetching objects. Without the support of priority discrimination, while frequent pre-fetching improves the response time of future web page requests, it also causes performance degradation of foreground requests since these different types of traffic will equally share the bottleneck capacity. On the other hand, object pre-fetching using a low-priority service can avoid any impacts on foreground requests, and still improve the response time of future requests. Generally, any kinds of servers with a rich mix of diverse applications can potentially take advantage of the priority service.

For such scenarios consisting of different service priorities, Differentiated Services (DiffServ) [1] and similar approaches have been studied extensively. Their deployment, however, has been slow because of large diversity of router architectures. In existing networks, a huge number of routers are operating and it would be so hard to modify all these routers to provide a new service. In addition, for end-to-end service scenarios, it would be even harder to negotiate a QoS framework over multiple carriers and ISPs.

For that reason, alternative end-to-end schemes, requiring no support from the network routers, have received increased attention. An overlay QoS mechanism relies on appropriate enhancements to transport protocols at middle-boxes, or also called PEPs (Performance Enhancement Proxies), which relay end-to-end sessions using the enhanced transport protocols having specific behaviors for required QoS. Therefore, traffic control algorithms play a key role in overlay QoS mechanism and determine efficiency and service level of the network.

In this paper, we present TCP Westwood Low-Priority (TCPW-LP) [12], an end-to-end transport protocol aiming to realize two-class service prioritization. Since TCP Reno and its variants are the dominant transport protocols carrying the majority of data traffic today, providing low-priority version of TCP that defers to TCP-Reno flows is a worthwhile goal. The objective of TCPW-LP is to control low-priority flows to ensure that such flows are: (1) non-intrusive to coexisting foreground traffic carried by legacy TCP or UDP, (2) capable of fully utilizing the bandwidth left unused by foreground traffic, and (3) capable of fairly sharing with other low-priority flows whatever bandwidth they are eligible for. Existing low-priority TCPs have been proven to achieve (1), but not effective in achieving (2) and (3).

TCPW-LP is based on TCP-Westwood (TCPW) [6]–[8]. TCPW is an extension of TCP-Reno. In TCPW-LP, the window reductions are triggered not only by packet losses but also by incipient congestion so as to realize low-priority transfer. To this end, a TCPW-LP flow estimates the amount of its packets queued in the path routers as a backlog, and reduces its congestion window if the backlog exceeds a threshold. We call this window reduction mechanism Early Window Reduction. Upon the congestion events, it decreases the congestion window according to the Eligible Rate Estimation (ERE), instead of halving the window as in Reno,

Manuscript received January 13, 2006.
Manuscript revised April 6, 2006.

[†]The authors are with System Platforms Laboratory, NEC Corporation, Kawasaki-shi, 211-8555 Japan.
[††]The authors are with UCLA Computer Science Department, 3732F/3531F Boelter Hall, Los Angeles, CA 90095-1596, USA.
a) E-mail: h-shimonishi@cd.jp.nec.com

DOI: 10.1093/ietcom/e89–b.9.2414

to avoid excessive window reduction and thus improve efficiency.

An appropriate threshold adjustment is a key to realize "efficient" low-priority service. In order to achieve non-intrusiveness, the threshold should be sensitive to congestion caused by coexisting foreground flows. On the other hand, to achieve high efficiency, the threshold should not be sensitive to congestion caused by low-priority flows. Thus, we propose a new congestion metric we call Foreground Traffic Ratio. When TCPW-LP recognizes that the congestion is due to foreground flows, it set the threshold very small to give the bandwidth to foreground flows, otherwise it set the threshold larger to efficiently utilize the residual capacity.

The rest of the paper is organized as follows. Section 2 discusses an overlay QoS mechanism and Sect. 3 describes the related low-priority TCPs. The detail of the TCPW-LP mechanisms is provided in Sect. 4. In Sects. 5 and 6, simulation results and Internet measurement results are presented, respectively. Concluding remarks are provided in Sect. 7.

2. Overlay QoS Mechanism

Existing QoS mechanisms including Differentiated Services (DiffServ) [1] and other proposals rely on router supports for traffic control at a bottleneck link. On the other hand, instead of deterministic traffic control at every router on a traffic path, an overlay QoS mechanism relies on appropriate enhancements to transport protocols at PEPs that relays end-to-end sessions. For example, as shown in Fig. 1, user initiated TCP sessions are terminated at TCP PEPs, and passed to the connecting TCP sessions having specific behaviors for required QoS services.

In overlay QoS mechanisms, traffic control algorithm plays a key role and determines efficiency and service levels of the network. A traffic control algorithm at PEPs needs to explore the path characteristics and appropriately tunes its control behavior in such a way that the behavior satisfies the QoS demands. As shown in Fig. 2, various traffic control algorithms would be used for various QoS demands. For example, Mul-TCP [2] is used to realize M times differentiated TCP throughputs. Also, for foreground/background priority services, TCP-Nice [3] and TCP-LP (Low-Priority) [4] are used.

Therefore, in this paper, we focus on traffic control algorithms for overlay QoS mechanisms, specifically, a traffic control algorithm for foreground/background priority services. TCP Reno and its variants are the dominant transport protocols carrying the majority of data traffic today; therefore, introducing and studying the behavior of prioritization schemes within the congestion control mechanisms of TCP is a worthwhile goal. Thus, assuming that legacy TCP carries foreground traffic, we present a low-priority transport protocol that defers to TCP-Reno flows but can efficiently utilize residual capacity.

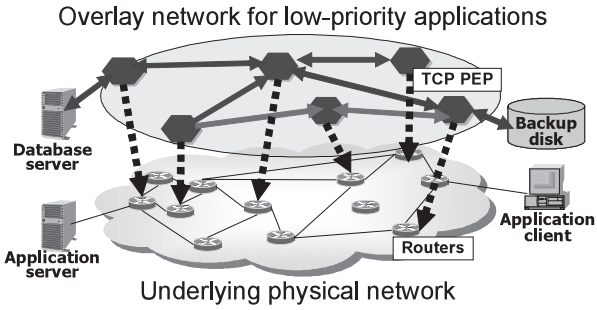


Fig. 1 Overlay network for service differentiations.

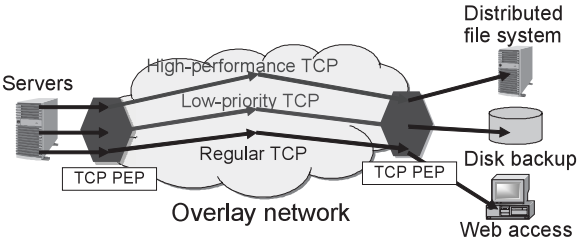


Fig. 2 Multiple QoS classes via transport differentiation.

3. Low-Priority TCPs

TCP-Nice and TCP-LP were proposed to provide two-class service prioritization. These protocols attempt to detect incipient congestion and significantly reduce their congestion windows before coexisting foreground flows react to the congestion. For example, TCP-LP uses queuing delay to detect incipient congestion. It sets a threshold between minimum queuing delay and maximum queuing delay, and whenever it detects congestion by comparing current queuing delay and the threshold, it halts the increase of its congestion window, or reduces the window to 1. Such queuing delay can be measured by the difference between current RTT (Round Trip Time) and its minimum value.

Although these schemes result in low-priority flows that are non-intrusive to foreground flows, they are often inefficient in utilizing the eligible capacity. Since the queuing delay does not indicate whether the congestion is caused by foreground flows or not, low-priority flows react to incipient congestion even if such congestion is caused by the low-priority flows themselves. In addition, it is difficult to set a proper queuing delay threshold because buffer sizes vary from one router to the other; and further, the bottleneck router can change frequently.

4. TCP-Westwood Low-Priority (TCPW-LP)

4.1 TCP-Westwood (TCPW)

TCPW [6]–[8] is a sender-side-only modification of TCP congestion control. TCPW follows an "additive increase" of its congestion window in congestion avoidance. The difference is that, upon packet losses, it reduces the congestion

window according to Eligible Rate Estimation (ERE), instead of halving the window as in Reno.

During congestion avoidance, a sender calculates an ERE based on information carried in the ACKs and the rate at which the ACKs are received. Detailed estimation algorithms are not provided here since they have been reported and analyzed in [6]–[8]. Upon a packet loss, due to either buffer overflow or random errors, the sender uses the ERE to properly reduce the congestion window and the slow start threshold, in essence as follows:

$$ssthresh = ERE \cdot RTT_{min},$$
$$cwin = ssthresh, \ if \ cwin > ssthresh,$$

where $ssthresh$, $cwin$, and RTT_{min} are the TCP slow start threshold, congestion window size, and minimum RTT measurement. Note that $ERE \cdot RTT_{min}$ is the amount of packets in a virtual pipe with length equal to RTT_{min} and diameter equal to ERE.

4.2 Early Window Reduction in TCPW-LP

In TCPW-LP, an Early Window Reduction (EWR) mechanism is added to realize low-priority transfer. In response to an incipient congestion, EWR limits the backlog allowance of a flow, by reducing the congestion window even before packet losses actually occur. This idea is similar to the window control mechanism in TCP Vegas [5] that limits the number of backlogged packets in the bottleneck queue.

The flow backlog over the path, also called virtual queue length, is given by:

$$Virtual\ queue\ length\ = cwin - ERE \cdot RTT_{min},$$

where $cwin$ is the total amount of outstanding packets in the path and $ERE \cdot RTT_{min}$ is the amount of packets in a virtual pipe with length equal to RTT_{min} and diameter equal to ERE. As in [7], TCPW-LP uses achieved rate of a flow as its ERE. The achieved rate, which can be derived by $cwin/RTT$, is the rate at which packets are delivered to the destination.

As a reaction to incipient congestion, TCPW-LP congestion window is reduced when the virtual queue length exceeds a threshold. Unlike TCP-Vegas, which tries to fix the congestion window within an ideal range, TCPW-LP always changes congestion window through additive increase and EWR. As described later, this behavior is important to infer the existence of foreground flows and dynamically adjust the EWR threshold.

4.3 Dynamic Threshold Adjustment in TCPW-LP

An appropriate threshold adjustment is key to realize "efficient" low-priority service because the threshold value impacts relative throughput (i.e. the larger the threshold, the larger the flow throughput [9]). In order to achieve non-intrusiveness, the threshold should be sensitive to congestion caused by coexisting foreground flows. On the other hand, to achieve high efficiency, the threshold should not be sensitive to congestion caused by low-priority flows.

Therefore, in TCPW-LP, the virtual queue threshold Q_{th} is dynamically adjusted based on both congestion level and its cause. Namely, Q_{th} is set smaller when the congestion level is high and the congestion is caused by coexisting foreground flows.

(a) Threshold adjustment via congestion estimation
TCPW-LP adjusts the threshold according to the congestion level by estimating a queuing delay value D_{loss} at which packet losses are expected to occur. The value is calculated as an exponential average of queuing delay just before packet losses. Q_{th} is set higher when current queuing delay D_{curr} is close to 0, which indicate underutilization of the path. On the other hand, Q_{th} is set close to 0 when the path is being congested and the current queuing delay is close to D_{loss}.

(b) Threshold adjustment via foreground traffic estimation
To improve efficiency and non-intrusiveness tradeoff, we introduce a new congestion metric we call Foreground Traffic Ratio, which is estimated by analyzing round trip time behavior. The ratio equals 1 when all queued packets belong to foreground flows, and becomes smaller when few packets are from foreground flows.

The foreground traffic ratio is estimated by the ratio of temporal minimum queuing delay, D_{min}, to temporal maximum queuing delay, D_{max}. D_{min} and D_{max} are calculated as an exponential average of maximum and minimum queuing delays measured between two consecutive early window reductions, as shown in Fig. 3. Then the ratio is set equal to D_{min}/D_{max}.

An intuitive understanding of this ratio is provided in the following. D_{max} represents the amount of bottleneck backlog contributed by all kinds of flows that coexist on the path. On the other hand, the difference between D_{max} and D_{min} represents the temporal fluctuations of measured RTT and it is mainly contributed by frequent EWR behavior of TCPW-LP flows. Therefore, when there are small number of TCPW-LP flows, foreground flows increases D_{max} while TCPW-LP flows yield relatively small RTT fluctuations; thus, $(D_{max} - D_{min})/D_{max}$ becomes small and the foreground traffic ratio approaches to 1. On the other hand, if there are only TCPW-LP flows, the queuing delay mainly consists of the fluctuation and thus the ratio becomes 0.

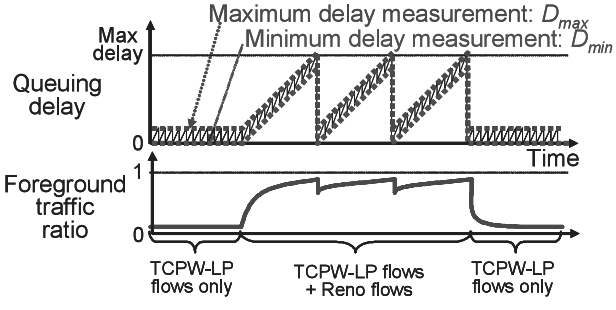


Fig. 3 Estimation of foreground traffic.

Thus, Q_{th} is set smaller when the congestion is mainly contributed by foreground flows and the ratio is close to 1, otherwise it is set larger.

Therefore, the threshold Q_{th} is set to:

$$Q_{th} = M \cdot (1 - D_{curr}/D_{loss}) \cdot (1 - D_{min}/D_{max}),$$

where M is an upper bound on TCPW-LP backlogged packets. As in Vegas, we set M equal to 3.

5. Simulation Experiments

5.1 Simulation Settings

We consider a dumb-bell topology in which many flows share a single bottleneck link, as shown Fig. 4. Unless otherwise mentioned, the bandwidth of the bottleneck link and access links are 10 Mbps and 100 Mbps, respectively. The end-to-end round trip propagation delay is 74 msec, unless otherwise noted. The routers employ single FIFO queuing with drop-tail discipline. The default value for the buffer capacity is 63 packets, which is equal to the bandwidth delay product in this setting.

We use TCP-Reno for foreground flows, and in some experiments, periodic on-off UDP traffic is used as non-responsive foreground traffic. TCPW-LP is used for low-priority flows. For purposes of comparison, we also tested TCP-LP, whose simulation code and default parameter setting were obtained from the web site [11].

5.2 Coexistence with Foreground Traffic

5.2.1 Basic Congestion Window Behavior

Figure 5 illustrates the transient behavior of TCPW-LP flows when foreground Reno flows come and go. In this figure, the average window sizes of Reno flows and TCPW-LP flows are shown. The minimum queuing delay, maximum queuing delay, and its associated early window reduction threshold in this scenario are shown in Fig. 6.

In this experiment, 3 TCPW-LP flows start at 0 sec and their congestion windows stay around the fair share value. In this situation, due to the EWR behaviors, the EWR threshold is set as high as about 2.3 packets and TCPW-LP flows obtain high link utilization.

As soon as 3 Reno flows join in after 30 sec, TCPW-LP flows reduce their congestion windows to almost 0 so that Reno flows can fully utilize the link capacity. In this situation, both minimum and maximum queuing delays grow and thus the threshold is around 0.3 packets, and thus, TCPW-LP gets very small bandwidth.

Then, TCPW-LP flows stop at 40 sec and restart at 50 sec. TCPW-LP flows receive little bandwidth at first, but subsequently they quickly manage to use the full link bandwidth soon after the Reno flows leave.

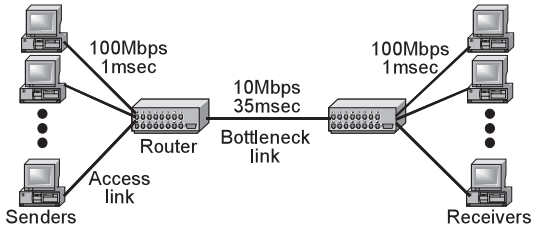


Fig. 4 Network topology (Simulation).

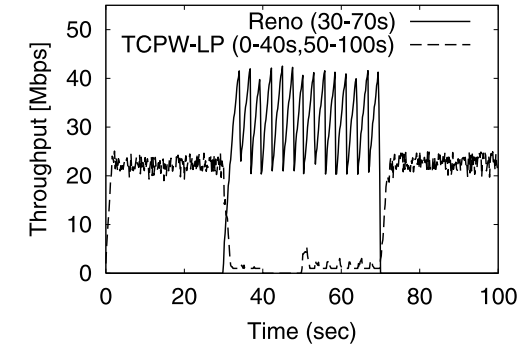


Fig. 5 Congestion window behaviors of coexisting 3 TCPW-LP flows and 3 Reno flows.

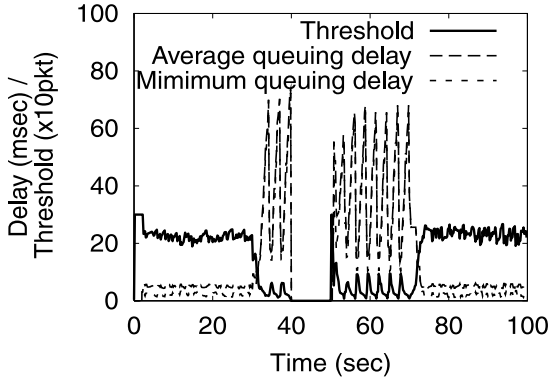


Fig. 6 Queuing delay and early window reduction threshold.

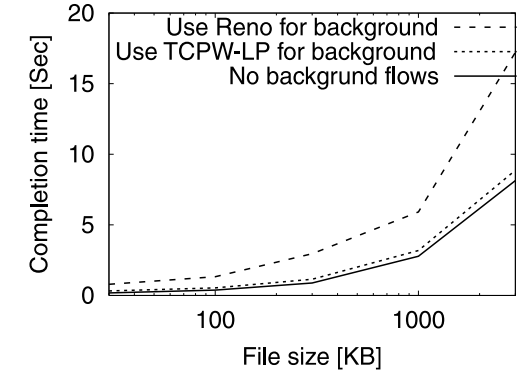


Fig. 7 Completion time of foreground file transfers.

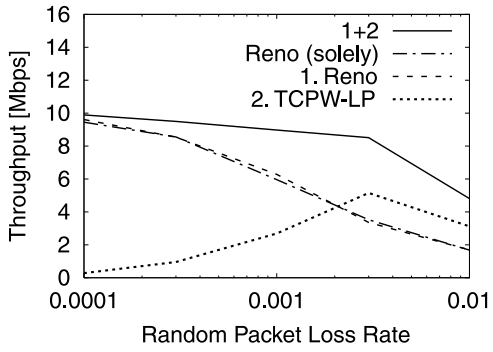


Fig. 8 Throughput of coexisting TCPW-LP and Reno flows vs. packet loss rate due to error.

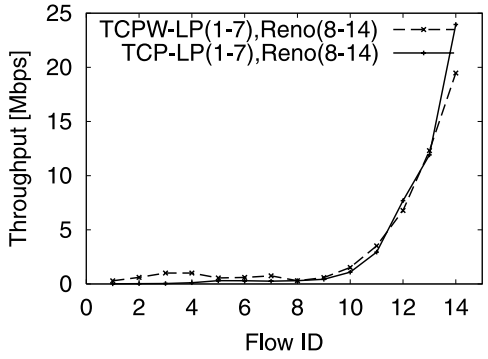


Fig. 9 Throughput of coexisting TCPW-LP flows (1-7) and Reno flows (8-14) with different propagation delays (74, 37, 19, 9, 4, 2, and 1 msec).

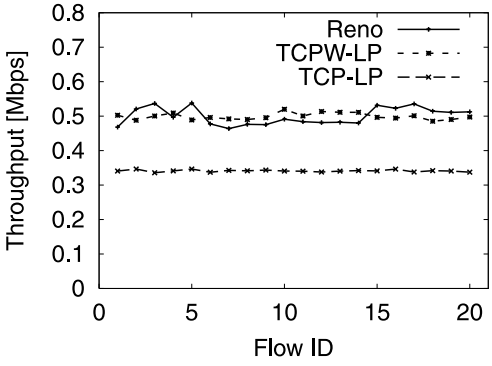


Fig. 10 Throughput of 20 individual flows of Reno, TCPW-LP, or TCP-LP.

5.2.2 Completion Time Evaluation Using Finite FTP Traffic

In Fig. 7, completion times of a finite foreground file transfer are shown in the presence of a background flow. The completion time here is defined as a time interval between when first packet of a file is sent out at the sender and when the arrival of the last packet is acknowledged. The foreground file sizes range from 3 KB to 3 MB. The background flow file size is infinite and uses either Reno or TCPW-LP, for comparison.

This figure shows that a low-priority flow using TCPW-LP hardly damages the completion time of foreground file transfers because of the EWR behavior and the small threshold setting. On the other hand, the completion time almost doubles when the background flow uses Reno because these flows try to share the bandwidth equally.

5.2.3 Effects of Packet Losses Due to Errors

In Fig. 8, the throughput of one Reno flow co-existing with one TCPW-LP flow is shown for different random packet loss rates. The maximum throughput of a lone Reno flow without low-priority flows is also shown.

As shown in the figure, the throughput of a Reno flow decreases as the loss rate increases. Accordingly, the throughput of TCPW-LP flow increases to utilize the unused bandwidth, hardly damaging the Reno flow.

Indeed, the TCPW-LP flow obtains larger bandwidth than the Reno flow when the Reno flow is inherently unable to utilize the bandwidth. In this situation, the Reno flow halves its congestion window as a reaction to random packet losses, whereas the TCPW-LP flow optimizes the window reduction using eligible rate estimation of TCP-Westwood.

5.2.4 Effect of Different Propagation Delays

Figure 9 shows throughputs of coexisting 7 Reno flows and 7 TCPW-LP flows with different propagation delays. This figure also shows another set of throughputs using TCP-LP instead of TCPW-LP. The propagation delay ranges from 74 msec, 37 msec, 19 msec, 9 msec, 4 msec, 2 msec, to 1 msec for flows 1 through 7 (and from 8 to 14). In this figure the bottleneck link bandwidth is set at 50 Mbps to clearly evaluate the throughput difference with this large range of congestion window sizes.

It is confirmed that all TCP-LP and TCPW-LP flows have very small bandwidth compared to Reno flows. Although TCPW-LP flows obtain a bit more bandwidth, the total throughput of all TCPW-LP flows is just 10% of that of Reno flows.

5.3 Efficiency in Bandwidth Utilization

5.3.1 Throughput

In Fig. 10, throughputs of 20 coexisting flows are shown for Reno, TCP-LP, and TCPW-LP. Both Reno and TCPW-LP can fully utilize the link capacity; the total throughputs of 20 Reno flows and 20 TCPW-LP flows are 9.99 Mbps and 9.98 Mbps, respectively. On the other hand, 20 TCP-LP flows just utilize 68% of the link capacity due to its incipient congestion detection and significant window reduction.

5.3.2 Effect of Error Losses

Figure 11 shows throughput of a lone Reno, TCPW-LP, or TCP-LP flow for different random packet loss rates. As is

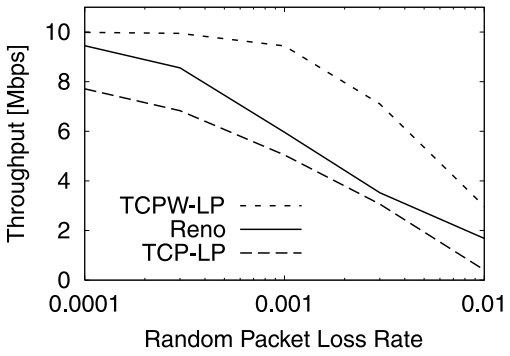


Fig. 11 Single flow throughput vs. packet loss rate due to errors.

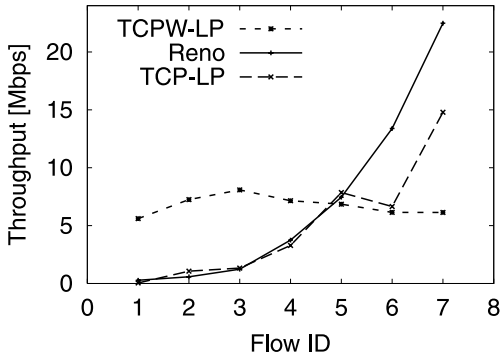


Fig. 12 Throughput of individual flows with different propagation delays (74, 37, 19, 9, 4, 2, and 1 msec).

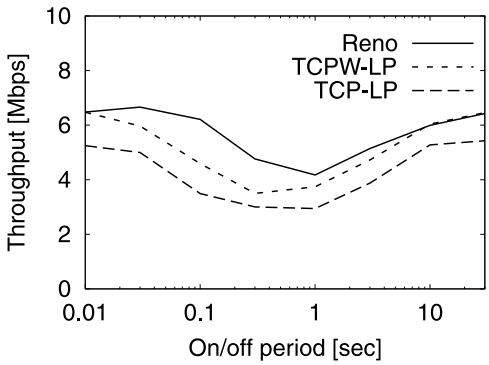


Fig. 13 Throughput of TCP flows coexisting with on/off UDP flows.

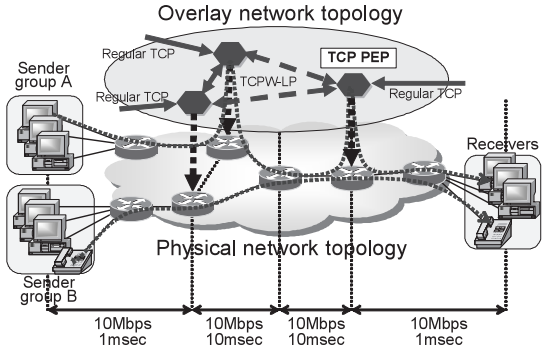


Fig. 14 Overlay network topology (Simulation).

known about Reno, its throughput decreases as the loss ratio increases. While the TCP-LP flow has the lowest throughput because of its sensitive reaction to the congestion, the TCPW-LP flow achieves the highest throughput. For example, the TCPW-LP flow acquires more than 6 times larger throughput than a TCP-LP flow at 1% of random losses.

5.3.3 Effect of Different Propagation Delays

Figure 12 shows throughputs of 7 TCPW-LP flows with different propagation delays, which vary from 1 ms to 2 ms, 4 ms, 9 ms, 19 ms, 37 ms, and 74 ms, as well as throughputs of 7 Reno and 7 TCP-LP flows.

While Reno and TCP-LP flows with shorter propagation delays have higher throughputs over the ones with longer propagation delays, the throughput of TCPW-LP flows are not really correlated to the propagation delays and even the longest TCPW-LP flow can obtain almost fair share of the bottleneck link bandwidth. For example, the longest Reno, TCP-LP, and TCPW-LP flow having 74 msec propagation delay obtains 5.6 Mbps, 0.28 Mbps, and 0.04 Mbps, respectively. The total throughput of 7 TCPW-LP flows is more than 94% of the link capacity, whereas that of TCP-LP flows is just 70% of the link capacity.

5.3.4 Coexistence with Non-responsive UDP Traffic

The last experiment evaluates the link utilization by a Reno, TCP-LP, or TCPW-LP flow when coexisting with non-responsive UDP flows. On-off UDP traffic with equal lengths of on-periods and off-periods is used in this experiment. The periods are varied from 0.003 sec to 30 sec. Since the available bandwidth during on- and off-periods is 3.3 Mbps and 10 Mbps, respectively, the average residual bandwidth is 6.7 Mbps.

In Fig. 13, average throughput of a Reno, TCPW-LP, or TCP-LP flow is shown. Although all flows, except TCP-LP, get almost full link bandwidth when the period is smaller than 0.003 sec or larger than 10 sec, they underutilize the link when the length of the period is around 1 sec. In this region, a Reno flow has the largest throughput because it can hold a large number of backlogged packets at the bottleneck queue. Still, a TCPW-LP flow can utilize the residual capacity more efficiently than a TCP-LP flow.

5.4 Simulation Experiments for Overlay QoS

Performance of TCPW-LP is evaluated using overlay network configuration shown in Fig. 14. As shown in this figure, each one of TCP PEPs is connected to either sender group A, sender group B, or receivers. TCP senders in groups A and B on the left side are connected to the receivers on the right side. All TCP senders and receivers use TCP-Reno, while they are relayed at TCP PEPs using TCW-LP to provide low-priority service. Also, one UDP/RTP session are established from sender group A to the receiver.

Sending rate of the RTP/UDP session is 128 kbps including header overheads. Propagation delays and link capacities are shown in Fig. 14.

In this scenario, we compared the overlay QoS mechanism using TCPW-LP and router-based QoS mechanism using priority queuing. In the overlay QoS case, TCP sessions are relayed at TCP PEPs and the routers employ FIFO queuing. On the other hand, in the router-based QoS case, TCP PEPs are not used and all routers employ priority queuing where RTP/UDP sessions have higher priority over TCP sessions.

In Fig. 15, average end-to-end delay of the RTP/UDP session are plotted for different number of coexisting TCP sessions. On the x-axis, the number of TCP sessions per sender group is shown. This figure shows that, without QoS mechanisms, RTP/UDP packets suffers large queuing delays even when there are one TCP sessions at each group, whereas priority queuing causes no queuing delays for RTP/UDP packets. If TCP sessions are relayed at TCP PEPs using TCPW-LP, despite that RTP/UDP packets and TCP packets share the same FIFO queue at the bottleneck router; queuing delays of these packets are well controlled and RTP/UDP packets experiences less queuing delay.

In Fig. 16, packet loss rate of the RTP/UDP sessions is plotted for the same experiment. Both TCPW-LP and priority queuing yield no packet losses for the RTP/UDP session, whereas there are considerable number of packet losses if no QoS mechanisms are employed. We note here that, in any cases, bottleneck link utilization is almost 100%, which means TCPW-LP provides high residual bandwidth utilization for TCP sessions while it gives high UDP/RTP quality.

These figures also provide some insights on scalability of the proposed overlay QoS mechanism. For example, given that RTP/UDP communication allows maximum 10 msec of average queuing delay, 24 TCPW-LP sessions are accommodated at the 10 Mbps bottleneck link, 420 kbps per session, and thus 2400 TCPW-LP sessions at a 1 Gbps link. Assuming that low-priority transfers are usually used for relatively smaller number of long-lived communications, such as database backup or buck content perfecting, this number of sessions sounds quite large for this purpose.

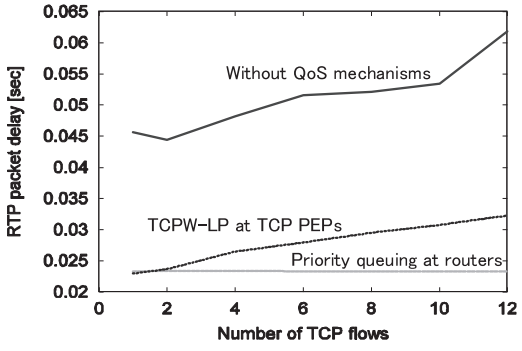


Fig. 15 End-to-end delay of RTP/UDP session.

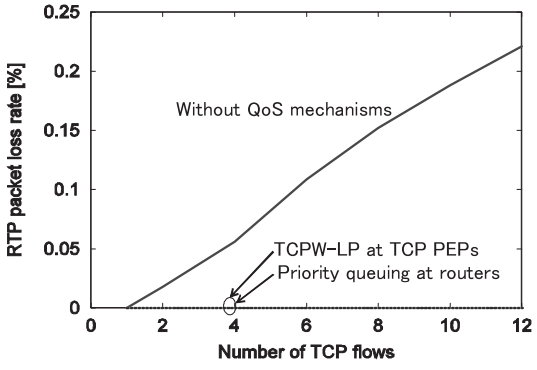


Fig. 16 Packet loss rate of RTP/UDP session.

6. Internet Measurements for Overlay QoS

6.1 Measurement Setup

Figure 17 shows the setup for a set of Internet measurements. Two senders at Osaka are connected to a receiver at Kawasaki via an Internet path with 17 hops and 17 msec round trip propagation delay. Although both senders use TCP-Reno, one of them is connected to out TCP middlebox [13] that relays TCP-Reno flows from the sender by TCPW-LP flows to the receiver. With this box, one can provide low-priority service without modifying user hosts and routers.

In this section, all TCP flows use Iperf [14] to generate packets and their throughputs are normalized to the access link capacity.

6.2 Coexistence with Foreground Traffic

Throughput of coexisting TCP-Reno and TCPW-LP flows are shown in Fig. 18 and Fig. 19. In Fig. 18, a TCPW-LP flow starts first and then a TCP-Reno flow starts at 30 sec, followed by a pause of the TCPW-LP flow at 60 sec. In Fig. 19, a TCP-Reno flow starts first and then a TCPW-LP flow starts at 30 sec, followed by a pause of the TCP-Reno flow at 60 sec.

From these figures, it is found out that average throughput of single TCP flow is around 0.6 to 0.8. And it is confirmed that TCPW-LP is at lease as efficient as TCP-Reno when the TCP-Reno flow is idle.

In Fig. 18, when a TCP-Reno starts sending while a TCPW-LP flow is already active, the TCPW-LP flow quickly recognizes the injection of the TCP-Reno flow

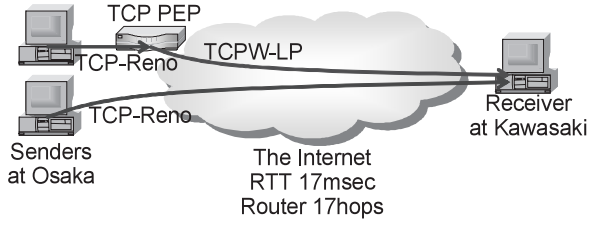


Fig. 17 Network topology (Internet measurement).

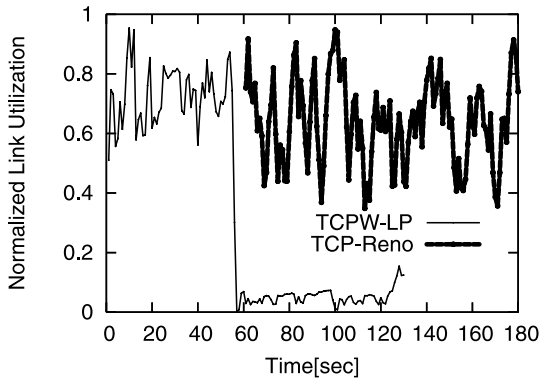


Fig. 18 Throughput of coexisting TCP-Reno and TCPW-LP flows (1).

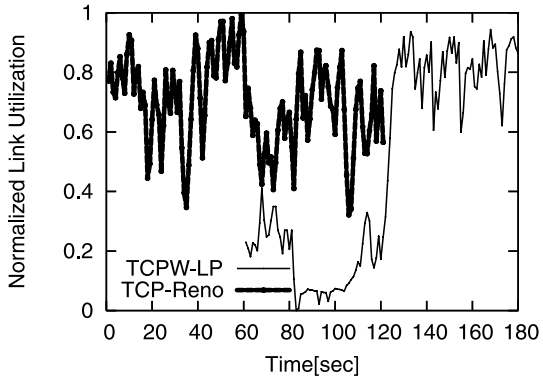


Fig. 19 Throughput of coexisting TCP-Reno and TCPW-LP flows (2).

and reduces its throughput down to around 0.05. When these two flows compete, we can hardly observe throughput degradations of the TCP-Reno flow. From 30 sec to 90 sec, the throughput of the TCP-Reno flow stays at around 0.6 with or without the TCPW-LP flow.

In Fig. 19, similar results are obtained, except that the TCPW-LP flow obtains rather larger throughput when it competes with the TCP-Reno flow. This is because, if a TCPW-LP flow starts when a TCP-Reno flow is already sending, the TCPW-LP flow observes larger minimum RTT than actual round trip propagation delay and it underestimates the queuing delay. This is a well known problem for TCP-Vegas [5]. But, since TCP-Vegas stabilizes congestion window in steady state while TCPW-LP follows AIMD (Additive Increase Multiple Decrease) behavior, TCPW-LP flows have much chance to get smaller minimum RTT measurement. In addition, TCPW-LP would be used for background transfers with relatively longer durations, and thus it may have a chance to observe minimum RTT as small as the round trip propagation delay.

6.3 Efficiency in Bandwidth Utilization

In Fig. 20, socket buffer size for TCP-Reno is reduced so that throughput of a TCP-Reno flow is limited to 0.5, to investigate how TCPW-LP flows utilize the residual bandwidth left unused by TCP-Reno flows. In this experiment, a

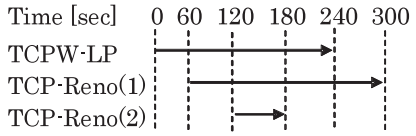


Fig. 20 Traffic pattern for Fig. 21.

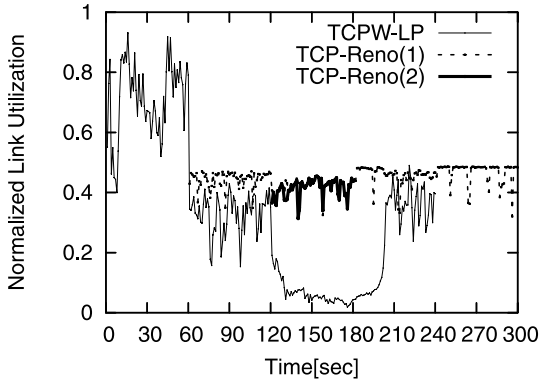


Fig. 21 Throughput of coexisting TCP-Reno and TCPW-LP flows (3).

TCPW-LP flow start at 0 sec and then two TCP-Reno flows starts at 60 sec and 120 sec, respectively. Then, the TCPW-LP flow stops at 240 sec and one of TCP-Reno flows stops at 300 sec.

As shown in Fig. 21, when there is one TCP-Reno flow (60–120 sec, 180–300 sec), the flow obtains throughput of 0.5 as it is expected, without any effects from coexisting TCPW-LP flow. The total throughput of these two flows is 0.8; the same throughput as a lone TCPW-LP flow obtains during 0–60 sec. The result means that the TCPW-LP flow can efficiently use up the residual bandwidth without damaging coexisting TCP-Reno flow. On the other hand, when there are two TCP-Reno flows, these flows use up the whole available bandwidth and coexisting TCPW-LP flow has very low throughput, as it is expected.

6.4 Fairness among TCPW-LP Flows

Finally, throughputs of coexisting two TCPW-LP flows are shown in Fig. 22. One of them is active during 0 sec to 120 sec, while the other flow is active during 60 sec to 180 sec.

When there is only one TCPW-LP flow (0–60 sec, 120–180 sec), a TCPW-LP flow can efficiently utilize the available bandwidth. When two TCPW-LP flows coexist, although they are efficiently utilizing the bandwidth, the second flow gets slightly higher throughput. Since the second flow starts when the first flow has already fully utilize the capacity, the second flow observes minimum RTT of 19 msec, while the first one observes minimum RTT of 17 msec. This is the same problem described above, but it may be expected that they can reach fair share after a while.

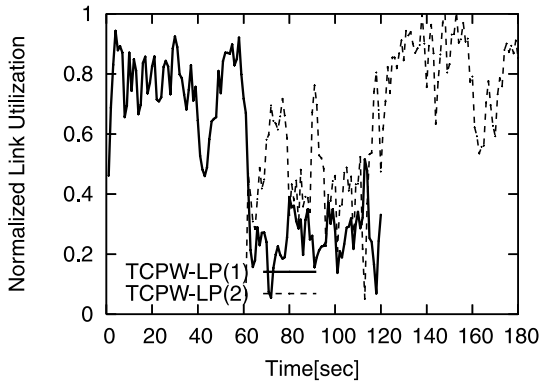


Fig. 22 Throughput of coexisting TCPW-LP flows.

7. Concluding Remarks

In this paper, we presented an overlay QoS mechanism using TCP Westwood Low-Priority, TCPW-LP, as an end-to-end transport protocol to realize two-class service prioritization. To optimize the fundamental trade off of TCPW-LP between efficiency and non-intrusiveness to foreground flows, we introduced a new congestion metric called Foreground Traffic Ratio. By dynamically adjusting the virtual queue length threshold based on this ratio, TCPW-LP flows react appropriately to congestion and whether such congestion is caused by foreground or background flows.

We conducted a series of simulation experiments and Internet measurements to evaluate TCPW-LP. The results show that TCPW-LP is largely non-intrusive to coexisting foreground flows and can utilize most of the residual capacity; thus, it can efficiently provide overlay prioritized QoS services. Remarkably, whenever a Reno flow underutilizes the path capacity, TCPW-LP successfully utilizes most of bandwidth left unused without degrading Reno flows.

Acknowledgements

The Internet measurements have not been done without a help of Prof. Murata, Prof. Hasegawa, and Mr. Mori, Osaka University. The measurements are carried out as collaboration between NEC and Osaka-U.

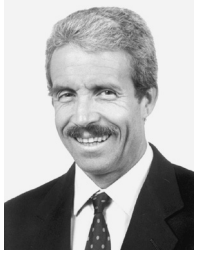
References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," IETF RFC 2475, 1998.
[2] P. Gervros, F. Risso, and O. Kirstein, "Analysis of a method for differential TCP service," Proc. Globecom, pp.1699–1708, 1999.
[3] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP-nice: A mechanism for background transfers," OSDI02, 2002.
[4] A. Kuzmanovic and E. Knightly, "TCP-LP: A distributed algorithm for low priority data transfer," Proc. IEEE INFOCOM, 2003.
[5] L.S. Brakmo and L.L. Perterson, "TCP vegas: End-to-end congestion avoidance on a global Internet," IEEE J. Sel. Areas Commun., vol.13, no.8, pp.1465–1480, 1995.
[6] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," Proc. Mobicom, pp.287–297, 2001.
[7] R. Wang, M. Valla, M.Y. Sanadidi, B.K.F. Ng, and M. Gerla, "Efficiency/friendliness tradeoffs in TCP westwood," Seventh IEEE Symposium on Computers and Communications, 2002.
[8] R. Wang, M. Valla, M.Y. Sanadidi, and M. Gerla, "Adaptive bandwidth share estimation in TCP westwood," Proc. Globecom, 2002.
[9] W. Feng and S. Vanichpun, "Enabling compatibility between TCP reno and TCP vegas," Proc. SAINT, 2003.
[10] T.V. Project, "UCB/LBNL/VINT network simulator - ns (version 2)," available from http://www.isi.edu/nsnam/ns/
[11] NS-2 TCP-LP code and simulation scripts, http://www-ece.rice.edu/networks/TCP-LP/ns2
[12] H. Shimonishi, M.Y. Sanadidi, and M. Gerla, "Service differentiation at transport layer via TCP Westwood Low-Priority (TCPW-LP)," Proc. ISCC2004, Jan. 2004.
[13] I. Maki, G. Hasegawa, M. Murata, and T. Murase, "Performance analysis and improvement of TCP proxy mechanism in TCP overlay networks," Proc. IEEE International Conference on Communications, Wireless Networking (ICC 2005), May 2005.
[14] NARANR, "NARANR/DAST: Iperf 1.7.0 The TCP/UDP bandwidth measurement tool," available from http://dast.nlanr.net/Projects/Iperf/

Hideyuki Shimonishi See this issue, p.2280.

Takayuki Hama received his M.E. from Graduate School of Science and Engineering, Waseda University, Tokyo, Japan in 2004. He joined NEC Corporation in 2004 and has been engaged in research on TCP overlay network. Mr. Hama is a member of IEICE.

M.Y. Sanadidi was born in Egypt where he received his high school diploma from College Saint Marc D'Alexandrie, and his B.Sc. from the University of Alexandria. He earned his M.Sc. from Penn State, and his Ph.D. in Computer Science from UCLA. He is currently an Adjunct Professor at the UCLA Computer Science Department, where he is co-Principal Investigator on NSF and industry sponsored research in high performance Internet protocols. At UCLA, he also teaches undergraduate and graduate courses on computer networks, queuing systems, and probability modeling and analysis. Dr. Sanadidi was a Manager and Senior Consulting Engineer at AT&T/Teradata, and previous to that, he held the position of Computer Scientist at Citicorp, and Assistant Professor at the Computer Science Department, University of Maryland, College Park, Md. Professor Sanadidi, who is a Senior Member of the IEEE and Member of the ACM, has co-authored over fifty conference and journal papers and has been awarded two best paper awards. He has served as reviewer of journal publications, member of technical program committees and organizing committees, keynote speaker, as well as chairman of professional conferences. Dr. Sanadidi has been awarded two patents, and has consulted for a number of industrial concerns. His current research interests are in path characteristics estimation and its applications in congestion control, adaptive multimedia streaming, and hybrid wire/wireless networks.



Mario Gerla received a graduate degree in engineering from the Politecnico di Milano in 1966, and the M.S. and Ph.D. degrees in engineering from UCLA in 1970 and 1973. He became IEEE Fellow in 2002. After working for Network Analysis Corporation, New York, from 1973 to 1976, he joined the Faculty of the Computer Science Department at UCLA where he is now Professor. His research interests cover distributed computer communication systems and wireless networks. He has designed and implemented various network protocols (channel access, clustering, routing and transport) under DARPA and NSF grants. Currently he is leading the ONR MINUTEMAN project at UCLA, with focus on robust, scalable network architectures for unmanned intelligent agents in defense and homeland security scenarios. He is also conducting research on scalable TCP transport for the Next Generation Internet (see www.cs.ucla.edu/NRL for recent publications).

Tutomu Murase See this issue, p.2280.