Vol. 41 No. 2 Transactions of Information Processing Society of Japan Feb. 2000

Regular Paper

TCP-Rate-Probing-Based Adaptation for Continuous Media Communications

Yoshito Tobe,[†] Yosuke Tamura,[††] Hideaki Nishino[†††] and Hideyuki Tokuda[†††]

Rate control of continuous media (CM) UDP flows over the Internet should retain TCP-friendliness. Despite many efforts to create TCP-friendly algorithms, it is often difficult to adjust parameters pertaining to the algorithms in order to achieve exact fairness with TCP flows. In this paper, we propose a scheme called TCP-Rate-Probing-Based Adaptation (TPBA). In TPBA, a CM flow intermittently changes its transport protocol to TCP to measure a TCP-exact rate, and uses the rate with UDP. We have built an experimental system with FreeBSD PCs and shown that TCP-friendly rate control can easily be achieved for both a shorter and a longer round-trip-time connections.

1. Introduction

With the growing need to carry continuous media (CM) and other traffic that requires high priority over the Internet, many efforts are being made to provide guaranteed[19] or differentiated service[2]. Although these are important, there are still many cases in which CM need to be transmitted on over a best-effort basis. In order to manage best-effort flows, several traffic management schemes for routers, such as Class-Based Queuing (CBQ)[7], Weighted Fair Queuing (WFQ)[6], and their enhanced versions have been implemented to ensure sharing of a limited amount of bandwidth. TCP senders' congestion control responds to the control excersized by these schemes whether or not the network is congested.

CM flows mostly use RTP[18]/UDP instead of TCP. UDP flows do not automatically control their rates to ensure fairness in the network, which may make both the network and themselves inefficient. To overcome such a problem, TCP-friendly rate control algorithms[15],[17],[20],[23] have been introduced.

Although these algorithms perform additive increase/multiplicative decrease rate control like TCP, our observation is that TCP flows are so quick in reducing the rate that they are deprived of bandwidth by the reduction. We believe that attaining fairness between UDP and TCP flows merely by means of the UDP sender's control with Receiver Reports (RRs) of RTCP, which is a control protocol for RTP, has difficulty in adjusting parameters pertaining to the control.

To overcome the above problem, we measure the TCP-exact rate that a UDP flow can utilize by replacing UDP by TCP at a certain point during the lifetime of the flow. We call such a period when TCP is used a "rate-probing period." Rate-probing periods are inserted repeatedly to facilitate adaptation to the available rate. This scheme is referred to as TCP-Rate-Probing-Based Adaptation (TPBA). Since rate probing requires the flow to switch transport protocols, there are concerns about degradation of the delay performance and possible corruption of Application Data Units (ADUs)[5] at the switching boundaries. We investigate the degradation through experiments both for shorter (approximately 12 ms) and longer (approximately 160 ms) round-trip-time connections. Although the TCP-exact rate is obtained with TPBA, there are still tunable parameters. Therefore, we also discuss the sensitivity to these parameters.

In this paper, we describe TPBA and show evaluation results obtained with our experimental system. After outlining related work in Section 2, in Section 3 we describe the difficulty of mixing UDP and TCP in our experimental system. In Sections 4 and 5, we detail TPBA and socket APIs for TPBA, respectively. In Section 6, we give the evaluation results, and in Sections 7 and 8, respectively, we discuss the

† Keio Research Institute at SFC, Keio University
†† Graduate School of Media and Governance, Keio University
††† Faculty of Environmental Information, Keio University

resuls further and outline our plans for future work.

2. Related Work

Rate probing is not a new topic, and various mechanisms have been proposed. Packet Pair[3),10)] is a technique for measuring a bottleneck bandwidth with little disturbance to the network. Variants of the Packet Pair algorithms are Sender Based Packet Pair, Receiver Based Packet Pair, Packet Bunch Mode[16)], and Receiver Only Packet Pair[11)]. Although they are able to measure bandwidth, a long measurement duration is necessary to obtain statistically stable values, they cannot be used to compute a TCP-equivalent rate. Our work differs from other schemes that use rate probing in that it probes for a TCP-exact rate.

Several TCP-friendly rate control algorithms have been proposed, all of them based on the reported macroscopic TCP characterization[12),13)]. According to the characterization, the steady state throughput of a TCP connection in the absence of timeouts, th is given as follows:

$$th = C * \frac{\mathrm{MTU}}{rtt * \sqrt{p}},$$

where C, rtt, and p are a constant value ranging from 0.9 to 1.5, the round-trip time (RTT), and the expected number of reduction events per packet sent. Since the formula deviates from the actual performance when p becomes higher, and retransmission timeouts were not taken into consideration, an improved model was introduced[15)].

LDA[20)] proposes a scheme based on RTCP reports with loss ratio and RTT. It also uses Packet Pair to estimate the bottleneck link bandwidth. Although it provides additive increase/multiplicative decrease control, there are many tunable parameters. Our scheme also has tunable parameters, but these parameters affect the TCP-friendliness less than those of LDA. We compare our scheme with LDA in Section 6. Rate Adaptation Protocol (RAP)[17)] is a fine-grained additive increase/multiplicative decrease control on the order of magnitude of the RTT. Its authors claim that RAP has been proven to provide TCP-friendliness by simulation study, but this has not been verified by real implementation. Other studies such as Turlettie, et al.[22)] and Vicisano, et al.[23)] are multicast-oriented. A variety of similar schemes have been proposed and are listed at a Web site[21)].

Recently Balakrishnan, et al.[1)] have proposed Congestion Manager. This is an entity included in an end host to maintain information about the rate and congestion. A pair of Congestion Managers exchange probing packets that are controlled under their own acknowledgment-based protocol. The advantage of this approach is that concurrent flows that share the same path can utilize the same information about congestion control. TCP and/or UDP flows notify the Congestion Manager of information about congestion control, such as the RTT measured in TCP flows. The Congestion Manager uses the information as hints for congestion control, and measures the available rate. It is therefore more efficient in congestion control than our scheme if the number of concurrent flows is large. Despite this advantage, our scheme is more precise in measuring a TCP-exact rate. First, it uses TCP to measure the TCP-exact rate, while the congestion manager probes for a rate with its own protocol. In general, the rate can be obtained with any probing protocol, but there is no guarantee that the rate measured by the probing protocol is the TCP-exact rate unless it behaves in the same way as TCP. Since the probing protocol of the Congestion Manager includes a window-based acknowledgment mechanism, it is considered to provide a more accurate TCP-like rate than Packet-Pair-based probing. However, the mechanism is not identical to TCP, and it is unknown how close the measured rate is to the TCP-exact rate. Second, out-of-band probing is not performed in our scheme. The Congestion Manager produces an extra bandwidth of probing. When the flows traverse a narrow link, out-of-band probing may come with a cost. In our scheme, a RTP/UDP flow replaces its transport protocol with TCP to measure the available rate with data on the flow, which avoids producing unnecessary bandwidth.

3. Observation of Intermingled UDP and TCP Flows

TCP-friendly algorithms described in previous studies to some extent rely on the indication of packet loss; the sending rate is increased until packet loss is reported by the receiver. We examine the validity of dependence on the packet loss through an experiment. **Figure 1** depicts the configuration of our experimental system. Router 1 is a personal computer with FreeBSD.

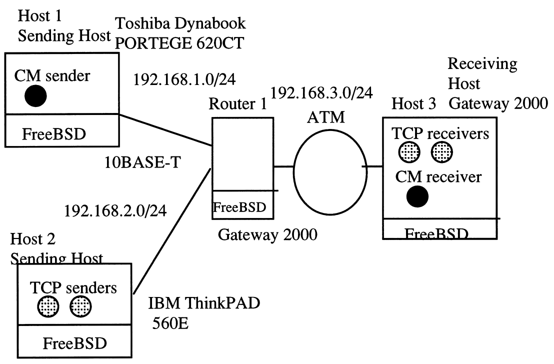


Fig. 1 Experimental system.

An Asynchronous Transfer Mode (ATM) driver code for Efficient Networks' ATM cards is installed on Router 1 and Host 3. Measurement was carried out on the incoming ATM link of Host 3 by obtaining the value of the Pentium counter when an interrupt of an AAL5-frame reception occurs. The ATM network consists of one ATM switch and the round-trip time between Host 1 and 3 is less than 2 ms.

Hosts 1 and 2 send an RTP/UDP flow and two TCP flows, respectively. Let us define these flows as follows:

- Flow 1: One TCP flow from Host 2 to 3.
- Flow 2: Another TCP flow from Host 2 to 3.
- Flow 3: An RTP/UDP flow from Host 1 to 3. Host 1 and 3 exchange RTCP messages.

All traffic is aggregated into one Virtual Channel (VC) over ATM and shaped into 1.2 Mbps at the ingress of the VC. The size of RTP payload of Flow 3 is 512 bytes. At time 0, Flows 1 and 2 are initiated. At time 1 s, Flow 3 starts.

As the rate control algorithm, we use LDA [20] in this experiment. The parameters and symbols used in LDA are listed in **Table 1**.

In LDA, the RTT as well as the loss ratio is measured and consequently rate control according to the formula of the TCP-equivalent rate can be applied. In addition, the bottleneck bandwidth is reported by using the Packet Pair technique and a maximum value of the rate is suggested. At each adaptation point, when there is no loss, the rate is increased by AIR. But AIR is limited by the rate at which TCP would increase until the next adaptation point, as shown in line (C) of **Fig. 2**. In contrast, when a loss is reported, the rate is multiplicatively decreased. Figure 2 shows the pseudo-code for the fundamental algorithm of LDA. Although the original LDA accommodates multiple receivers, Figure 2 is confined to a single-receiver case, since we are investigating TCP-friendliness for unicast communications. Although lines (A) in the figure are not presented in Ref. 20), we added lines (A) because there are cases in which the bottleneck bandwidth can decrease without a report of packet loss. Without lines (A), the value of AIR can become minus.

Table 1 Parameters and symbols of LDA (*: configurable parameters).

$T_{adapt}(*)$	interval between two adaptation points
T_{rtcp} (*)	interval between two RTCP reports
r	sending rate
$r_0(*)$	initial value of r
$r_{min}(*)$	minimum value of r
AIR	additive increase rate
$AIR_0(*)$	initial value of AIR
R_f (*)	reduction factor
b	bottleneck bandwidth
p_{loss}	indicated loss ratio

```
Initialization:
  $r = r_0$;
  $AIR = AIR_0$;
At each adaptation point:
  if ($p_{loss}$ == 0) {
       if ($r < b$) { ..............................................(A)
          $AIR = AIR \times (1 - \frac{r}{b})$;.........................(B)
          $AIR = \min(AIR, \frac{\text{packet size}}{2\tau}(\frac{T_{adapt}}{\tau} + 1))$;
                                              ....................(C)
          $r = r + AIR$;
       } ..........................................................(A)
       if ($r >= b$) $r = b$; ...................................(A)
  }
  else { /* $p_{loss} > 0$ */
       $AIR = AIR_0$;
       $r = r \times (1 - p_{loss}R_f)$;
       if ($r < r_{min}$) $r = r_{min}$;
  }
```

Fig. 2 Algorithm of LDA (for unicast).

We set the parameters for LDA as follows: AIR_0 = 10 kbps, R_f = 3.0, T_{adapt} = 3 s, and T_{rtcp} = 1 s. Let us initiate Flow 3 at 600 kbps, which is intentionally a higher value than a fair shared rate (= 400 kbps). The measured rates of Flows 1 to 3 are shown in **Fig. 3**. Although the difference in rates between a UDP and two TCP flows remains, RTCP RR reports no loss. Hence the difference is not reduced but becomes slightly larger at the adaptation points. This is mainly due to the granularity of rate control. TCP is window-based, and a single packet loss immediately triggers fast retransmit or retransmission timeouts. Many rate control algorithms for RTP/UDP flows rely on information

Table 2 Parameters and symbols of TPBA (*: configurable parameters).

$T_{running}(*)$	Length of the running period
$T_{TCP}(*)$	Interval between monitoring actions during the rate-probing period (1 s)
$\varepsilon(*)$	Threshold value to determine the end of the rate-probing period (30 kbps)
$T_{rtcp}(*)$	Interval between RTCP reports during the running period
$R_f(*)$	Reduction factor (3.0)
t_{maxseg}	Maximum segment size of TCP
p_{loss}	Indicated loss ratio
$r[i]$	Calculated rate in the i-th cycle of the rate-probing or running period.
$cwnd_{new}$	Value of the congestion window when the running period starts
$cwnd_{old}$	Value of the congestion window when the running period ends

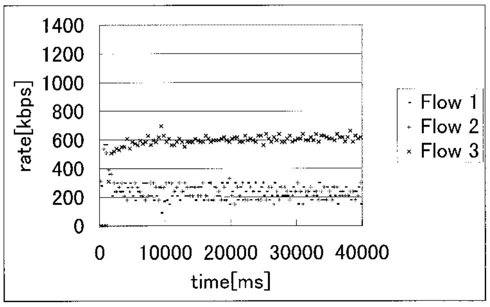


Fig. 3 Measured rates with LDA.

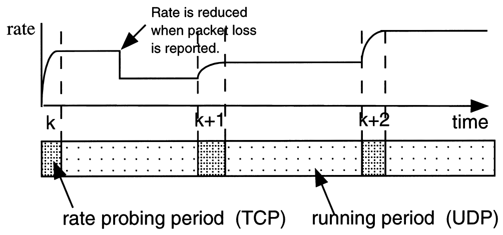

rate
Rate is reduced when packet loss is reported.
k
k+1
k+2
time
rate probing period (TCP)
running period (UDP)

Fig. 4 Rate probing periods appear repeatedly.

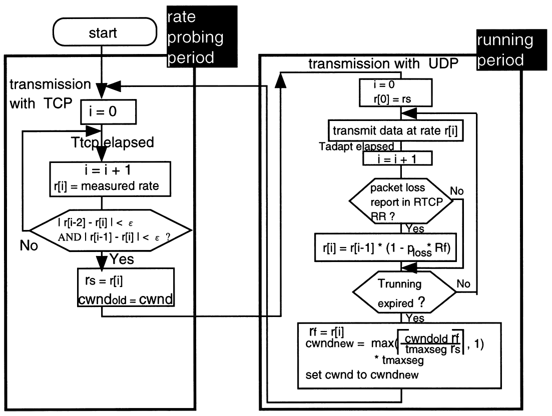


Fig. 5 TPBA sender's behavior.

on packet loss. Therefore even when the rate control is executed, there is no decision to decrease the rate until the rate for the UDP flow becomes so high that some packets are lost. It should be noted that even at the adaptation point, without a report of loss, the rate is not reduced sufficiently to be fair with TCP, since the reported bandwidth is not always the TCP-equivalent rate.

We further investigate the behavior of LDA in Section 6, but it is important to note the difficulty of increasing the rate without interfering with TCP.

4. TCP-Rate Probing Based Adaptation

In accordance with our previous observation, we propose TPBA. In TPBA, like other CM applications, a CM flow uses RTP/UDP. However, it repeatedly switches its transport protocol to TCP in order to measure the **exact TCP-equivalent** rate. Thus, the rate-probing period and the running period appear alternately, as shown in **Fig. 4**. The parameters and symbols used in TPBA are listed in **Table 2**. **Figure 5** illustrates the TPBA sender's behavior. The TPBA sender starts transmission with the rate-probing period; data are transmitted over TCP. During the rate-probing period, only the usual flow data are transmitted and no data for probing is used. Conseqently, the action of probing does not generate extra bandwidth that is irrelevant to data transmission. For every cycle of T_{tcp}, the sender monitors the transmission rate r, which is calculated from the movement of the TCP sender's unacknowledged sequence number *snd_una*. The monitored rate is stored as $r[i]$, where i is the number of cycles since the rate probing period started. The TPBA sender moves to the running period if the following conditions are satisfied:

$$|r[i-2] - r[i]| < \varepsilon,$$

and

$|r[i-1] - r[i]| < \varepsilon$.

Before the transition to the running period, $r[i]$ is stored as r_s. Once the TPBA sender moves to the running period, it transmits data at the rate r_s. The rate control scheme in the running period is similar to that of LDA. At each adaptation point, the status provided by RTCP RR messages is examined. If packet loss is reported, the rate is reduced:

$$r[i] = r[i-1] \times (1 - p_{loss} R_f),$$

where i is the number of adaptation cycles. However, unlike LDA, the TPBA sender does not increase its rate in the running period. If short-term heavy congestion occurred during the running period for some reason, the rate could be raised in the following rate-probing period. When a time $T_{running}$ has passed since the beginning of the running period, the TPBA sender moves to the rate-probing period.

When a TPBA-capable flow is created, both UDP and TCP sockets are created. The TCP connection may be idle during the running period although it remains active. In the current TCP implementation, the sender's congestion window size, *cwnd*, is reduced to the minimum level when the sender detects that the TCP connection is idle. To avoid a slow start, we have disabled this reset of *cwnd* for the TPBA-capable flow. Instead, *cwnd* is set to the following value, $cwnd_{new}$ when the rate-probing period starts:

$$cwnd_{new} = \max(\lceil cwnd_{old} r_f / (t_{maxseg} r_s) \rceil, 1) \times t_{maxseg},$$

where $cwnd_{old}$, r_f, and t_{maxseg} are the value of *cwnd* when the running period began, the rate of UDP when the running period lasts, and the maximum segment size of the TCP connection. Note that r_f does not become higher than r_s. Thus TPBA provides UDP and TCP with shared information about the rate at the boundary of both periods.

To discuss rate control, three issues must be addressed[9]: the decision function, the increase/decrease algorithm, and the decision frequency. TPBA's handling of these issues may be summarized as follows.

- Decision function: If no congestion is detected during the running period, do not change the transmission rate. If congestion is detected at the adaptation point in the running period, decrease the transmission rate. During rate-probing period, comply with the TCP algorithm.
- Decrease algorithm: During the running period, the transmission rate is decreased in accordance with LDA. During the probing period, the transmission rate is decreased in accordance with the algorithm of TCP. In both periods, therefore, the rate is decreased multiplicatively.
- Decision frequency: The transmission rate is never increased, but can be reduced during the running period. Therefore, even when the interval between rate probing actions becomes longer, disturbances to other TCP and TCP-compliant flows are minimized. In this sense, the frequency does not strongly affect the network performance.

As a metric for evaluating fairness, we use the following f [4),8)] for rates b_i of flow i ($i = 1 \ldots n$):

$$f \equiv \frac{(\sum_{i=1}^{n} b_i)^2}{n * (\sum_{i=1}^{n} b_i^2)},$$

where f is 1 when b_i's are exactly the same.

Timing of Switching Periods

When the flow switches from the rate-probing period to the running period and vice versa, there will be a sudden change in the delay, and ADUs can be corrupted. To prevent ADU corruption, the switching is delayed until an ADU boundary is detected at the sender. Socket API is enhanced to indicate the ADU boundary, as described in the next section. The change in delay is minimized by tuning *cwnd* as previously mentioned, but it should be evaluated through experiments. Let j_A denote the inter-arrival time of ADUs at the receiver. We use j_A for evaluation; our scheme should not incur a large difference in j_A at the switching point.

When a flow moves from the probing period to the running period, we need to ensure that the data stored in the socket buffer of TCP is evacuated. To enable such evacuation, the TCP_NODELAY option is set before the last data is written into the socket buffer. The TCP_NODELAY option is disabled when the flow enters the probing period again.

Sensitivity to Parameters

Since there are several parameters in TPBA, we discuss how they affect the behavior of TPBA.

- T_{TCP}: Calculation of a rate requires some averaging period. In particular, window-based protocol TCP generates bursty traffic, and a shorter period than the dynamics of the congestion window is meaningless. On the assumption that we use a network with an average RTT less than 200 ms, we set T_{TCP} to 1 s, a value several times larger than the average. For a lower-delay network, it is possible to shorten the rate-probing period by using a smaller value of T_{TCP}. In contrast, for a larger-delay network such as a satellite network, T_{TCP} should be larger. Although we are using a static value for T_{TCP} at the moment, we think that manual configuration of this parameter can be eliminated by examining the statistics of RTT. This topic is discussed further in Section 7.

- ε: The value of ε determines the allowable difference in rate. Although we chose 30 kbps, this value may be too strict or too loose when the average rate goes above 10 Mbps or below 100 kbps. An alternative criterion to determine the end of the rate-probing period is based on the proportional fluctuation of $r[i]$. This topic is discussed further in Section 7.
- $T_{running}$: While it is desirable to set $T_{running}$ to 0 from the viewpoint of TCP-exactness alone, CM applications need to run in the running period as long as they can in order to minimize the jitter in data delivery. We should therefore ask ourselves how much we can increase $T_{running}$ and what effect a longer value of $T_{running}$ has. During the running period, the rate may be reduced but cannot be raised. Therefore, if packet loss takes place frequently with a large value of $T_{running}$, the rate may remain unnecessarily small and cannot be updated to a fair rate. Although the possibility of heavy packet loss is small in the running period, since additive increase is disabled, running at an unnecessarily small rate should be avoided. One solution to this situation is to transit immediately to the rate-probing probing period after successive large reductions in rate.
- T_{adapt} and R_f: These parameters are the same as those of LDA. Unlike LDA, TPBA does not use the information contained in RTCP RR to increase the rate. Therefore, TPBA is less sensitive to T_{adapt}. Furthermore, a flow in the running period of TPBA does not encounter packet loss caused by overshooting of its rate. Therefore, the value of R_f is less sensitive to the stability of behavior in TPBA. We use 3.0 for R_f, which is a suggested value in Ref. 20).

5. Fsocket

In this section, we describe our implementation of TPBA over FreeBSD and the application programming interface.

5.1 Flow Pair

To facilitate smooth transition between two protocols, flows with both UDP and TCP are created. Only one flow is used for data transmission and the other is used as a backup. More specifically, the TCP flow is used in the rate-probing period, and data is carried over the UDP flow in the running period. The pair of these flows is referred to as a flow pair. Besides the flow pair, another UDP flow is created for exchanging RTCP messages.

5.2 Application Programming Interface

We design a socket interface that keeps users unaware of the underlying transport protocols. For that purpose, an interface of a flow pair socket, *fsocket*, is defined. In the socket interface of BSD UNIX, streams and datagrams use different parameters or even system calls for communication. Examples of such differences are accept(), connect(), send(), and sendto(). The fsocket interface conceals the difference and provides a set of unified system calls listed in **Fig. 6**. Since TPBA communications are limited to remote ones with the AF_INET domain, parameters for specifying the domain are omitted. f_socket() returns a descriptor of fsocket, fsock_fd. The descriptor is used throughout the lifetime of communication. f_dst() is a function for registering a remote site. A parameter, adu_ind, in f_send specifies the boundary of an ADU.

In addition to the fsocket interface, an *fsocket library* that performs TPBA operations is introduced. All functions of TPBA described in the previous section are confined in this library. **Figure 7** illustrates a flow pair, fsocket interface, and fsocket library. To enable receiving both TCP and UDP data in the receiver, select() is used inside the fsocket library.

```
int f_socket(struct sockaddr_in *local_addr);
int f_dst(int fsock_fd, struct sockaddr_in
        *remote_addr);
int f_close(int fsock_fd);
int f_send(int fsock_fd, caddr_t msg, size_t len,
          int flags, int adu_ind);
int f_recv(int fsock_fd, caddr_t msg, size_t len,
          int flags);
```

Fig. 6 Fsocket API.

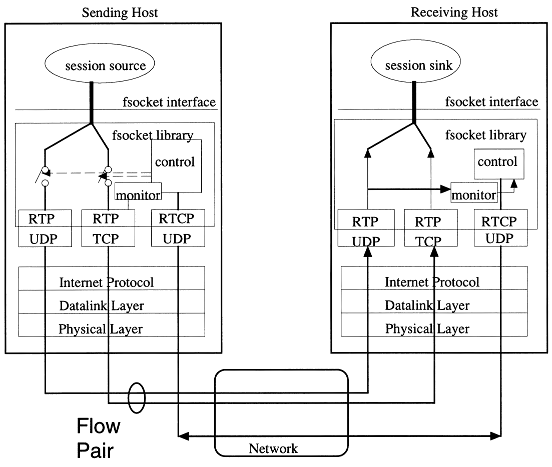


Fig. 7 Flow pair.

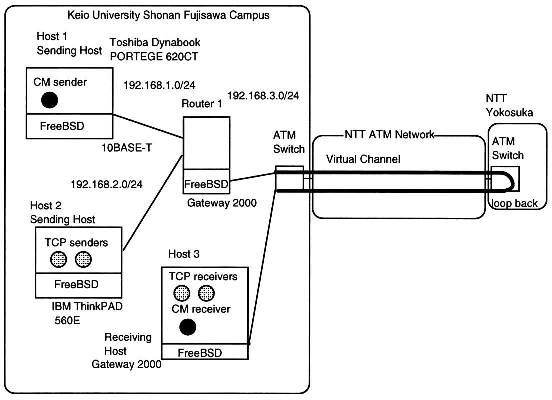


Fig. 8 Topology 1.

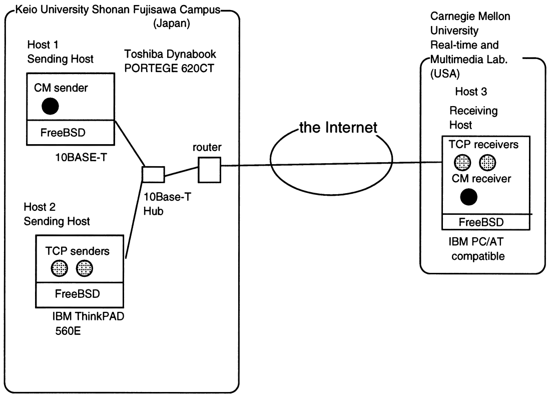


Fig. 9 Topology 2.

A slight modification to the kernel is required to monitor the TCP transmission rate at the sender, but there is no need to change the kernel at the receiver.

6. Experiment

In this section, we first examine the behavior of TPBA over an isolated testbed network and over the public Internet, and then compare TPBA with other TCP-friendly schemes over the testbed network. We also examine the sensitivity of parameters pertaining to TPBA.

6.1 Examining the Behavior of TPBA

We used the same hosts and router as shown in Fig. 1. TPBA is implemented on Hosts 1 and 3 with FreeBSD 2.2.6R. We used two types of networks. In Topology 1, a loop-back VC over ATM from Keio University to NTT Yokosuka is used as shown in **Fig. 8**. There is no router along the VC, and the ATM interfaces of Router 1 and Host 3 belong to the same IP subnet. All traffic generated at Hosts 1 and 2 is aggregated into one VC at the ATM output link of Router 1 and shaped into 1.2 Mbps. In Topology 2, we used an Internet path from Keio University (Japan) to Carnegie Mellon University (U.S.A.) shown in **Fig. 9**. In Topology 2, measurements are done at the receiver socket instead of the driver level. We chose these topologies in order to test our scheme in both traffic-controlled and public networks.

The conditions of flows are the same as those described in Section 3, but Flow 2 was modified such that Flow 2 starts at time 10 s and ends at time 25 s. In addition, we used two types of ADUs.

- ADU1: one ADU consists of one 64-byte payload.
- ADU2: one ADU consists of ten 512-byte payload.

As TPBA parameters, $T_{running}$, T_{adapt}, and T_{rtcp} were set to 10 s, 3 s, and 1 s, respectively.

First we used ADU2 to observe the transition of rates in Topology 1. The measured round-trip time between Host 1 and 3 was around 12 ms. Snapshots of the measured rates of Flows 1 to 3 and fairness f are shown in **Figs. 10** and **11**. When Flow 2 starts at 10 s, Fairness f is dropped to 0.7 because of TCP's slow start of Flow 2. Although Flow 2 increases its rate, Flow 3 does not decrease its rate, because there is no packet loss. As a result, the rate of Flow 1 is decreased. After Flow 3 enters the rate-probing period at time 13.8 s, the rate of Flow 3 is adjusted to a fair shared value that can be used during the following running period. Thus fairness is recovered to a value near 1. After Flow 2 has ended, the difference between the rates of Flow 1 and 3 is corrected during the following rate-probing period. It has been shown that rate increase is possible with

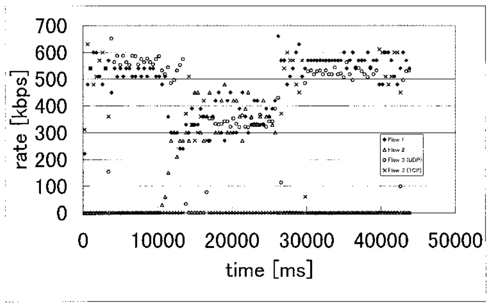


Fig. 10 Measured rates for ADU2 (Topology 1).

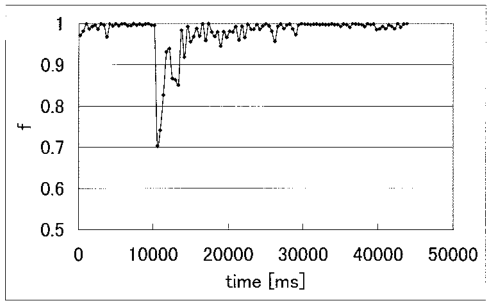


Fig. 11 Measured fairness f for ADU2 (Topology 1).

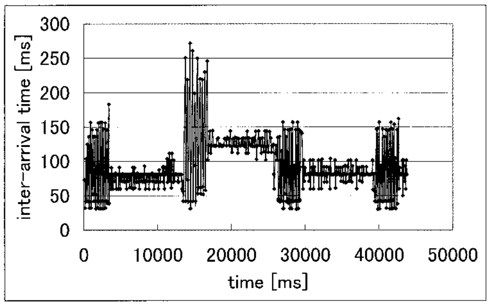


Fig. 12 Measured inter-arrival time of ADUs for ADU2 (Topology 1).

```
   133.27.186.177 ........................................... Host 1
 1 yamabiko (133.27.186.1) 0.580 ms 0.454 ms 0.423 ms
 2 gw11-v3.sfc.keio.ac.jp 1.082 ms 1.018 ms 1.014 ms
 3 fw2-f.sfc.keio.ac.jp 0.962 ms 0.911 ms 0.858 ms
 4 fw1.sfc.keio.ac.jp 1.072 ms 1.074 ms 1.003 ms
 5 wide-keio-p2p.sfc.keio.ac.jp 1.205 ms 1.136 ms 1.192 ms
 6 cisco12.fujisawa.wide.ad.jp 2.162 ms 2.118 ms 2.139 ms
 7 cisco2.otemachi.wide.ad.jp 6.795 ms 6.989 ms 6.809 ms
 8 cisco5.otemachi.wide.ad.jp 6.863 ms 7.974 ms 6.931 ms
 9 tpr-loopback0.jp.apan.net 11.486 ms 9.367 ms 7.850 ms
10 203.181.248.241 145.166 ms 145.634 ms 145.175 ms
11 203.181.248.238 146.154 ms 147.948 ms 146.968 ms
12 cs-atm0-0-11.psc.vbns.net 156.471 ms 158.382 ms 157.331 ms
13 garcia-72.psc.edu 156.966 ms 156.410 ms 156.680 ms
14 140.173.6.78 158.736 ms 159.248 ms 159.234 ms
15 MODERN.ART.CS.CMU.EDU 158.312 ms 157.744 ms 157.673 ms
   .... Host 3
```

Fig. 13 Route from Host 1 to Host 3 obtained by traceroute.

the rate-probing.

Let us observe the transition of the inter-arrival time of ADUs, j_A. **Figure 12** is a snapshot of j_A of Flow 3. Since there is no packet loss at the switching points of periods, we can observe no significant jump in j_A. It is also noted that fluctuation of j_A is larger in the rate-probing period than in the running period. From this figure, it is suggested that UDP is preferable if the rate is controlled in a TCP-friendly manner and there is no packet loss.

Similar experiments were conducted in Topology 2. The path in Topology 2 has a longer round-trip time around 160 ms. The result of executing the *traceroute* program is shown in **Fig. 13**. Measurements were made at the user level of Host 3 with Topology 2, because we could not use a special-purpose ATM card. The results in Topology 2 are shown in **Figs. 14** to **16**. Unlike in the experiment with Topology 1, Flows 1 to 3 traverse the Internet and thus the activeness of Flow 2 does not influence the rates of Flows 1 and 3. From some reason during time 4 s to 6 s, the rate of TCP-controlled Flow 1 falls to 0, which results in a drop in f. Similar curvev of f and j_A are observed in Figs. 15 and 16. However, there is one significant difference: a large j_A at time 27 s. This is caused by packet loss when UDP is used. The loss cuases a decrease in the rate of Flow 3, but this recovers to a stable value in the following rate-probing period. Although this is a phenomenon that has to be explored in detail, it is important to note that Flow 3 automatically runs at a TCP-equivalent rate.

Experiments were also conducted for ADU1. Snapshots of the measured rates and fairness in Topology 2 are shown in **Figs. 17** and **18**. There is no key difference between the results for ADU1 and ADU2.

Next, we investigated the dependence on the number of TCP connections.

Let N_{tcp} denote the number of simultaneous TCP connections from Host 2 to Host 3. We conducted ten trials of 40-s duration in both Topologies 1 and 2. **Figure 19** shows the average values of fairness for ADU1 and ADU2 after the first rate-probing period has finished. As can be seen, TPBA has no dependence on N_{tcp}.

Finally, we observed the response to a change in the availability of bandwidth. In this experiment, we used ADU2 with Flow 3 over Topology 1 and set up two kinds of test cases:

- Case a: Flows 1 and 3 begin at time 0, whereas Flow 2 begins at time 10 s.
- Case b: Flows 1 to 3 begin at time 0, and Flow 2 ends at time 10 s.

Let T_{pr} and T_{pi} denote the elapsed time from

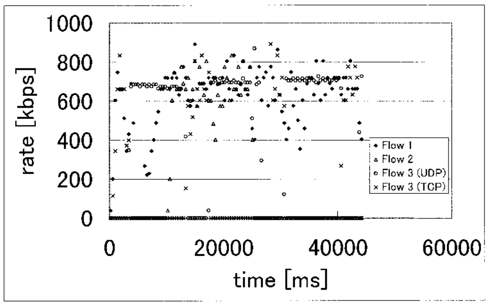


Fig. 14 Measured rates for ADU2 (Topology 2).

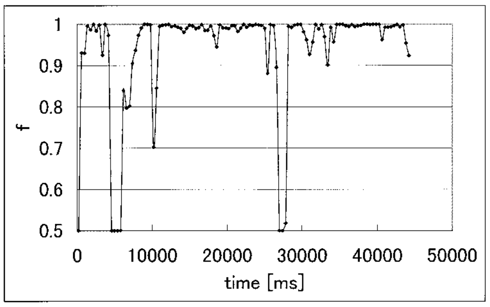


Fig. 15 Measured fairness f for ADU2 (Topology 2).

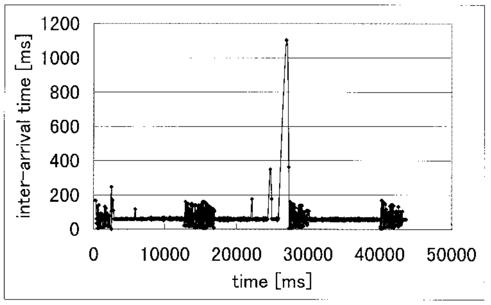


Fig. 16 Measured inter-arrival time of ADUs for ADU2 (Topology 2).

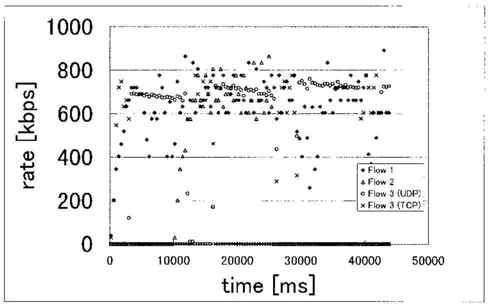


Fig. 17 Measured rates for ADU1 (Topology 2).

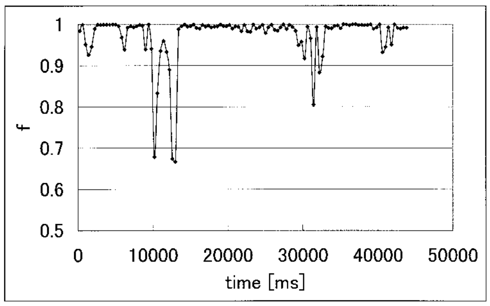


Fig. 18 Measured fairness f for ADU1 (Topology 2).

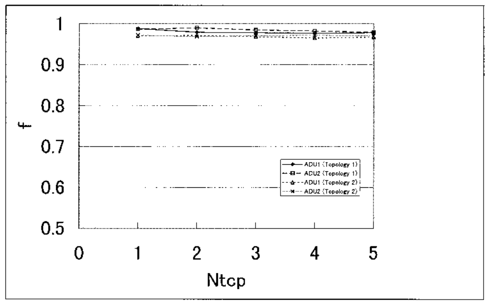


Fig. 19 Number of TCP connections vs. fairness.

Table 3 Response to the change in the availability of bandwidth.

	T_{pr}	T_{pi}
maximum	2.8 s	3.2 s
average	2.0 s	2.8 s

the beginning of Flow 2 until the rate of Flow 3 becomes stable in Case a, and the elapsed time from the end of Flow 2 until the rate of Flow 3 becomes stable in Case b, respectively. Ten trials were conducted for each case and the rates were calculated every 400 ms. The measured results are shown in **Table 3**. They indicate that T_{tcp} can be set to a smaller value than 1 s; a smaller T_{tcp} allows an earlier switch to the running period.

6.2 Comparison between TPBA and LDA

We now compare our scheme with another TCP-friendly scheme, LDA. We also tested RAP[17] by executing a program developed by its author☆, but as stated later, it seems currently premature to compare RAP with other schemes on an implementation basis.

To obtain statistical values through repetitive experiments, we used Topology 1 instead of Topology 2. As traffic for transmission, we used ADU2. As in the previous experiments, Host 1 generates two TCP flows, Flows 1 and 2, to Host 3, while Host 2 generates a UDP flow, Flow 3, which is under the control of ei-

☆ Readers who are interested in using RAP software can contact the author of RAP at <reza@isi.edu>.

Table 4 Compared cases (a: T_{adapt}, b: T_{rtcp}, c: $T_{running}$, d: AIR_0).

Test Case	Scheme	a [s]	b [s]	c [s]	d [kbps]	R_f
TPBA-3-1-10-3	TPBA	3.0	1.0	10	-	3.0
TPBA-3-1-10-6	TPBA	3.0	1.0	10	-	6.0
TPBA-3-1-30-3	TPBA	3.0	1.0	30	-	3.0
TPBA-1-0.5-10-3	TPBA	1.0	0.5	10	-	3.0
TPBA-1-0.5-30-3	TPBA	1.0	0.5	30	-	3.0
LDAF-3-1-10-3	LDAF	3.0	1.0	-	10	3.0
LDAF-3-1-10-6	LDAF	3.0	1.0	-	10	6.0
LDAF-3-1-100-3	LDAF	3.0	1.0	-	100	3.0
LDAF-1-0.5-10-3	LDAF	1.0	0.5	-	10	3.0
LDAF-1-0.5-100-3	LDAF	1.0	0.5	-	100	3.0
LDA-3-1-10-3	LDA	3.0	1.0	-	10	3.0
LDA-3-1-10-6	LDA	3.0	1.0	-	10	6.0
LDA-3-1-100-3	LDA	3.0	1.0	-	100	3.0
LDA-3-1-100-6	LDA	3.0	1.0	-	100	6.0
LDA-1-0.5-10-3	LDA	1.0	0.5	-	10	3.0
LDA-1-0.5-100-3	LDA	1.0	0.5	-	100	3.0

ther TPBA or LDA. In addition, Host 1 creates another TCP flow, Flow 4, to change the traffic load. The initial value of the rate, r_0, is set to 10 kbps as suggested in the literature[20].

To examine longer-term dynamics than in the previous experiments, we set the time sequence of Flows 1–4 as follows:

- At time 0, Flows 1–3 start simultaneously.
- At time 200 s, Flow 4 also starts. Therefore the four flows coexist after time 200 s.
- At time 300 s, all flows are terminated.

To observe the sensitivity of the parameters, several cases were created for TPBA and LDA, as listed in **Table 4**. In the table, LDAF stands for "LDA with a Fixed AIR", The intention of testing LDAF was to investigate the mechanism of updating the value of AIR. More specifically, LDAF replaces line (B) in Fig. 1 with "$AIR = AIR_0 \times (1 - \frac{r}{b})$".

Prior to comparing all the cases, we collected some snapshots of transitions in the rate so as to determine suitable metrics for comparison. Unfortunately, we were confronted with the question of how to define the "rate" itself. For example, we cannot tell whether or not the result of LDAF-1-0.5-10-3 in **Fig. 20** provides almost fair rates without any concrete definition of the rate. Each plot in Fig. 20 represents an average value over a 500-ms period. The rates of Flows 1 and 2 fluctuate almost synchronously, while that of Flow 3 moves in the opposite way, and we can hardly say that they are fairly shared on the time scale of seconds. However, if we drew the figure with a 50-s period average, the rates of Flows 1–3 would not differ from each other by more than 50 kbps. This kind of consideration was not required in the previous experiments, since long-term oscillation was not seen in TPBA.

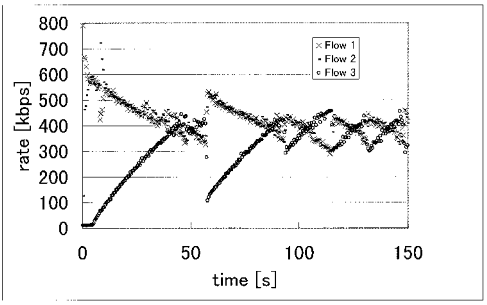


Fig. 20 Snapshot of the rate (LDAF-1-0.5-10-3).

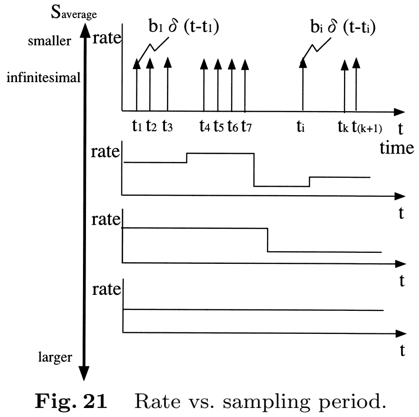


Fig. 21 Rate vs. sampling period.

The difficulty of defining a rate results from the inherent nature of packet arrival; packet arrival is a discrete-time process. At one extreme, the receiving rate is expressed in a series of Delta functions, as shown in **Fig. 21**. In the figure, t_i and b_i represent the time of receiving

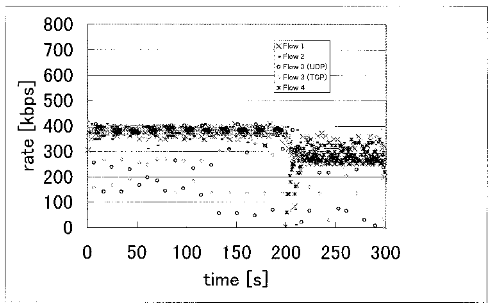


Fig. 22 Snapshot of the rate (TPBA-3-1-10-3: the rate of Flow 3 is obtained by adding the rate of Flow 3 (UDP) to that of Flow 3 (TCP)).

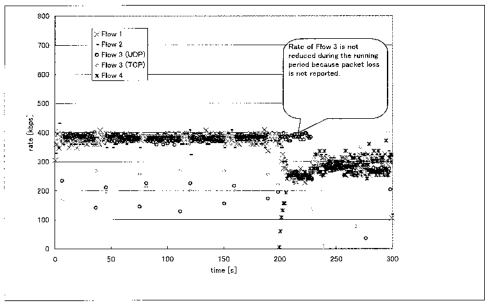


Fig. 23 Snapshot of the rate (TPBA-3-1-30-3).

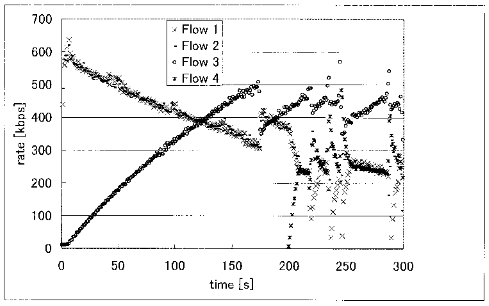


Fig. 24 Snapshot of the rate (LDAF-3-1-10-3).

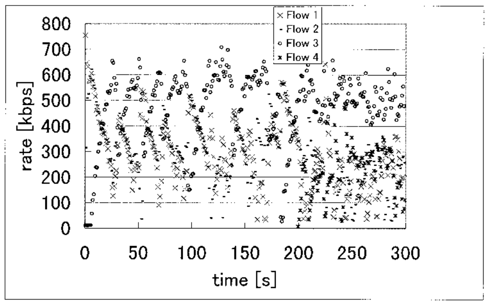


Fig. 25 Snapshot of the rate (LDAF-3-1-100-3).

and the data size of the received packet, respectively. Let $S_{average}$ denote the sampling period of averaging for rate calculation at the receiver. The longer $S_{average}$ is, the smoother the rate becomes, but the more information about the short-term change may be lost. Therefore, we need a tradeoff to determine the value of $S_{average}$.

In the particular case of discussing TCP-friendliness, if $S_{average}$ is a relatively low value at less than 10 ms, an oscillation in the rate due to the window-based and ACK-clocked characteristics of TCP appears even when the TCP flow is stable. For this reason, we set $S_{average}$ to 1 s, which is considered to provide a rate that is smooth enough to suppress the window-based burstiness and also to capture a significant change.

Having determined the value of $S_{average}$, we now examine snapshots of rates for other cases. We focus on how the parameters $T_{running}$, T_{adapt}, R_f, and AIR affect the fairness property. **Figures 22** and **23** correspond to TPBA-3-1-10-3 and TPBA-3-1-30-3, respectively. As can be seen in the figures, the response to the change at time 200 s becomes worse with a larger value of $T_{running}$. However, the damage to TCP flows, until the rate-probing is triggered, is minimized, since the rate of Flow 3 is not increased.

Unlike $T_{running}$, T_{adapt} hardly affects the behavior with TPBA. The behavior with TPBA-3-1-10-6 is almost the same as that with TPBA-3-1-10-3. In contrast, T_{adapt} strongly affects the behaviors with LDAF and LDA. In the rate-increasing phase of LDAF, the slope of the increase is inversely proportional to the value of T_{adapt}. This can be recognized by comparing **Figs. 24** and **25**. In LDA, the behavior differs slightly, since the value of AIR changes at each adaptation point, but the smaller value of T_{adapt} accelerates the convergence (**Figs. 27** and **28**). Despite this advantage, updating the rate frequently necessitates more bandwidth for RTCP messages and requires a trade-off.

The value of 3.0 for R_f was suggested on the basis of extensive simulations in the literature[20]. We also tested a case with a value of 6.0. However, there was no significant difference between the two in TPBA, LDAF, and LDA. We revisit the influence of R_f on the fairness later on the basis of the calculated values of metrics.

Finally, we discuss the selection of the value of AIR_0. Figures 24–27 show the difference of AIR_0 in LDAF and LDA. As expected, a large value of AIR causes oscillation in the rate in LDAF, since AIR is not iteratively reduced. In

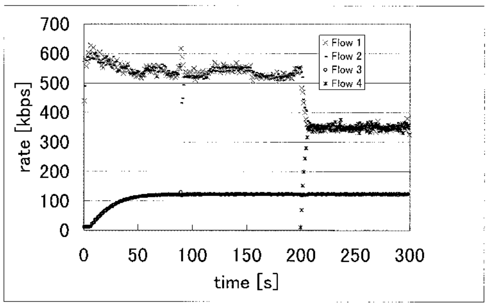


Fig. 26 Snapshot of the rate (LDA-3-1-10-3).

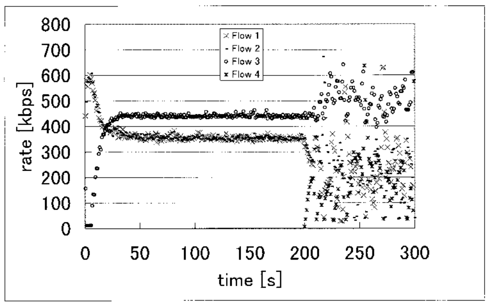


Fig. 27 Snapshot of the rate (LDA-3-1-100-3).

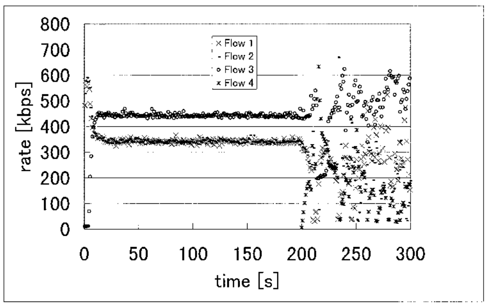


Fig. 28 Snapshot of the rate (LDA-1-0.5-100-3).

this sense, the idea contained in Line (B) of Fig. 2 is important. This enables a faster raising of the rate at the beginning or after a heavy loss, and a slowing down of the update after iterations. However, as can be seen in **Fig. 26**, a smaller value may cause the rate to stay at an unexpectedly low value, whereas a larger value may result in it remaining at an overshot value (Fig. 27). From a comparison of cases with different AIR_0, it can be said that LDA is very sensitive to AIR_0 and that the algorithm for updating the value of AIR requires improvements.

Quantitative Comparison

We now define metrics for comparing all the cases. Let $R_i(j)$ denote the rate over $[S_{average} \times j,\ S_{average} \times (j+1)]$ for flow i. Then, let n_{cross} and n_{join} represent the minimum value of j such that $R_3(j) \geq R_1(j)$ or $R_3(j) \geq R_2(j)$ and the minimum value of j such that $R_4(j) \geq R_1(j)$ or $R_4(j) \geq R_2(j)$, respectively. In addition, let n_{change} and n_{final} be $(200\,\text{s}/S_{average} - 1)$ and $(300\,\text{s}/S_{average} - 1)$, respectively. Under these definitions, we define δR_{intra}, δR_{ave1}, and δR_{ave2} as follows:

$$\begin{aligned}
&\delta R_{intra} \\
&\quad \equiv \max_{i=1,2,3}(\max\nolimits_{n_{cross} \leq j \leq n_{change}} R_i(j) \\
&\qquad - \min\nolimits_{n_{cross} \leq j \leq n_{change}} R_i(j)),
\end{aligned}$$

$$SR1_i = \frac{\sum_{j=n_{cross}}^{n_{change}} R_i(j)}{n_{change} - n_{cross} + 1},$$

$$SR2_i = \frac{\sum_{j=n_{join}}^{n_{final}} R_i(j)}{n_{final} - n_{join} + 1},$$

$$\delta R_{ave1} \equiv \max_{i \neq j} |SR1_i - SR1_j|,$$

$$\delta R_{ave2} \equiv \max_{i \neq j} |SR2_i - SR2_j|.$$

Intuitively, δR_{intra}, δR_{ave1}, and δR_{ave2} provide the magnitude of fluctuation in a steady state, the difference in the average rate of flows in a steady state, and the difference in the average rate of flows after traffic change.

We conducted ten trials for each case and calculated δR_{intra}, δR_{ave1}, and δR_{ave2}. We dealt with LDAF-3-1-10-3 and LDAF-3-1-10-6 differently in calculating δR_{intra} and δR_{ave1}. These two cases do not reach a repetitive cycle at time 200 s. Therefore we collected rates over 400 s instead of 200 s to calculate the two metrics. In addition, since LDA-3-1-10-3, LDA-3-1-10-6, and LDA-1-0.5-10-3 do not provide n_{cross}, δR_{intra} and δR_{ave1} for them were calculated with the minimum vaule of n that satisfies the following condition:

$$\forall i, j \epsilon [n-10, n] : |R_3(i) - R_3(j)| < 20\,\text{kbps}.$$

Figures 29–31 show δR_{intra}, δR_{ave1}, and δR_{ave2} for all cases, respectively. Since LDAF was tested for the purpose of reference, we focus solely on a comparison of TPBA and LDA. In every metric, TPBA outperforms LDA. In particular, LDA does not converge to a TCP-friendly rate after a traffic change. We also cannot observe a significant difference between the cases in which R_f is 3.0 and 6.0. It can be concluded from this comparison that TPBA is robust in traffic change because a TPBA-capable flow reaches a TCP-friendly rate after the change.

Finally, we conducted the same experiment using RAP instead of LDA. The result with the current implementation by Rejaie, et al.[17)]

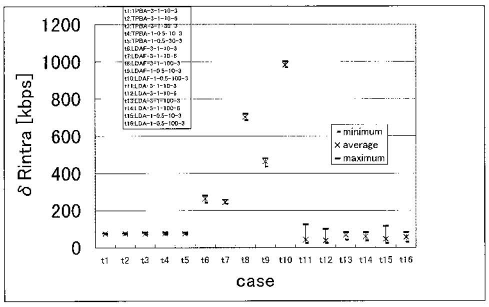


Fig. 29 Calculated δR_{intra}.

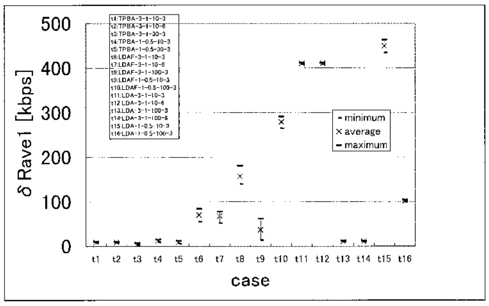


Fig. 30 Calculated δR_{ave1}.

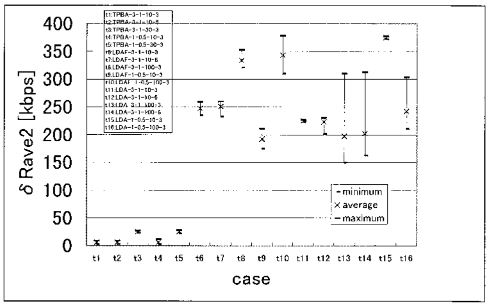


Fig. 31 Calculated δR_{ave2}.

is shown in **Fig. 32**. Although fairness is suggested in the simulation[17], the result is far from exhibiting fairness. We are currently investigating the cause of the problem. One thing we noticed is the RAP algorithm's strong dependence on calculating the RTT. Unlike in common implementations of TCP, the RTT is calculated at the user level in the current implementation of RAP. We are afraid that unpredictable system calls with a large overhead may cause inaccurate RAP behavior. In order to make a fair comparison, we are enhancing the implementation.

7. Discussion

Switching protocols may cause an extra overhead in processing and possible corruption of ADU at the boundaries of switching. Using

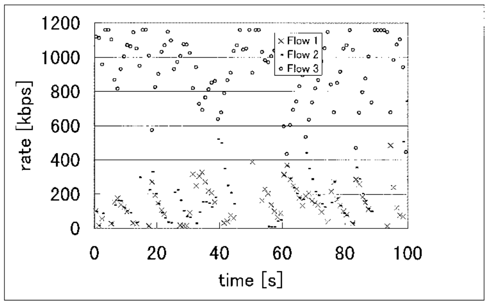


Fig. 32 Snapshot of the rate with RAP (We used the author's implementation. We plan to change the implementation for future comparison).

TCP entirely instead of UDP may be an alternative. However, the delay performance of TCP is often poor, it is desirable to use UDP whenever possible. As can be seen in Figs. 13 and 16, UDP is superior in terms of reducing jitter. One drawback of our approach is that it limits unicasting. However, a TCP-friendly algorithm for unicasting has not yet been established and should still be investigated.

We have not investigated any cases with multiple concurrent RTP/UDP flows. Such a case is complicated than that with a single RTP/UDP flow. However, as long as each flow uses rate-probing, the rate of each flow is adjusted in the rate probing period even when a concurrent RTP/UDP flow is dynamically created or terminated.

The algorithm of TPBA does not include critical parameters that influence long-term fairness with TCP, such as *AIR* of LDA. However, there is a possibility of instability caused by misconfiguration of T_{TCP} and ε. In one case, when T_{TCP} and ε are set to 1 ms and 1 kbps, respectively, the rate-probing period may not be completed. In another case, when they are set to 1 ms and 10 Mbps, the measured rate during the rate-probing period is not trustworthy.

The solution to this is to embed a rate-monitoring module inside the TCP sender. In TCP-Reno, the most popular version of TCP, *cwnd* at the sender is halved when the sender receives three duplicate acknowledgments triggered by a packet loss. After the reduction of *cwnd*, the sender increases *cwnd* every time it receives an acknowledgment. Although the TCP sender behaves differently when a retransmission timeout occurs under heavy congestion, the TCP sender usually repeats this cycle of increasing and decreasing *cwnd* in the steady-state. This mechanism determines the rate of the TCP. Therefore, instead of a fixed value of T_{TCP} at the user library, the period for calculating the TCP rate can be decided by monitoring the interval between the time when *cwnd* is reduced. The period can vary dynamically.

Criterion for determining the end of the rate-probing period should be also changed. It is more natural that fairness should be evaluated according to the proportion of difference rather than its absolute value: a 100-kbps difference affects 300-kbps flows much more than 10-Mbps flows. For this reason, instead of ε, we define an allowable difference of $z\%$ (e.g., 5%) in our design. The embedded rate-monitoring module can then determine whether the fluctuation of the rate is within this range.

We intended to minimize the modification to the kernel when we begin designing TPBA. However, the current scheme cannot avoid misconfiguration of T_{TCP} and ε. Therefore, we plan to develop an embedded rate-monitoring module inside the kernel.

8. Future Work

We have several plans for future work. First, we plan to use a network simulator. In the work described here, we conducted experiments over only two types of networks. Although the results of the experiments show that our scheme is effective in the tested networks, we plan to investigate the performance on an ns-2 [14] simulator to generalize the evaluation. Second, we will extend the comparison with other schemes. It is still unknown whether either the software or the algorithm of RAP has any problems. Since the simulation described in the literature [17] shows a good result, we currently doubt about the implementation of the software and are investigating its details. Implementation-based comparison with the congestion manager is also within our future scope. Finally, we will develop the embedded rate-monitoring module described in the previous section.

9. Concluding Remarks

In this paper, we have proposed a new mechanism for adapting CM flows on the basis of TCP-rate probing. Our scheme, TPBA, focuses on the rate control of CM flows by probing for the TCP-exact rate. We have implemented TPBA on FreeBSD PCs and evaluated the effectiveness of TCP-friendliness over our own load-controlled network and public networks. Experimental results have shown that a TPBA-capable flow is adaptive to a TCP-equivalent rate and that there is no significant degradation in performance at the boundaries of the running and rate-probing periods. We have also compared TPBA with LDA extensively and shown that TPBA outperforms LDA in achieving TCP-friendliness.

In future work, we plan to compare TPBA with other schemes, through simulation as well as implementation, and to develop a rate-monitoring module inside the kernel.

Acknowledgments This research is conducted under the fund of Information-technology Promotion Agency, Japan (IPA). The authors are grateful to members of the Keio MKng/R3 Project and Carnegie Mellon University Real-Time and Multimedia Laboratory for valuable comments and criticisms. Special thanks to Seiji Kihara and Yasunori Matsui of NTT, and Sourav Ghosh of Carnegie Mellon University for their assistance in our experiments. Finally, we appreciate Reza Rejaie's kindness in providing us with RAP software.

References

1) Balakrishnan, H., Rahul, H. and Seshan, S.: An integrated congestion management architecture for Internet hosts, *ACM SIGCOMM '99* (1999).

2) Blake, S., Black, D., Carlson, D., Davies, E., Wang, Z. and Weiss, W.: An architecture for differentiated services, RFC 2475 (1998).

3) Bolot, J.C.: Characterizing end-to-end packet delay and loss in the Internet, *ACM SIGCOMM '93* (1993).

4) Chiu, D. and Jain, R.: Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, *Computer Networks and ISDN Systems*, Vol.17, pp.1–14 (1989).

5) Clark, D.D. and Tennenhouse, D.L.: Architectural consideration for a new generation of protocols, *ACM SIGCOMM '90*, pp.200–208 (1990).

6) Demers, A., Keshav, S. and Shenker, S.: Analysis and simulation of fair queueing algorithm, *ACM SIGCOMM '89*, pp.1–12 (1989).

7) Floyd, S. and Jacobson, V.: Link-sharing and resource management models for packet networks, *IEEE/ACM Trans. Networking*, Vol.3, No.4 (1995).

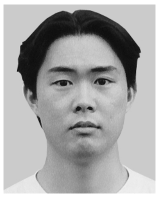
8) Henderson, T.R., Sahouria, E., McCanne, S. and Katz, R.H.: On improving the fairness of TCP congestion avoidance, *GLOBECOM '98* (1998).

9) Jain, R.: A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks, *Computer Communication Review*, Vol.19, No.5, pp.56–71 (1989).
10) Keshav, S.: A control-theoretic approach to flow control, *ACM SIGCOMM '91* (1991).
11) Lai, K. and Baker, M.: Measuring bandwidth, *INFOCOM '99*, pp.235–245 (1999).
12) Mahdavi, J. and Floyd, S.: TCP-friendly unicast rate-based flow control, http://www.psc.edu/networking/papers/tcp_friendly.html.
13) Mathis, M., Semke, J., Mahdavi, J. and Ott, T.: The macroscopic behavior of the TCP congestion avoidance algorithm, *Computer Communication Review*, Vol.27, No.3 (1997).
14) ns-2, http://www-mash.cs.berkeley.edu/ns.
15) Padhye, J., Kurose, J. and Towsley, D.: A TCP-friendly rate adjustment protocol for continuous media flows over best effort networks, University of Massachusetts at Amherst CMPSICS Technical Report, TR98-047 (Oct. 1998).
16) Paxson, V.: Measurements and analysis of end-to-end Internet dynamics, Ph.D. Thesis, University of California, Berkeley (Apr. 1997).
17) Rejaie, R., Handley, M. and Estrin, D.: RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet, *INFOCOM '99* (1999).
18) Schulzrinne, H., Casner, S., Frederick, R. and Jacobson, V.: RTP: A transport protocol for real-time applications, RFC 1889 (1996).
19) Shenker, S., Partridge, C. and Guerin, R.: Specification of guaranteed Quality of Service, RFC 2212 (1997).
20) Sisalem, D. and Schulzrinne, H.: The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme, *NOSSDAV '98*, pp.215–226 (1998).
21) http://www.psc.edu/networking/tcp_friendly.html.
22) Turletti, T., Parisis, S. and Bolot, J.: Experiments with a layered transmission scheme over the Internet, Technical Report RR-3296, INRIA, France (1997).
23) Vicisano, L., Rizzo, L. and Crowcroft, J.: TCP-like congestion control for layered multicast data transfer, *INFOCOM '98* (1998).

(Received April 30, 1999)
(Accepted December 2, 1999)

Yoshito Tobe is a research staff at Keio Research Institute at SFC, where he is currently studying QoS-aware protocol processing implementation. He is a member of the IEEE Communications Society, the ACM, and the IEICE.

Yosuke Tamura received his B.S. degree in environmental information from Keio University in 1997. He received his M.A. degree in Media and Governance from Keio University in 1999. He is a Ph.D. candidate at graduate school of Media and Governance, Keio University. He is currently studying flow control for internetworked data communications. He is a member of the IEEE Computer Society and the IEICE.



Hideaki Nishino is a bachelor's course student at department of Environmental Information, Keio University. He is currently studying packet scheduling algorithms for mobile computing environments.

Hideyuki Tokuda received his B.S. and M.S. degrees in electrical engineering from Keio University in 1975 and 1977, respectively; a Ph.D. degree in computer science from the University of Waterloo in 1983. He joined the School of Computer Science at Carnegie Mellon University in 1983, and is an Adjunct Associate Professor from 1994. He joined the Faculty of Environmental Information at Keio University in 1990, and is a professor since 1996. His current interests include distributed operating system and computer networks. He is a member of the ACM, the IEEE, the IEICE, and the JSSST.