Vol. 43 No. 6 IPSJ Journal June 2002

Recommended Paper

Analysis of TCP Throughput Anomaly over Congested ATM Networks

Motoki Nakanishi,[†] Ishtiaq Ahmed,[†] Hayato Ishibashi,[††] Yasuo Okabe[†] and Masanori Kanazawa[†††]

ATM is one of the key technologies for high-performance networks. TCP is most popular protocol used in these networks and the Internet. Throughput of TCP on IP over ATM networks suffers congestion and deadlock due to their different characteristics. In this paper, we have investigated factors that includes; ATM switch buffer size, TCP window size, MTU size and external ATM traffic. We have studied TCP over UBR streams running with CBR streams, which make ATM switch congested and lead to heavy cell loss situation. We have proved that TCP data transfer on ATM congested link suffers throughput deadlock due to switch buffer size, MTU size, TCP window size, and CBR streams. We have explained the TCP throughput deadlock anomaly and have proved that TCP's performance is less than 2% of the available bandwidth over plain ATM.

1. Introduction

The ATM (Asynchronous Transfer Mode) networks provide QoS (Quality of Service) to multimedia application, like audio/video data transfer and video conferencing. It is one of the key technologies for high-speed networks. Many campus networks in Japan have been adopting ATM LANs (Local Area Networks) as their backbone[10)].

TCP (Transport Control Protocol) has got much importance and dominance as the protocol in these networks and today's Internet, but TCP does not necessarily fit well with any high-speed network. Rather, TCP on IPoA (IP over ATM)[16),17)] is known to have several implementation problems. The large MTU size of ATM[3),17)] causes considerable degradation of the performance in TCP data transfers[21)]. ATM data contains payload of 48 [bytes] and TCP does fragmentation that causes cell loss at the ATM level. One cell loss at ATM level heavily affects the performance at the protocol level[24)].

In this paper, we have investigated traffic dynamics of TCP over UBR (Unspecified Bit Rate) streams running with CBR (Constant Bit Rate) streams, which make ATM switch congested and lead to heavy cell loss situation. While the simulation results shown in Ref. 24) are derived using traffic of the same precedence (UBR), we focus ourselves on the dynamics of TCP over congested ATM links where stream data, typically high-density video streams, occpies most of the available bandwidth with absolutely high precedence (CBR).

We first show that TCP data transfer suffers throughput deadlock because of ATM switch buffer size, MTU size, and CBR pressure traffic. Time-outs occur too frequently in TCP, and in the worst case the performance is less than 2% of the available bandwidth over plain ATM. We have tried to reveal throughput characteristics of IP over ATM traffic. We have explained the TCP throughput deadlock anomaly and have done extensive experimentation with other parameters for its analysis.

In Section 2, we describe protocols TCP/IP, ATM and discussed problems related to performance degradation of TCP throughput. In Section 3, the experimental design is described. In Section 4, we focused on TCP throughput deadlock and analyzed it. In Section 5, the other effective deadlock parameters are described. In Section 6 conclusions are presented.

2. TCP/IP over ATM

Before discussing the problems, let us briefly go through these protocols.

2.1 TCP/IP

TCP/IP suit is the protocol for the Inter-

† Graduate School of Informatics, Kyoto University
†† Media Center, Osaka City University
††† Data Processing Center, Kyoto University

The initial version of this paper was presented at the DSM simposium 2000 held on Feb. 2000, which was sponsored by SIGDSM. This paper was recommended to be submitted to the Journal of IPSJ by the chairperson of SIGDSM.

net. IP (Internet Protocol) provides an unreliable and connectionless datagram delivery service with a simple error-handling algorithm[27]. TCP (Transport Control Protocol) was specifically designed to provide a reliable end-to-end block (set of bytes) stream over an unreliable network[28]. Networks are different because they may have different topologies, bandwidths, delays, packet sizes, and other parameters. TCP was designed to adapt dynamic properties of the internetwork and to be robust in the face of failures. Each machine supporting TCP has a TCP transport entity, either a user process or a part of the kernel that manages TCP streams and interfaces to IP layer. A TCP entity accepts user data stream from local processes, breaks them up into pieces not exceeding 64 K bytes and sends each piece as a separate IP datagram. When IP datagrams containing TCP data arrive at a machine, they are given to the TCP entity, which reconstructs the original block streams. This TCP entity is called TCP protocol. A TCP connection is a block stream, not a message stream. Message boundaries are not preserved end to end. The sending and receiving TCP exchange data in the form of segments. A segment consists of a fixed 20-byte header (plus an optional part) followed by zero or more data bytes.

The TCP software decides how big segments should be. It can accumulate data from several writes into one segment or split data from one write over multiple segments. Two limits restrict the segment size.

- Each segment, including the TCP header, must fit in the 65,535 bytes IP payload.
- Each network has a maximum transfer unit or MTU, and each segment must fit in the MTU.

The basic protocol used by TCP is the sliding window protocol. When a sender transmits a segment, it also starts a timer. When the segment arrives at the destination, the receiving TCP sends back a segment (with data if any exists, otherwise without data) bearing an acknowledgement ACK number equal to the next sequence number it expects to receive. If sender's timer goes off before the ACK is received, the sender transmits the segment again.

2.2 ATM

Asynchronous Transfer Mode (ATM) is a connection-oriented protocol that provides four basic service classes namely ABR (Available Bit Rate), CBR (Constant Bit Rate), real-time & non real-time VBR (Variable Bit Rate), and UBR (Unspecified Bit Rate).

The unit of transport in ATM is a 53 byte fixed length PDU (Protocol Data Unit) called a cell. A cell consists of a 5 bytes header and a 48-byte payload. Variable length PDUs, must be segmented by the sender to fit into the 48-byte ATM cell payload, and reassembled by the receiver. The header field, in each cell, mainly used to determine the virtual channel and to perform the appropriate routing. Cell sequence integrity is preserved per virtual channel. The header values are assigned to each section of a connection for the complete duration of the connection. Signaling and user information are carried on separate channels. All services (voice, video, and data) can be transported via ATM, including connectionless services. To accommodate various services an adaptation function is provided to fit information of all services into ATM cells and to provide services specific functions. For computer communication the recommended end-to-end ATM service is through an ATM Adaptation Layer (AAL), either AAL3/4 or AAL5[15]. The adaptation layer aggregates cells into much larger data units that are more efficiently handled by higher layers and match the application data units better. The TCP/IP protocols use the size of these AAL data units (ATM network MTU) to compute MSS (Maximum Segment Size). The MSS is defined as MTU − 40 (TCP/IP header)[6],[25]. The AAL5 payload is 9,180 and TCP/IP header is 40. According to Ref. 25), $MSS = 9{,}180 - 40$.

But we find in our experiments by Tcpdump software[29] that MSS is fixed by the TCP protocol and does not follow the above formula for different values of MTU. **Figure 1** shows the relationship between MSS and MTU.

2.3 Previous Works on Performance Degradation of TCP/IP over ATM

The following problems have been discussed in various publications.

Packet fragmentation can result in wasted bandwidth and packet retransmission[24]. Because the network might drop only one fragment from a packet, "the loss of any one fragment means that resources expended in sending other fragments of that datagram are entirely wasted"[15]. It is shown that smaller switch buffer and larger TCP packet sizes (MTU in our case) both reduce the effective throughput for TCP over plain ATM[24]. Effective throughput is defined as the throughput that is "good"

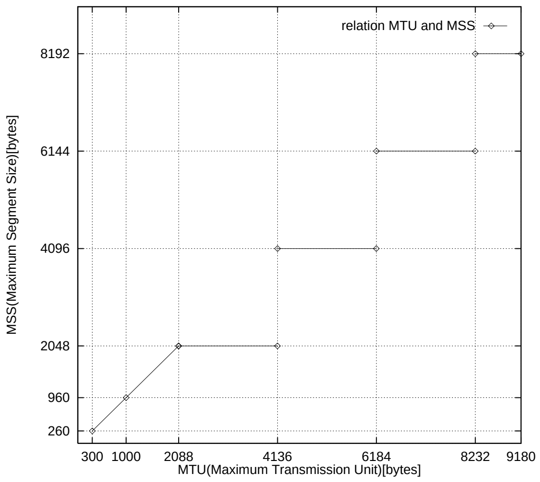


Fig. 1 Relationship between MSS and MTU.

in terms of the higher-layer protocol. TCP window size also affects the effective throughput of TCP over plain ATM. TCP window size ranging from 8 Kbytes to 64 Kbytes show a lower effective throughput with larger window.

The drastic drop in performance is caused by a deadlock situation in the TCP connection, which is broken up by the 200 ms timer generated TCP acknowledgment[22]. It causes TCP to behave as a stop-and-go protocol with one or two data segments sent every 200 ms. The deadlock occurs when the amount of data sent is not enough to trigger a TCP window update packet at the receiver, and at the same time there is not enough space in the send buffer to create a segment of size MSS bytes. Nagle's algorithm prohibits the sending of non-MSS segments if there are unacknowledged bytes. Since TCP piggybacks acknowledgments onto window updates, the connection is deadlock until the receiver sends a timer generated acknowledgment.

Buffer size in ATM switch may suffer overflow during congestion and suffer cell loss. Increase in buffer size is not the solution for this problem as buffer is going to be overflowed again making the equipment more costly. The cell loss problem has given rise to Early Packet Discard EPD[24] and Partial Packet Discard PPD[2]. In PPD, if a cell is dropped from a switch buffer, the subsequent cells in the higher layer protocol data unit are discarded. In EPD, when the switch buffer queues reach a threshold level, entire higher-level data units (e.g., TCP/IP packets) are dropped.

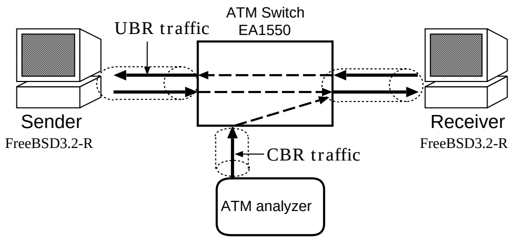


Fig. 2 Experimental configuration.

3. Experimental Design and Method

3.1 Experimental Design

We are assuming a congested ATM link where:

- Only two flows exist. One is CBR (Constant Bit Rate) flow, which has absolutely high precedence, and the TCP flow (which is on UBR- Un-specified Bit Rate) is getting pressure of the CBR streams.
- The cell buffer size in the ATM switch for UBR is relatively of small size. It is smaller than window size of TCP, although enough larger than a packet.

This is a very typical situation where a stream data (like MPEG video stream) and ordinary TCP flow are get encountered on ATM switch make congestion. TCP data transfer suffers several timeouts due to heavy cell loss at ATM level.

The experimental configuration is shown in **Fig. 2**. Two permanent virtual channels (PVC) have been created in the ATM switch, which are used by two computers. The ATM switch, Fujitsu EA1550, has output-buffer architecture and we can change the size of allocated cell buffer to each port. No packet discarding schemes have been used within ATM switch. ATM traffic analyzer, Hewlett Packard's HP E5200A, can generate CBR traffic and monitor the traffic of these two PVC's. These PVC's are used for UBR and CBR streams.

The FreeBSD 3.2 is installed on each system that has TCP Reno[13], with a few other features like the TCP timestamp option, window scaling, and T/TCP.

3.2 Experimental Method

We have made use of Netperf[14] as a tool, a benchmark that can be used to measure various aspects of networking performance. Its primary focus is on bulk data transfer and request/response performance using either TCP or UDP and the Berkeley Sockets interface. We injected CBR streams from the ATM analyzer's

Table 1 System equipment specifications.

CPU	Pentium II 266 MHz
M/B	P2L97
Memory	SD-RAM 64 Mbytes 10 ns
ATM NIC	ENI-155p
Operating System	FreeBSD3.2-Release
ATM Driver	HARP

traffic generator. TCP's data transfer is over UBR streams, through netperf running for 10 sec and the output of netperf gives the TCP's effective throughput over UBR streams. UBR streams and CBR streams are active on the same port. We can set send/receive socket size as command line arguments in the netperf. For the throughput analysis we have used Tcpdump software, which dumps statistics on the segments sent by the sender towards the host. The equipment listing and specifications are given in **Table 1**.

4. TCP Throughput Deadlock

4.1 TCP Throughput Affected by Cell Buffer Size

In the following experiment we used CBR streams of 60 M[bps] and measured the UBR traffic throughput as a function of cell buffer size. MTU sizes were selected as 512, 1,500, 4,352, 6,500, 9,180 [bytes] while the socket buffer size is 64 K[bytes] as shown in **Fig. 3**.

When the cell buffer size is 150 K[bytes] (3 K cell) or more no throughput deadlock occurs. For example, the throughput is 82.94 M[bps] when MTU = 9,180 [bytes]. When cell buffer size goes down from 150 K[bytes], throughput sharply decreased. When cell buffer is 50 K[bytes] (1 K cell) or less, UBR traffic throughput is less than 2% of the available bandwidth. Below this point the value of TCP's throughput is referred as deadlock. Simulation results have been shown for a number of TCP connections using UBR traffic over plain ATM, throughput as function of cell buffer size[24]. It is proved that 1) Throughput for a number of TCP connections over plain ATM is about 10 $\sim$ 30% when cell buffer size is equal to 1 K cell and 2) TCP's throughput suffers performance degradation with large MTU sizes. We found that:

- TCP over UBR streams suffers deadlock with the CBR streams.
- The larger is the MTU size, the better is the TCP throughput. TCP performs better for MTU 9,180 than for MTU 1,500.

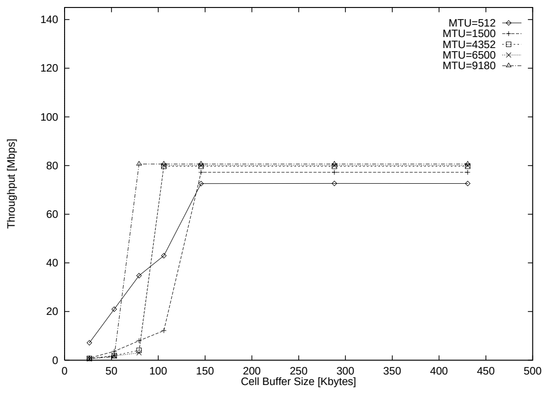


Fig. 3 Throughput versus switch buffer size, TCP window 64 K and CBR 60 Mbps.

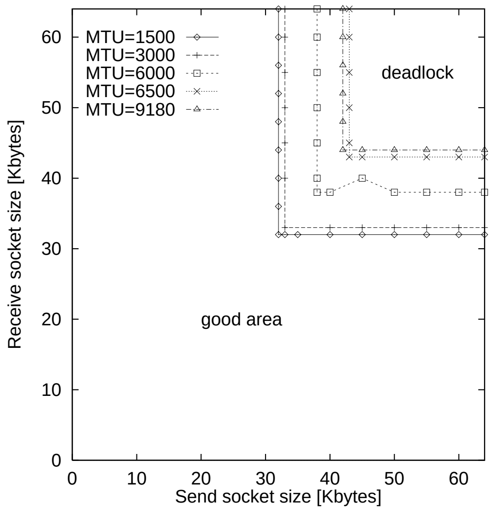


Fig. 4 Throughput versus socket buffer size.

4.2 Throughput Affected by TCP Window Size

The TCP window size is directly related with the size of the sender/receiver socket size option used in the netperf. **Figure 4** makes it clear that effectiveness of TCP window quite depends upon socket size. We have plotted send & receive socket sizes along x- & y-axis respectively and the points correspond to the UBR traffic throughput. CBR traffic is 60 M[bps] and MTU sizes are 1,500, 3,000, 6,000, 6,500, and 9,180 [bytes] while cell buffer size is 1 K cell. Above each line deadlock always occurs, no deadlock occurs below the lines (shown in Fig. 4). We mean by good that when throughput of UBR streams is more than 10% of the available bandwidth. Fig. 4 shows that an increase in socket buffer size brings increase in deadlock area. Both MTU size and socket size

Table 2 Patterns selected for analysis.

CBR [Mbps]	60	
MTU [bytes]	1,500	
Socket Size [Kbytes]	25	38
State	Good	Deadlock
Pattern	(a)	(b)

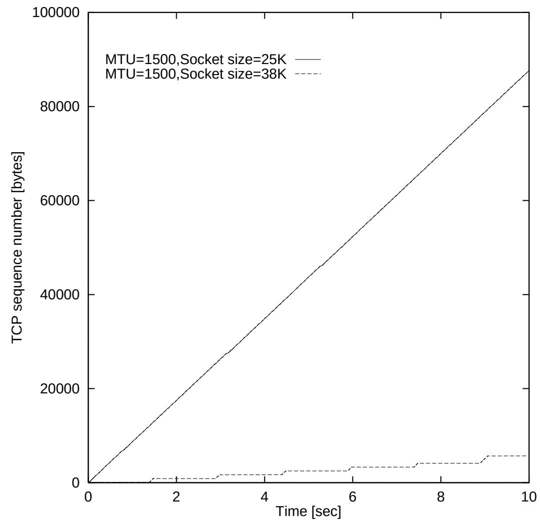


Fig. 5 Segment's sequence number versus time.

are large enough during deadlock situation.

4.3 Analysis of Deadlock Mechanism

To analyze the TCP's throughput deadlock for the same MTU, we used Tcpdump[29] software. This software dumps statistics on the segments sent by the sender towards the host. We selected two patterns mentioned in **Table 2** and did the Tcpdump test for these patterns. **Figure 5** show relation between time at which the segments are sent by the sender and the sequence number of the segments. It shows patterns (b) as step line that represents deadlock.

After analyzing the Tcpdump output for pattern (b), we could find that:

(1) TCP algorithm retransmits a lost packet if the sender receives three duplicate acknowledgments or the time-out for the lost packet is generated[27]. Time-out is generated in a time lag that causes poor throughput. Time-out happens seven times in whole span of 10 seconds (approximately).

(2) Before the time lag occurs, sender sent the packet and received the ACK from receiver with a window size smaller (by one MSS) than the usual advertised window. Here the sender is waiting for new acknowledgment and not sending, on the other hand receiver is waiting for the new segment. Both are waiting and not sending any thing, so it occurs deadlock during this time. After retransmission time out, the normal procedure of sending data starts once again. The steps in the figures are due to retransmission time outs.

A question arises here that why both sender and receiver are waiting and sending nothing. When an ACK advertise a window smaller than usual, following may be the possible reasons[27]. When a packet arrives at the receiver side then it is initially processed by the device driver's interrupt service routine. It is then placed onto IP's input queue. Thus segments arrived one after the other are placed onto IP's input queue in the received order. IP will pass them to TCP in the same order. When TCP is processing the segments then the connection is marked to generate a delayed ACK. The ACK advertising smaller window size implies that there are still some bytes of data in the TCP receiver buffer that application has not read.

We have found that there is a segment loss with pattern (b) and no segment loss with pattern (a). Transmission fashion with pattern (a) & (b) is that sender receives one ACK and then sends two segments after handshaking and initial stages of data transfers. Pattern (a) most of the time follow this fashion. While on the other hand in pattern (b), sender sends three segments upon receiving one ACK.

Consider the patch shown in **Fig. 6** of Tcpdump out file for socket size 38 KB. Here the first time out location has been explained. The transmission fashion here, sender receives one ACK and sends two segments. The sender has sent 26 segments (line 1 to 36) until it receives ACK (line 37) for segment 99,281 (line 1), which is expected by the sender. Sender will process this ACK and came to know that data until 99,280 bytes has been acknowledges by the receiver. Sender then sends two more segments (lines 38, 39). Sender now has sent 27 segments, which is the maximum limit (39,420/1,460=27) of the sender and receiver buffer sizes because the window size is 39,420. Thus TCP receiving entity is processing these segments setting the connection to generate the delayed ACK every time it process the new segment. From line No.40 to 42 sender receives duplicate ACKs for 99,281. This shows segment has lost and sender retransmits the segment

```
1.      16:55:54.093915 Sender.1264 > Recver.4785: . 97821:99281(1460) ack 1 win 39420 (DF)
2.      16:55:54.093982 Sender.1264 > Recver.4785: P 99281:100741(1460) ack 1 win 39420 (DF)
3.      16:55:54.094110 Recver.4785 > Sender.1264: . ack 70081 win 39420 (DF)
4.      16:55:54.094225 Sender.1264 > Recver.4785: . 100741:102201(1460) ack 1 win 39420 (DF)
5.      16:55:54.094283 Sender.1264 > Recver.4785: . 102201:103661(1460) ack 1 win 39420 (DF)
6.      16:55:54.094412 Sender.1264 > Recver.4785: . 103661:105121(1460) ack 1 win 39420 (DF)
7.      16:55:54.094481 Sender.1264 > Recver.4785: P 105121:106581(1460) ack 1 win 39420 (DF)
8.      16:55:54.094511 Recver.4785 > Sender.1264: . ack 73001 win 39420 (DF)
9.      16:55:54.094611 Sender.1264 > Recver.4785: . 106581:108041(1460) ack 1 win 39420 (DF)
10.     16:55:54.094721 Recver.4785 > Sender.1264: . ack 75921 win 39420 (DF)
11.     16:55:54.094769 Sender.1264 > Recver.4785: . 108041:109501(1460) ack 1 win 39420 (DF)
12.     16:55:54.094869 Sender.1264 > Recver.4785: . 109501:110961(1460) ack 1 win 39420 (DF)
13.     16:55:54.094904 Sender.1264 > Recver.4785: P 110961:112421(1460) ack 1 win 39420 (DF)
14.     16:55:54.095018 Sender.1264 > Recver.4785: . 112421:113881(1460) ack 1 win 39420 (DF)
15.     16:55:54.095098 Recver.4785 > Sender.1264: . ack 78841 win 39420 (DF)
16.     16:55:54.095189 Sender.1264 > Recver.4785: P 113881:115341(1460) ack 1 win 39420 (DF)
17.     16:55:54.095277 Sender.1264 > Recver.4785: . 115341:116801(1460) ack 1 win 39420 (DF)
18.     16:55:54.095371 Recver.4785 > Sender.1264: . ack 81761 win 39420 (DF)
19.     16:55:54.095401 Sender.1264 > Recver.4785: P 116801:118261(1460) ack 1 win 39420 (DF)
20.     16:55:54.095629 Recver.4785 > Sender.1264: . ack 84681 win 39420 (DF)
21.     16:55:54.095736 Sender.1264 > Recver.4785: . 118261:119721(1460) ack 1 win 39420 (DF)
22.     16:55:54.095794 Sender.1264 > Recver.4785: P 119721:121181(1460) ack 1 win 39420 (DF)
23.     16:55:54.095931 Recver.4785 > Sender.1264: . ack 87601 win 39420 (DF)
24.     16:55:54.095959 Sender.1264 > Recver.4785: . 121181:122641(1460) ack 1 win 39420 (DF)
25.     16:55:54.096053 Sender.1264 > Recver.4785: P 122641:124101(1460) ack 1 win 39420 (DF)
26.     16:55:54.096135 Sender.1264 > Recver.4785: . 124101:125561(1460) ack 1 win 39420 (DF)
27.     16:55:54.096236 Recver.4785 > Sender.1264: . ack 90521 win 39420 (DF)
28.     16:55:54.096272 Sender.1264 > Recver.4785: P 125561:127021(1460) ack 1 win 39420 (DF)
29.     16:55:54.096390 Sender.1264 > Recver.4785: . 127021:128481(1460) ack 1 win 39420 (DF)
30.     16:55:54.096464 Sender.1264 > Recver.4785: P 128481:129941(1460) ack 1 win 39420 (DF)
31.     16:55:54.096551 Recver.4785 > Sender.1264: . ack 93441 win 39420 (DF)
32.     16:55:54.096835 Recver.4785 > Sender.1264: . ack 96361 win 39420 (DF)
33.     16:55:54.096915 Sender.1264 > Recver.4785: . 129941:131401(1460) ack 1 win 39420 (DF)
34.     16:55:54.096951 Sender.1264 > Recver.4785: . 131401:132861(1460) ack 1 win 39420 (DF)
35.     16:55:54.097025 Sender.1264 > Recver.4785: . 132861:134321(1460) ack 1 win 39420 (DF)
36.     16:55:54.097077 Sender.1264 > Recver.4785: P 134321:135781(1460) ack 1 win 39420 (DF)
37.     16:55:54.097157 Recver.4785 > Sender.1264: . ack 99281 win 39420 (DF)
38.     16:55:54.097266 Sender.1264 > Recver.4785: . 135781:137241(1460) ack 1 win 39420 (DF)
39.     16:55:54.097332 Sender.1264 > Recver.4785: P 137241:138701(1460) ack 1 win 39420 (DF)
40.     16:55:54.097567 Recver.4785 > Sender.1264: . ack 99281 win 39420 (DF)
41.     16:55:54.098788 Recver.4785 > Sender.1264: . ack 99281 win 39420 (DF)
42.     16:55:54.098935 Recver.4785 > Sender.1264: . ack 99281 win 39420 (DF)
43.     16:55:54.098985 Sender.1264 > Recver.4785: . 99281:100741(1460) ack 1 win 39420 (DF)
44.     16:55:54.099088 Recver.4785 > Sender.1264: . ack 99281 win 39420 (DF)
45.     16:55:54.099235 Recver.4785 > Sender.1264: . ack 99281 win 39420 (DF)
46.     16:55:54.099871 Recver.4785 > Sender.1264: . ack 99281 win 39420 (DF)
47.     16:55:54.100017 Recver.4785 > Sender.1264: . ack 99281 win 39420 (DF)
48.     16:55:54.100167 Recver.4785 > Sender.1264: . ack 99281 win 39420 (DF)
49.     16:55:54.100319 Recver.4785 > Sender.1264: . ack 99281 win 39420 (DF)
50.     16:55:54.100473 Recver.4785 > Sender.1264: . ack 99281 win 39420 (DF)
51.     16:55:54.100768 Recver.4785 > Sender.1264: . ack 100741 win 37960 (DF)
52.     16:55:54.231388 Sender.1263 > Recver.netperf: . ack 513 win 17520 (DF)
53.     16:55:55.466216 Sender.1264 > Recver.4785: . 100741:102201(1460) ack 1 win 39420 (DF)
```

Fig. 6 Tcpdump patch from pattern (b) just before Time-Out occurs.

99,281 (line 43). This will again make full both send and receive buffers. Sender again received seven duplicate ACKs for segment 99,281 (line 44 through 50). Sender will ignore all these ACKs because sender has just sent that segment. Sender receives ACK acknowledging data up to 100,740 (line 51). But the advertised window is 37,960, which is smaller by one MSS than the usual 39,420. It is indicating that there is still 1,460 bytes of data in TCP receive buffer, which is not read by the application. The window size 37,960 means 26 segments can be sent but sender has already sent 26 segments. Sender could not sent the new segment until it will receive a new ACK for one of the 26 segments sent by the sender. Sender receives another ACK (line No.52) but it is not the one expected by the sender. Here sender is waiting for the new ACK while receiver is waiting for new segment to come. Both are waiting and not sending anything. Retransmission possible only in two cases; 1) Three duplicate ACKs, 2) Time out. As there is no duplicate ACKs, so the only possibility is time out. Thus, time out happens (line No.53) which causes throughput deadlock. After this time out, TCP slow start algorithm starts once again.

According to fast retransmit and fast recovery algorithm [27], when third duplicate ACK is received, slow start threshold ssthresh is set to one-half of the minimum of the current congestion window (cwnd) and the receiver's advertised window. Retransmit the missing segment and set the cwnd to ssthresh plus 3 times the segment size. Each time another duplicate ACK arrives, increment cwnd by the segment size and transmit a packet if allowed by the new cwnd value. It means that fast retransmit and fast recovery algorithm only recovers one missing segment. It seems that there is more than one segment loss in one window, which is not recovered by TCP's fast retransmit and recovery algorithm.

It just reflects that the receiver has had to buffer a lot of data because it has sequence holes and can't deliver what it's holding. Sequence holes mean that there are some segment losses due to cell loss at ATM level. The time out occurs 7 times in about 10seconds. The gap between the acknowledged and unacknowledged bytes in all time out cases is 39,420, which is the window size. This gap never goes to this limit during 25 KB flight size.

TCP does not suffer throughput deadlock when we increase the ATM switch buffer size to 2kCells. When the connection is marked to generate a delayed ACK while buffer size is 1kCells, then this delay is not sufficient enough for ATM switch to buffer up data. Hence becomes overflow. But when Buffer is 2kCells then it has sufficient place to buffer the data for a longer delay than 1k Cells. This means that there is enough capacity (bandwidth, delay product) available along the path for 25 KB in flight but not for 38 KB. For 25 KB in flight, the connection becomes window-limited, and TCP over UBR streams achieve steady state. For 38 KB in flight, TCP suffers continuous buffer overflow and take a timeout. This contributes to TCP throughput deadlock.

Again if we turned on the EPD option implemented in the ATM switch, TCP throughput is recovered. Which proves our claim that there is a cell loss at ATM level, causing TCP throughput deadlock.

Thus the above discussion concludes that large window and small buffer sizes cause TCP throughput deadlock. As a future work, we would like to find out the number of missing segments during flight size of 38 KB and would like to increase more TCP connections.

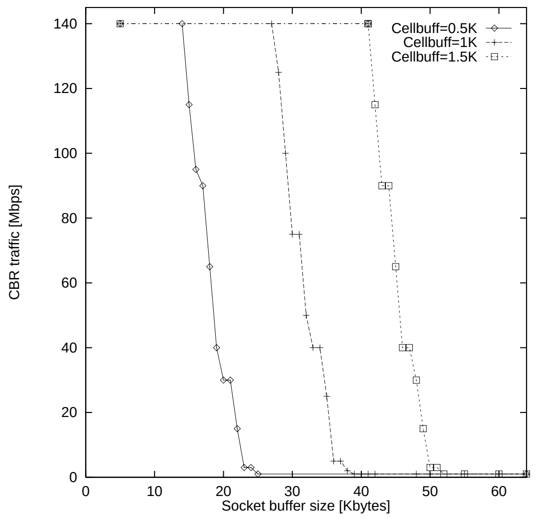


Fig. 7 CBR pressure when there is deadlock due to socket buffer size (cell buffer=0.5 K, 1 K, 1.5 K[cells]).

5. Deadlock Parameters

5.1 Combined Effectiveness of Socket Buffer and Cell Buffer Sizes on Deadlock

We carried on further experiments to find out the points where the UBR traffic throughput falls into deadlock. The result is shown in **Fig. 7**. For example for the first point we fixed CBR traffic 140 M[bps] and discovered the socket buffer size at which UBR throughput suffers deadlock. UBR throughput does not suffer deadlock with small socket sizes. From this point we decreased CBR pressure linearly and find out the last point up to which UBR suffers deadlock. This will become the first point. For the second point again the procedure will be same. We will go to the next socket size and decrease the CBR pressure until the final point of deadlock. It is quite possible that for the last CBR pressure, the new socket size will also suffers the final deadlock (the two consecutive points show these values). We noted the last value of CBR pressure and socket buffer size where UBR traffic suffers final deadlock. At the same point if we decrease the CBR pressure then there is no deadlock. The MTU size is 1,500.

Looking at Fig. 7, we have come to know that:

- If the socket buffer size is small enough, then there is no deadlock even through heavy CBR traffic.
- The switch buffer size is almost twice the

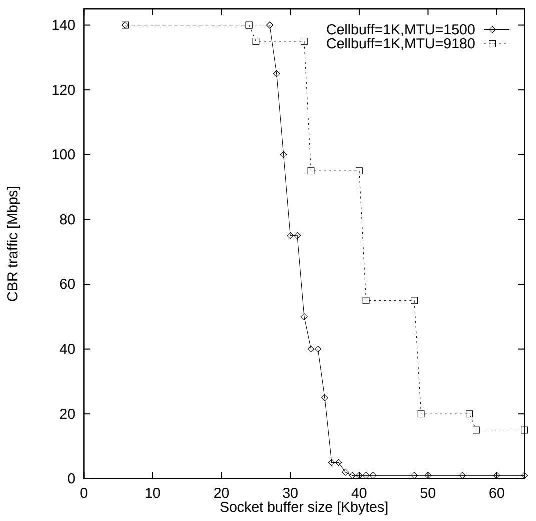


Fig. 8 CBR pressure when there is deadlock due to socket buffer size (MTU=1,500, 9,180 [bytes]).

socket buffer size when deadlock occur initially.

- Socket buffer size is directly proportional to the switch buffer size when deadlock starts initially.
- When socket size goes down from this initial point, CBR pressure decreases linearly and their slopes are same (parallel) regardless of cell buffer sizes.
- When this linearity comes to an end, even a CBR stream of 1M[bps] causes deadlock.

5.2 Deadlock as a Function of Socket Buffer and MTU Sizes

This experiment is exactly same as previous one but here we fixed cell buffer size as 1 Kcell and used two MTU sizes 1,500 and 9,180. We did the same procedure for each MTU and draw the graph as shown in **Fig. 8**, which shows:

- When deadlock begins to occur, there is no relationship between socket buffer size and MTU.
- When we increase the socket buffer size from this point, the linearity degree of CBR pressure is inversely proportional to MTU.

The steps, when MTU is 9,180, are due to the fact that window size is set by the TCP, rounding the MSS value.

6. Concluding Remarks

In this paper, we have shown that TCP Reno over UBR streams congested with CBR streams in ATM networks suffers throughput degradation and is less than 2% of the available bandwidth. The throughput suffers several timeouts due to inefficiency of Fast Retransmit and Fast Recovery Algorithms, which were unable to recover more than one segment loss in a window. A number of researchers have observed that TCP's loss recovery strategies do not work well when congestion window at a TCP sender is small [1]. This can happen, for instance, because there is only a limited amount of data to send, or because of the limit imposed by the receiver-advertised window, or because of the constraints imposed by end-to-end congestion control over a connection with a small bandwidth-delay product [4),5),18),23),26)]. What we have found is another occasion when the mechanism does not work well.

There has been proposed new strategies, like NewReno [8], TCP SACK [19], TCP Vegas [7], and many others [1),9),20)]. We have already reported some results on NewReno and TCP SACK [11], but comprehensive survey of other existing TCP implementation, including Tahoe [12], over congested ATM networks would be left as a future work.

As we have shown in detail that the large MTU that causes deadlock. It is well known that the large MTU of 9180 bytes recommended in Ref. 3)[☆] causes deadlocks [21], but here we have shown that another kind of deadlocks will occur under CBR pressure and the situation is worse. We dare say that there is little positive reason to use such large MTU now. In other words, we can get more performance on ATM-backbone solutions (where hosts have Ethernet NIC and ATM is deployed only on backbone routers) or on ATM LANE solutions, than on native end-to-end IPoA solution. This is quite opposite of what were said before.

We may avoid the deadlock by preparing enough size of cell buffer, or deploying PPD (Partial Packet Discard) and EPD (Early Packet Discard) schemes at ATM switch level as was discussed in Ref. 24). Actually brand-new ATM switches now sold for ATM LANs have enough cell buffer, possibly 16 Kcells or more (mainly for ABR support, we guess), and they have also PPD and/or EPD features. However, we must note that ATM switches already installed in campus ATM LANs or carriers' national backbones do not necessarily have so much buffer nor such features. So there still remain strong needs to improve the performance of TCP over ATM by introducing newer algo-

[☆] Now in Ref. 17).

rithms to the implementation.

References

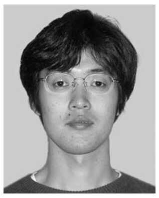
1) Allman, M., Balakrishnan, H. and Floyd, S.: Enhancing TCP's Loss Recovery Using Limited Transmit, RFC 3042 (Jan. 2001).
2) Armitage, G. and Adams, K.: Packet reassembly during cell loss, *IEEE Network Magazine*, Vol.7, No.5, pp.26–34 (Sept. 1993).
3) Atkinson, R.: Default IP MTU for Use over ATM AAL5, RFC1626, (May 1994) (Obsoleted by RFC 2225).
4) Balakrishnan, H., Padmanabhan, V., Seshan, S., Stem, M. and Katz, R.: TCP Behavior of a busy Web Server: Analysis and Improvements, Technical Report UCB/CSD-97-966 (Aug. 1997). Available from URL: http://nms.lcs.mit.edu/hari/papers/csd-97-966.ps. (Also in *Proc. IEEE INFOCOM Conf.*, San Francisco, CA, March 1998.)
5) Balakrishnan, H.: Challenges to Reliable Data Transport over Heterogeneous Wireless Networks, Ph.D. Thesis, University of California at Berkeley (Aug. 1998).
6) Braden, B. (Ed.): Requirements for Internet hosts-communication layers, RFC 1122 (Oct. 1989).
7) Brakmo, L.S. and Peterson, L.L.: TCP Vegas: End-to-End Congestion Avoidance on a Global Internet, *IEEE Journal on Selected Areas in Communication*, Vol.13, No.8 (Oct. 1995).
8) Floyd, S. and Henderson, T.: The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 2582 (Apr. 1999).
9) Hoe, J.: Improving the Start-up Behavior of a Congestion Control Scheme for TCP, *SIGCOMM Symposium on Communications Architectures and Protocols* (Aug. 1996).
10) Ishibashi, H., Okabe, Y. and Kanazawa, M.: Implementation of a very large ATM LAN for Kyoto University, *Proc.13th International Conference on Systems Science*, III, pp.60–67 (Nov. 1998).
11) Ahmed, I., Okabe, Y. and Kanazawa, M.: Throughput Evaluation of Several TCP Implementations over Congested ATM Networks, *Proc. 9th IFIP Working Conference on Performance Modelling and Evaluation of ATM & IP Networks*, Budapest, Hungary (June 2001).
12) Jacobson, V.: Congestion Avoidance and Control, *SIGCOMM Symposium on Communications Architectures and Protocols*, pp.314–329 (1988).
13) Jacobson, V.: Modified TCP Congestion Avoidance Algorithm, Technical Report (Apr. 1990). Available at following URL. ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.
14) Jones, R.: *Netperf, A benchmark for measuring network performance*, Hewlett-Packard Co. (1993).
15) Kent, C. and Mogul, J.: Fragmentation Considered Harmful, ACM SIGCOMM '87, pp.390–401 (Aug. 1987).
16) Laubach, M.: Classical IP and ARP over ATM, RFC 1577, (Jan. 1994). (Obsoleted by RFC2225).
17) Laubach, M. and Halpern, J.: Classical IP and ARP over ATM, RFC2225 (Apr. 1998). (Obsoletes RFC1626, RFC1577) (Status: PROPOSED STANDARD).
18) Lin, D. and Kung, H.T.: TCP Fast Recovery Strategies: Analysis and Improvements, Proceedings of InfoCom (March 1998).
19) Mathis, M., Mahdavi, J., Floyd, S. and Romanow, A.: TCP Selective Acknowledgement Options, RFC 2018 (Oct. 1996).
20) Mathis, M., Semke, J. and Mahdavi, J.: The Rate-Halving Algorithm for TCP Congestion Control (June 1999). http://www.psc.edu/networking/ftp/papers/draft-ratehalving.txt
21) Moldeklev, K. and Gunningberg, P.: How a large ATM MTU causes deadlocks in TCP data transfers, *IEEE/ACM Transactions on Networking*, Vol.3, No.4 (Aug. 1995).
22) Moldeklev, K. and Gunningberg, P.: Deadlock situations in TCP over ATM, *Proc. IFIP IV Workshop on Protocols for High Speed Networks*, Vancouver, B.C., Canada (Aug. 1994).
23) Morris, R.: TCP Behavior with Many Flows, *Proc. Fifth IEEE International Conference on Network Protocols* (Oct. 1997).
24) Romanow, A. and Floyd, S.: Dynamics of TCP traffic over ATM networks, *IEEE Journal on Selected Areas in Communications*, Vol.13, No.4 (May 1995).
25) Postel, J.: The TCP Maximum Segment Size and Related Topics, RFC 879, SRI Network Information Center (Nov. 1983).
26) Rizzo, L.: *Issues in the Implementation of Selective Acknowledgements for TCP* (Jan.1996). URL: http://ww.iet.unipi.it/luigi/selack.ps
27) Stevens, W.R.: *TCP Illustrated: Volume 1: The Protocols*, Addison-Wesley, Reading, MA (1994).
28) Tanenbaum, A.S.: *Computer Networks*: 3rd ed., Prentice-Hall International Editions (1994).
29) Tcpdump 3.4, Lawrence Berkeley National Laboratory, Network Research Group.

(Received September 25, 2001)
(Accepted March 14, 2002)

Editor's Recommendation

The authors observe how lower priority traffics are affected if higher priority traffics exist, and make it clear that, under some conditions, deadlocks can be occurred. Major reasons of deadlocks are analysed and conditions that may cause deadlocks are experimentally derived. The situations reported here are getting more significant as the broad band transmission applications such as motion picture transmissins are frequently performed.

(Chairman of SIGDSM Katsuya Hakozaki)



Motoki Nakanishi received the B.E. degree from Kobe University, Japan, in 1998. He received the Master of Informatics degree from Kyoto University, Japan, in 2000. Since 2000, he is working in Toshiba corporation.

Ishtiaq Ahmed received M. Sc and Master of Philosophy (M. Phil) degrees in Electronics from Quaid-i-Azam University, Pakistan in 1991 and 1994 respectively. He has completed Ph.D. degree in Informatics, from Graduate School of Informatics, Kyoto University, Japan. Currently working as a Research Associate in Kyoto University. His research interests involve high-speed network protocols, Ethernet Gigabit networks and DiffServ architecture.

Hayato Ishibashi received the M. Eng. degree in information science from Kyoto University in 1987. He is presently with Osaka City University as a lecturer of the Media Center. His current research interests include computer network security and next generation networking. He is a member of the IEEE, the ACM, the IEICE, and the JSAI.

Yasuo Okabe received the ME from Department of Information Science, Kyoto University in 1988. From 1988 he was an Instructor of Faculty of Engineering, and from 1994 he was an Associate Professor of Data Processing Center, Kyoto University. He is now an Associate Professor of Graduate School of Informatics, Kyoto University. Ph.D. in Engineering. His current research interests include multimedia networking and distributed systems. He is a member of IEICE, ISCIE, JSSST, IEEE, ACM, and EATCS.

Masanori Kanazawa was born in 1946. He received the ME from Department of Applied Mathematics and Physics, Kyoto University in 1971. From 1972 he was an Assistant Professor and Associate Professor of Data Processing Center Kyoto University. From 1995 he is a Professor of Data Processing Center Kyoto University. Ph.D. in Engineering. His current research interests include computer networks and distributed systems. He is a member of IEICE, ACM, and JSIAM.