輻輳した ATM 網において TCP のデッドロックを 回避する動的粒度制御アルゴリズム*

Ishtiaq Ahmed[†] · 岡部 寿男[†] · 金澤 正憲[‡]

A Dynamic Granularity Control Algorithm for Avoiding TCP Deadlocks over Congested ATM Networks*

Ishtiaq Ahmed[†], Yasuo OKABE[†] and Masanori KANAZAWA[‡]

Conventional TCP suffers throughput deadlocks over congested ATM (Asynchronous Transfer Mode) networks due to retransmission timeouts caused by cell loss at ATM level under heavy congestion. A modified congestion control algorithm for TCP is proposed to improve the effective throughput of IP over ATM UBR even in a situation such that an external CBR pressure exist at the same port of the ATM switch. When TCP faces congestion then TCP will reset its retransmission timer. In the proposed algorithm, the MSS (maximum segment size) is reduced from 9140 bytes, the recommended value in RFC 1612, at the same time. When TCP is in fast retransmit machine phase, the MSS is reduced to 512 bytes. A considerable improvement in the effective throughput of TCP has been achieved by these two modifications under relatively small switch buffer size and high CBR stream pressure. The idea, called “Dynamic Granularity Control (DGC)”, have been implemented and evaluated on the Linux kernel.

1. Introduction

TCP [11] is the most important protocol in today's Internet and high-speed networks. ATM is one of the key technologies for broadband networks since the last decade. ATM networks can provide QoS (Quality of Service) to multimedia applications as CBR (Constant Bit Rate), simultaneously providing best-effort service, UBR (Unspecified Bit Rate), to other traffics like TCP/IP, via IPoA (IP over ATM) [13,15] or ATM LANE (LAN Emulation) [16,17].

However, new problems arise when we deploy TCP on IPoA networks. The large Maximum Transfer Unit (MTU) of ATM [14] causes considerable degradation of the performance in TCP data transfers [12]. ATM cell contains payload of 48 bytes and 9,182 bytes of an IP packet is fragmented into many cells via AAL5. Only one cell loss at ATM level heavily affects the performance at the protocol level [19]. This prob-

lem gives rise to Partial Packet Discard (PPD) and Early Packet Discard (EPD) strategies to improve the throughput of TCP for best effort traffic [19]. PPD is referred as “Selective Cell Discarding” in Ref. [20].

TCP Reno for the first time added fast retransmit and recovery algorithms to recover the lost segments [9,10]. But TCP Reno could not recover multiple segment losses in a single window of data [1]. TCP suffers timeouts too frequently due to inefficiency of fast retransmit and fast recovery algorithm [6] over a congested ATM link and results in duplicate transmission of data segments during slow start. The fast retransmit and recovery algorithms are modified in a way that the sender does not quit fast retransmit phase till it receives an acknowledgement from the receiver acknowledging the outstanding data in the network [1]. This modification is adopted in NewReno algorithm [2]. Another modified fast retransmit and fast recovery algorithm called as Selective Acknowledgment Options (SACK) [3] is presented to recover lost segments and to avoid duplicate retransmissions at the same time. This algorithm is going to be adopted as standard in near future. There is a comparative study about these algorithms based on simulation results [4].

The algorithms TCP NewReno [2] and SACK [3], significantly recover the segment losses but still have

* 原稿受付 2001 年 9 月 7 日

[†] 京都大学 大学院 情報学研究科 Graduate School of Informatics, Kyoto University; Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, JAPAN

[‡] 京都大学 大型計算機センター Data Processing Center, Kyoto University; Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, JAPAN

Key Words: TCP, congestion control, ATM, IPoA.

the possibility to suffer retransmission timeouts causing throughput deadlocks. We have conducted an experimental study on a real congested ATM network and found that once the congestion occurs at the ATM switch, the performance of TCP becomes less than one fifth of the available bandwidth [5]. Through our extensive experimentation, we have examined that MTU size of 9180 bytes causes throughput of TCP worst and have found that small MTU size improves the throughput of TCP under small ATM switch cell buffer in size. It contributes to the improvement of not only the packet loss ratio but the behavior of congestion control mechanism for small MTU packets. This mechanism is quite different from that for large MTU packets. The cell loss at ATM level is due to mismatch of bandwidth of the input link and the output link on ATM switch. Although flow control mechanism suppresses the speed of input flow in long term by setting the window size smaller, but they are sent bursty with the full speed of the data link once data is set up on the window as packets. If the ATM switch receives (cellized) packets sent bursty, the cell buffer on the intermediate ATM switch will soon overflow, and since then cells are transferred partially. Under the standard AAL5 encoding with large MTU recommended in RFC1626 [14]/RFC2225 [15], loss of only one cell (53 bytes) causes discard of whole 9180 bytes of the packet.

We proposed a new TCP congestion avoidance scheme, which dynamically changes the MSS in anticipation of congestion at the ATM switch. When TCP faces congestion, TCP will reset its retransmission timer, and decrease the MSS to 1460 bytes if MSS is greater than 1460 bytes. If congestion still persists, then it further decreases the MSS to 512 bytes. If the sender is in the fast retransmit and fast recovery phase, we set the MSS to 512 bytes. If congestion is over, the algorithm will change the MSS size to its initial value 9140 bytes. We call this idea “Dynamic Granularity Control” (DGC) for TCP. We have presented the results showing that throughput has improved significantly without experiencing a retransmission timeouts. The rest of the paper is organized as: Section 2. describes experimental design and connection setup. Section 3. explains proposed DGC TCP algorithm. Section 4. and 5. present results and conclusions respectively.

2. Experimental Design

In this section, we will show our real experimental model, equipment used, connection configuration and method of experimentation.

2.1 Real Experimental Model

We are assuming a congested ATM link where:

- (1) Two flows exist. One is CBR (Constant Bit Rate) flow, which has absolute precedence, and the TCP flow (which is on UBR—Unspecified Bit Rate) is getting pressure of the CBR streams

at the same port.

- (2) The cell buffer size in the ATM switch for UBR is relatively of small size.

The real experimental setup is shown in Fig. 1 and the equipment is being listed in Table 1. RedHat Linux-7.0 with 2.4.0-test10 kernel with ATM-0.78 distribution is used, which has NewReno and SACK implemented together. The ATM-0.78 distribution supports LANE, MPOA [18] and classical IP over ATM (CLIP). The CLIP setup has used for experimentation. We used The ATM switch, Fujitsu EA1550 that has output-buffer architecture. We can change the size of allocated cell buffer to each port. Packet discarding schemes are not enabled within the switch. We have monitored the CBR and UBR traffic on SVCs and PVCs with the Hewlett Packard’s ATM traffic analyzer HP E5200A. It can also generate the CBR streams of different pressures.

Fig. 1 Experimental configuration

Table 1 System equipment specifications

CPU	Pentium III 550MHz
M/B	BE6
Memory	SD-RAM 64Mbytes
ATM NIC	ENI-155p
Operating System	Red Hat Linux-7.0J Kernel-2.4.0-test10
ATM Driver	ATM-0.78 CLIP

2.2 Experimental Method

Netperf [7] is used as a tool that can be used to measure various aspects of networking performance. We injected CBR streams from ATM HP traffic analyzer’s traffic generator. TCP data transfer is over UBR streams. The netperf test is running for 10 seconds and the output of netperf shows the TCP’s effective throughput over UBR streams. UBR streams and CBR streams are running on the same port within the switch.

3. DGC Algorithm

TCP SACK implementation in Linux does not allow the flight size to be greater than 3 or 4 segments, thus it seems trivial to recover a segment loss. It has been found that TCP SACK is robust in face of segment loss and can recover multiple segment loss. But still TCP SACK faces retransmission timeouts [5]. Thus, SACK will be bad in a connection sensitive situation due to retransmission timeouts.

Another important factor is found that SACK consumed a lot of time to recover a segment loss. These factors trigger us to propose a new algorithm to increase fair share of UBR bandwidth avoiding retransmission timeouts. Since the TCP's standard congestion control algorithm consists of slow start, congestion avoidance, fast retransmit and fast recovery [9,10,12]. Therefore, the following modification applies to these algorithms without changing the fundamental algorithms itself except the behaviour of the sender facing congestion at the ATM switch. Every TCP implementation MUST follow these algorithms as laid out by Ref. [12]. The slow start algorithm is active at the start of the TCP connection. The sender can pump the data into the network according to the initial window IW, whose value must be less than $2 \times \text{MSS}$ and it MUST not be greater than two segments. TCP SACK implemented in Linux follows a non-standard [21], experimental TCP extension, which allows a TCP MAY use a larger initial window (IW), as defined in eq. (1):

$$IW = \min(4 \times \text{MSS}, \max(2 \times \text{MSS}, 4380(\text{bytes}))) \quad (1)$$

With this extension, a sender MAY use a 3 or 4 segment initial window, provided the combined size of the segments does not exceed 4380 bytes. This change is NOT allowed as part of the standard algorithm defined by Ref. [12] but for experimental purpose it has been allowed. We agree with this extension as part of the TCP algorithm implemented in Linux and found to be effective. Because one of the drawbacks found with TCP Reno is that it injects too many packets during slow start, while sender's rate defined in Ref. [12] is too conservative. In our opinion, it should be slightly aggressive than that is proposed by Ref. [12] but should be more conservative than Reno.

The slow start algorithm is used when congestion window $\text{cwnd} < \text{ssthresh}$, while the congestion avoidance algorithm is used when $\text{cwnd} > \text{ssthresh}$. When cwnd and ssthresh are equal, the sender may use either slow start or congestion avoidance. TCP SACK follows congestion avoidance when $\text{cwnd} > \text{ssthresh}$. During congestion avoidance, cwnd is incremented by 1 full-sized segment per round-trip time (RTT). Congestion avoidance continues until congestion is detected. One formula commonly used to update cwnd during congestion avoidance is given in eq. (2):

$$\text{cwnd} += \text{MSS} \times \text{MSS} / \text{cwnd} \quad (2)$$

This adjustment is executed on every incoming non-duplicate ACK. Eq. (2) provides an acceptable approximation to the underlying principle of increasing cwnd by 1 full-sized segment per RTT. For a connection in which the receiver acknowledges every data segment, (2) proves slightly more aggressive than 1 segment per RTT. For a receiver acknowledging every-other segment (2) is less aggressive. TCP SACK receiver in Linux acknowledges every-other seg-

ment of data. The congestion avoidance follows $\text{snd.cwnd} += 1 / \text{snd.cwnd}$ instead of (2) in Linux implementation.

The first modification is proposed at this stage. The sender is going to face congestion soon at this point due to high-speed nature of ATM networks. This zone is defined as dangerous zone in Linux. The idea here is to reduce the size of MSS to 1460 bytes if current MSS is greater than 1460 bytes. MSS is further reduced to 512 bytes if congestion persists or current MSS is equal to 1460 bytes. At the same time, the retransmission timer is being reset. But whenever the sender will come out from this dangerous zone, it will go into the safe zone (i.e., slow start). MSS will be set to its original size which was discovered during path MTU discovery process Ref. [22]. For implementation purposes, the variable named CONGES is used as shown in the flowchart Fig. 2. It is more conservative approach and will take slightly more time to reach up to the buffer overflow situation. It has been found that congestion limit is achieved soon if MSS value is large enough [5].

When a sender detects segment loss using the retransmission timer or the duplicate ACKs, the value of ssthresh MUST be set to no more than the value given in eq. (3):

$$\text{ssthresh} = \max(\text{FlightSize}/2, 2 \times \text{MSS}) \quad (3)$$

FlightSize is the amount of outstanding data in the network.

The second modification is applied to this stage when TCP is in fast retransmit and fast recovery phase. This phase is called Fast Retransmit Machine (FRM) in the TCP SACK implementation of Linux. It has been further observed that during FRM, the sender could not send any segment because congestion window does is greater not allow the sender to send data [5]. This phase has a number of stages depending upon the activities of TCP during congestion situation. These stages include reordering of segments, fast retransmit, fast recovery and segment loss. The variable named STATE is being used for implementation. During FRM the MSS is reduced to 512 bytes in any case (if current MSS is greater or equal or less than 1460 bytes). The logic is clear from the flow chart diagram shown in Fig. 2. This algorithm is named as "Dynamic Granularity Control"—DGC algorithm for avoiding TCP deadlock over congested ATM networks.

These modifications are required at the sender side only and are equally good for both TCP NewReno as well as TCP SACK. We found that effective throughput of TCP has improved to a significant level and retransmission timeout does not expire.

4. Results and Discussions

The experiment is conducted to show the effect of congestion introduced by reducing the switch buffer

Fig. 2 Flowchart for DGC algorithm

size and the improved throughput of TCP. Throughput of DGC TCP is significantly improved when switch buffer size is small (specially when buffer size is less than 53 Kbytes) as we can see the Fig. 3.

The throughput improvement of DGC MTU of 9180 bytes from SACK MTU of 9180 bytes is clear when switch buffer size ranges from 511 cells to 1K cells (26 Kbytes to 53 Kbytes). In this region the effective throughput is improved and becomes 98% of the available bandwidth.

Fig. 3 Throughput versus UBR switch buffer size. Window size is 64 Kbytes.

Although SACK MTU of 512 bytes does not fall into a fatal deadlock stage at any condition, we can not utilize a small value like MTU=512 bytes as the fixed recommended value of IP over ATM, instead of the currently recommended value 9180 bytes, since the performance degradation all over the range cannot be ignored. The result shows:

- Throughput bandwidth gap at CBR 100Mbps between DGC MTU=9180 bytes and SACK MTU=512 bytes is 2.76Mbps approximately, which is 6.9% of the available bandwidth, at switch buffer size of 53 Kbytes.
- Throughput bandwidth gap at CBR=60Mbps between DGC MTU=9180 bytes and SACK MTU=512 bytes is 9.26Mbps approximately, which is 11.57% of the available bandwidth, at switch buffer size of 53 Kbytes.

In the next experiment, we have examined how much CBR traffic effects on the performance of SACK TCP and how tolerant the DGC TCP is. The switch buffer size is kept constant at 53 Kbytes and window size is 64 Kbytes. Fig. 4 shows the graph for DGC TCP and SACK TCP comparison with reference to effective bandwidth utilization. From Fig. 4 it is clear that:

- SACK MTU of 9180 bytes could not utilize the available bandwidth when CBR pressure is being increased beyond 90Mbps. The decrease of throughput is extremely large, less than 1/5 of the available bandwidth.
- SACK MTU of 512 bytes could not utilize the available bandwidth on all range of the CBR pressure, and the performance has reached ceiling at 86Mbps when CBR pressure is less than 40Mbps or even no pressure.
- In contrast, DGC TCP performs very well and almost all of the available bandwidth is utilized in any CBR pressure.

The next experiment shows the number of segments acknowledged (Number of bytes) by the receiver at a given time. We used Tcpdump [8] software

Fig. 4 Throughput versus CBR pressure. Window size = 64 Kbytes and buffer size is 53 Kbytes.

Fig. 5 Number of Segments acknowledged by the receiver

to dump the segment statistics. The experimental parameters are MTU=9180 bytes, CBR=100Mbps (the external pressure) and the windows size is 64 Kbytes. The comparison between TCP SACK and DGC is shown in Fig. 5, which shows that:

- There is no retransmission timer is being experienced by the DGC TCP algorithms.
- The effective throughput is 41.71 Mbps whereas that of SACK is 7.06 Mbps, which is 80% more than that of SACK.
- By closely looking into the Tcpdump file, it is stated that there is only instance when TCP falls in to FRM phase and successfully recovers segment loss, which can be seen in the Fig. 5 at the start of the graph.
- It is easy to handle the smaller segments as compared to big data segments (in number of bytes). Thus, when the TCP goes into the dangerous zone, the sender will follow a very conservative approach to avoid the equilibrium state and never enters a FRM state. The sender rarely meet a segment loss situation following this approach.
- TCP SACK receiver acknowledges every other

segment. It is interesting to note that during path MTU discovery, MTU is going to be discovered 9180 bytes and receiver always sends an ACK acknowledging the same number of bytes each time.

5. Conclusion

We have analyzed the TCP SACK algorithm, the known best algorithm ever to recover a segment loss when TCP deployed over ATM. It performs fairly much better than conventional implementation of TCP like TCP Reno. But the effective throughput is still less than 20% of the available bandwidth in situation such that CBR pressure exist at the same port of the ATM switch. Despite the fact that the effective throughput of TCP SACK is improved, it is still unable to recover multiple segment losses with in the limit of retransmission timeout. The retransmission timeouts and segment loss recovery processes of TCP SACK consume more than half of the total time. TCP SACK will be bad in connection sensitive applications because of its retransmission timeout expires.

We have proposed the Dynamic Granularity Control Algorithm for TCP over congested ATM networks. We have modified the Jacobson's congestion avoidance algorithm such that TCP DGC dynamically reduces the MSS size in anticipation to the congestion at ATM switch, and resets the retransmission timer at the same time. During fast retransmit and fast recovery phase, the MSS value is reduced to 512 bytes in any case. We have presented practical results, which have shown that TCP has improved the effective throughput to more than 80% to that of SACK TCP.

References

- [1] J. Hoe: Improving the start-up behavior of a congestion control scheme for TCP; *SIGCOMM Symposium on Comm. Architectures and Protocols* (1996)
- [2] S. Floyd and T. Henderson: The NewReno modification to TCP's fast recovery algorithm; RFC 2582 (1999)
- [3] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow: TCP selective acknowledgement Options; RFC 2018 (1996)
- [4] K. Fall and S. Floyd: Simulation-based comparisons of Tahoe, Reno, and SACK TCP; *ACM Computer Communication Review* (1996)
- [5] A. Ishtiaq, Y. Okabe and M. Kanazawa: Throughput evaluation of several TCP implementations over congested ATM networks; *Proceedings of 9th IFIP Conference on Performance Modelling and Evaluation of ATM & IP Networks 2001*, pp. 284-295 (2001)
- [6] A. Ishtiaq, M. Nakanishi, H. Ishibashi, Y. Okabe and M. Kanazawa: Throughput deadlock mechanism of TCP over congested ATM networks; *Proceedings of First International Workshop on Distributed Computing, Comm. and Applications (Islamabad)*, pp. 56-63 (2000)
- [7] R. Jones: *Netperf, A Benchmark for Measuring Net-*

- work Performance*, Hewlett-Packard Co. (1993)
- [8] Tcpdump 3. 4, Lawrence Berkeley National Laboratory, Network Research Group.
 - [9] V. Jacobson: Congestion avoidance and control; *SIGCOMM Symposium on Comm. Architectures and Protocols*, pp. 314–329 (1988)
 - [10] V. Jacobson: Modified TCP congestion avoidance algorithm; Technical report, April 30th, 1990. Available at following URL: <ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt>.
 - [11] M. Allman, V. Paxson and W. Stevens: TCP congestion control; RFC 2581 (1999)
 - [12] K. Moldeklev and P. Gunningberg: How a large ATM MTU causes deadlocks in TCP data transfers; *IEEE/ACM Transactions on Networking*, Vol. 3, No. 4 (1995)
 - [13] M. Laubach: Classical IP and ARP over ATM; RFC 1577 (1994) (Obsoleted by RFC2225)
 - [14] R. Atkinson: Default IP MTU for use over ATM AAL5; RFC1626 (1994) (Obsoleted by RFC 2225)
 - [15] M. Laubach and J. Halpern: Classical IP and ARP over ATM; RFC2225 (1998) (Status: Proposed Standard)
 - [16] ATM Forum. LAN Emulation Over ATM – Version 1. 0 (1996)
 - [17] ATM Forum. LAN Emulation Over ATM – Version 2 — LUNI Specification (1997)
 - [18] ATM Forum. Multi-Protocol Over ATM – Version 1. 0 (1997)
 - [19] A. Romanow and S. Floyd: Dynamics of TCP traffic over ATM networks; *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 4 (1995)
 - [20] G. Armitage and K. Adams: Packet reassembly during cell loss; *IEEE Network Magazine*, Vol. 7 No. 5, pp. 26–34 (1993)
 - [21] M. Allman, S. Floyd and C. Partridge: Increasing TCP's initial window size; RFC 2414 (1998)
 - [22] J. Mogul and S. Deering: Path MTU discovery; RFC 1191 (1990)
-