A Throughput Deadlock-Free TCP for High-Speed Internet [1]

Rocky K.C. Chang[2] Ho Y. Chan
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
Email: csrchang@comp.polyu.edu.hk

Abstract

Throughput deadlocks were observed when TCP was operated on high-speed networks. This deadlock problem is caused by the interaction of the sender-side and receiver-side silly window syndrome avoidance algorithms, because a TCP connection's Maximum Segment Size is no longer small on high-speed Internet when compared with the send and receive socket buffer sizes. In this paper we propose a new Congestion-Sensitive, Adaptive Acknowledgment Algorithm (CS-AAA) to solve the deadlock problem. Unlike our previously proposed AAA, the CS-AAA is able to respond to and to recover from congestion much faster than AAA. The CS-AAA solves this problem by detecting congestion, and performing a slow-start-like mechanism. Extensive simulation results support that CS-AAA's throughput performance significantly exceeds that of AAA, especially when the send buffer size is relatively large.

1. Introduction

Recently, much attention has been drawn to improving performance of Transport Control Protocol (TCP) because it provides end-to-end reliability, flow control, and congestion control services to a number of very popular application and session protocols, such as HTTP, FTP, TELNET, SSL, e.g., [1,2]. In this paper, we consider a TCP throughput deadlock problem which, unlike other TCP performance problems, is caused by implementation issues rather than protocol design issues. This problem could occur in low-speed networks as well as high-speed networks. However, the impact on high-speed networks is definitely more observable and therefore more detrimental.

This problem was first reported by Moldeklev and Gunningberg, and by Comer and Lin independently [3,4]. In their investigation, they discovered that the deadlock problem was caused by a circular-wait condition exhibited between the sender-side and receiver-side Silly Window Syndrome Avoidance Algorithms (SWSAAs) that are implemented by Nagle's algorithm and a delayed acknowledgment algorithm, respectively. When the send-receive socket buffer sizes fall in a certain region, the sender will not send small segments due to Nagle's algorithm, and the receiver will not acknowledge because of the delayed acknowledgment algorithm. This deadlock can be resolved only by a 200-ms delayed acknowledgment timer.

A main factor triggering the circular-wait condition is that a TCP connection's Maximum Segment Size (MSS), which is used by the SWSAAs, is no longer small on high-speed Internet when compared with the send and receive socket buffer sizes. Nagle's algorithm defines a small segment to be one whose length is less than the connection's MSS, and it usually limits the number of outstanding small segments (nonMSS segments) to one [5]. The delayed acknowledgment strategy, on the other hand, prevents a receiver from acknowledging small segments by delaying acknowledgments until they can be piggybacked onto either a data segment or a window update packet [6]. For example, in the SunOS implementation, a separate window update with a piggybacked acknowledgment will be sent if the window can slide more than either (1) 35% of the receive buffer size or (2) two MSSs of the size. As a result, for a TCP connection with a large MSS (on a high-speed end-to-end connection), a TCP sender may not be able to compose a MSS segment if its send buffer size is not large enough. Similarly, a TCP receiver may not be able to acknowledge since the amount of data received is not large enough when compared with MSS.

Several straightforward solutions to solving the deadlock problem are either infeasible or re-introduce the SWS problem [3]. In [7], we introduced a new Adaptive Acknowledgment Algorithm (AAA) and demonstrated that this algorithm ensures a deadlock-free TCP connection provided that the connection does not experience network congestion. In this paper, we probe further into this solution. First, we show that the algorithm's performance is not significantly affected by the sampling frequency. Second, we show that the algorithm is not sufficient for preventing throughput deadlock when TCP connections experience network congestion. Third, we propose and evaluate an enhanced AAA by take into consideration network congestion. They are described in Section 2, 3, and 4, respectively. Finally, we conclude this paper with future work in section 5.

[1] This work is partially supported by the Hong Kong Polytechnic University Research Grant S909.
[2] Corresponding author

0-7695-0777-8/00 $10.00 © 2000 IEEE

2. An adaptive acknowledgment algorithm

In [7], we proposed a new Adaptive Acknowledgment Algorithm (AAA) as a new delayed acknowledgment strategy to remove throughput deadlocks and, at the same time, to maintain the SWS avoidance mechanisms. In this algorithm, the receiver is responsible for "resolving" deadlock situations. The key operation undertaken by the receiver is to estimate the sender's Maximum UnAcknowledged Data Size (MUADS) in terms of bytes, which is the maximum amount of data continuously sent by the sender when the receiver is not acknowledging. When deadlock situation does not occur and there is no network congestion, the MUADS is upper bounded by $\min\{B_s, B_r\}$, where B_s and B_r are the send buffer size and receive buffer size, respectively. The exact value of MUADS depends on buffer sizes, data copy rule, and other implementation details. An AAA receiver sends an acknowledgment when the segments received reach or exceed 35% of the MUADS estimate. It has been shown that the AAA removes all deadlocks by having the receiver promptly send back acknowledgments [7].

An AAA receiver performs packet samplings to estimate the sender's MUADS after a TCP connection is established by maintaining one timer, one counter, and two thresholds:

1. Sampling period timer (SP_timer): A timer to control the length of a sampling period
2. Sampling period threshold (SP_t): A value (in time units) used in conjunction with the SP_timer to control the length of a sampling period
3. Inter-sampling period counter (ISP_counter): A counter to control the inter-sampling period
4. Inter-sampling period threshold (ISP_t): A value (in number of packets) used in conjunction with the ISP_counter to control the inter-sampling period

In Fig. 1, we show the state transition diagram when a receiver enters a packet sampling period. Please refer to [7] for further details about inter-sampling period control and sampling initialization, sampling period control, and threshold setting.

One possible way of further improving AAA's throughput performance is to reduce the number of samplings required for computing MUADS. There are two ways to achieving that: increase the inter-sampling period and decrease the sampling period. Modifications to the corresponding algorithms are shown in Fig. 2. In Table 1, we present simulation results to compare the original AAA with the modified AAA. Although the number of samplings decreases significantly (not shown here), the throughput improvement is not as significant.

3. Effect of network congestion on AAA

The AAA was designed without network congestion in mind. In this section, we show and explain why the AAA fails to remove throughput deadlocks when TCP connections experience network congestion.

It is well known that a TCP sender detects possible network congestion either by receiving three duplicate acknowledgments (fast retransmission) or by a timeout. In either case, the sender retransmits the segments that have not been acknowledged. At the same time, the sender also adjusts its congestion window size. In the case of fast retransmission, the sender reduces its congestion window to only half of the current value. In the case of timeout, the window size is reset to one MSS. Both cases effectively reduce the sender's sending rate. As a result, the receiver's MUADS estimate that is based on pre-congestion statistics will over-estimate the actual value. Therefore, the receiver may not be able to receive enough data to trigger acknowledgments, and a throughput deadlock occurs again.

To further investigate the throughput deadlock problem under network congestion, we have performed several simulation experiments. In all experiments, we assume that the sender will execute slow start whenever detecting network congestion. A C++ object simulator *simtcp1* was developed for this purpose. We simulated a 10-MB data transmission from a sender to a receiver in a continuous data stream. There are 8×8 send-receive buffer size combinations, ranging from 4KB to 52KB. We further divide the 64 combinations into seven areas, as shown in Table 2. Each area is constructed based on send-receive buffer sizes, and the likelihood that throughput deadlock would occur. For example, areas 1-3 is a deadlock-free region, and the three areas are further distinguished by their send-receive buffer size combinations. Moreover, areas 4-6 is a low-risk deadlock-prone region, and area 7 is a high-risk deadlock-prone region.

3.1 Deadlock-free region

In Fig. 3, we show the throughput performance for a scenario in area 1. We injected congestion every 500s, starting from time 0. Because the congestion duration is very short, only a very small throughput drop is observed in the curve with congestion. Throughputs for areas 3-4, which are shown here, exhibit similar behavior as in Fig. 3. Specific points about this set of experiments are as follows:

1. In area 2, because the send buffer is small, the MUADS estimate cannot be very large. As a result, even if the congestion window is set to one MSS after detecting congestion, the subsequent data sent out by the sender may still be 35% or higher of the MUADS estimate. Thus, the receiver could send back acknowledgments, and no deadlocks occur.
2. In area 3, the MUADS estimate cannot be larger than the actual receive buffer size, which is small in this case. As a result, the small amount of data sent by the sender may still be 35% or more of the MUADS estimate. Again, acknowledgments can be sent back.

3.2 High-risk deadlock-prone region

In Fig. 4, we observe three significant throughput drops, corresponding to three network congestion instants. The sender responds to congestion by performing slow start, thus decreasing the data rate. However, the MUADS estimate used by the receiver was derived from pre-congestion data. As a result, the sender's data is unable to meet 35% of the "incorrect" MUADS estimate, resulting in throughput deadlocks.

3.3 Low-risk deadlock-prone region

Here we show two cases in Figs. 5-6. Similar to the high-risk deadlock-prone region, both cases show significant throughput drops. However, the variance in the magnitude of throughput degradation is higher than that in Fig. 4. The main reason responsible for this is that the MUADS estimate depends very much on the connection's state just before the sender detects congestion. Thus, in some cases the MUADS estimate does not result in deadlocks but in some other cases it does.

4. A congestion-sensitive AAA (CS-AAA)

One approach to solving the deadlock problem in midst of network congestion is to reset the MUADS estimate and to perform sampling all over again to find the new value whenever congestion is detected. The main disadvantage of this approach is that deadlock still exists before sampling is completed. As a result, the sampling period needs to run for a long time, and the overhead incurred would be great.

4.1 Congestion detection and avoidance at receivers

We employ another method to tackle this problem, and we refer the enhanced AAA to as Congestion-Sensitive AAA (CS-AAA). First of all, the receiver now needs to detect network congestion. Whenever congestion is detected, the receiver performs a procedure on the MUADS estimate, similar to slow start at the sender. That is, the MUADS estimate is first reset to one MSS of the connection. The MUADS estimate is then incremented for every nonduplicate acknowledgment sent back to the sender, and this step continues until the MUADS estimate reaches the previous value when congestion is detected. We postpone the detail explanation of this procedure to a later stage in this section. Instead, we first consider the ways that a receiver can detect network congestion. There are three situations to consider:

1. A short-term congestion causes routers to drop a small number of data packets, and the receiver sends three duplicate acknowledgments back to the sender. In this case, the receiver resets the MUADS estimate to one MSS and starts the CS-AAA.
2. The receiver's acknowledgments are dropped by routers due to congestion. In this case, the sender times out and re-sends data packets. The receiver, therefore, receives duplicate data segments, and it resets the MUADS estimate to one MSS and starts the CS-AAA.
3. A longer-term congestion causes routers to drop a large number of data packets. The sender times out and re-sends packets. Thus, the receiver in this case can rely only on timing out the current MUADS estimate and starts the CS-AAA.

The algorithm that the receiver starts after resetting the MUADS estimate to one MSS in response to congestion is summarized as follows:

```
if (s_start) {
   MUADS = min (segsize, MUADS);
   if (MUADS > segsize){
      r_ssthresh = MUADS / 2;
      while (s_start)
   if (a new ACK sent out from the receiver)
            if (MUADS <= r_ssthresh) MUADS = MUADS * 2;
            else MUADS = MUADS + (segsize * segsize) /
                 MUADS + MUADS / 8;
      if (MUADS == r_ssthresh * 2) s_start = 0;
   };
   else s_start = 0;
};
```

s_start is a flag that signals the starting of CS-AAA, and *r_ssthresh* remembers the value of the MUADS estimate when congestion is detected. The logic of this algorithm follows closely the slow start algorithm at the sender.

4.2 Performance evaluation of CS-AAA

We performed simulation experiments for all three regions to evaluate the performance of CS-AAA. Since region 1 is deadlock-free, we consider regions 2-3 only. In Fig. 7 we show a case for the high-risk region and in Figs. 8-10, three cases for the low-risk region. We summarize in the following two major observations from the simulation results.

1. When the send buffer is relatively large, as shown in Figs. 7 and 10, CS-AAA responds to and recovers from network congestion much faster than AAA. In this scenario, it is highly likely that AAA will get into deadlock again when congestion occurs, because the MUADS estimate assumes a higher value due to a large send buffer. On the other hand, CS-AAA, by resetting the MUADS estimate to one MSS, is able to keep the packets flowing, and picks up much faster than AAA.
2. When the send buffer is relatively small, as shown in Figs. 8-9, CS-AAA attains similar throughputs as AAA at the beginning, but CS-AAA again responds to and recovers from network congestion much faster than AAA in the latter congestion. In this scenario. The MUADS estimate may or may not be large enough to cause deadlock. Apparently, when the first congestion occurs, the CS-AAA receiver is unable to detect network congestion (because the timeout value is too large);

therefore, there is no throughput gain by using CS-AAA. However, in the latter congestion the CS-AAA receiver is able to detect congestion and to respond to it promptly. This shows that CS-AAA is not penalized in the throughput performance even though it occasionally fails to detect congestion.

5. Conclusions and future works

In this paper we added a new component to AAA to handle the deadlock problem in midst of network congestion. The simulation experiments have shown that the new CS-AAA reacts to congestion much faster than AAA, and it also recovers from congestion much faster, thus resulting in a higher throughput performance. There are still a number of areas to explore. First, we need to investigate AAA's performance when sender uses fast recovery procedure. Second, CS-AAA can be further refined by allowing a receiver to take different actions in response to different types of congestion. Third, the effect of timeout value for detecting congestion at the receiver needs to be further investigated in order to increase the chance of accurately detecting congestion. Fourth, we will implement a prototype for this new TCP and measure the performance.

References

[1] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE J. Sel. Commun.*, vol. 13, no. 8, pp. 1465-1480, Oct. 1995.

[2] A. Bestavros and G. Kim, "TCP Boston: A Fragmentation-tolerant TCP Protocol for ATM Networks," *Proc. IEEE INFOCOM'97*, pp. 1212-1219, Apr. 1997.

[3] K. Moldeklev and P. Gunningberg, "How a Large ATM MTU Causes Deadlocks in TCP Data Transfers," *IEEE/ACM Trans. Networking*, vol. 3., no. 4., pp. 409-422, Aug. 1995.

[4] D. E. Comer and J. C. Lin, "TCP Buffering and Performance Over an ATM Network," *Internetworking: Research and Experience*, vol. 6, pp. 1-13, May 1995.

[5] J. Nagle, "Congestion Control on TCP/IP Internetworks," *RFC 896*, Jan. 1984.

[6] D. D. Clark, "Window and Acknowledgment Strategy in TCP," *RFC 813*, July 1988.

[7] Wing K. Leung and Rocky K. C. Chang, "Improving TCP Throughput Performance on High-speed Networks with a Receiver-side Adaptive Acknowledgment Algorithm," *Proc. SPIE's Intl. Symp. Voice, Video, and Data Commun. (Internet Routing and Quality of Service)*, pp. 4-13, Nov. 1998.

S/R	4KB	8KB	16KB	24KB	32KB	40KB	48KB	52KB
4KB	20 (21)	20 (21)	20 (20)	20 (20)	20 (20)	20 (21)	20 (20)	20 (21)
8KB	20 (20)	20 (20)	25 (25)	25 (26)	25 (25)	25 (26)	25 (25)	25 (26)
16KB	20 (20)	41 (41)	50 (50)	50 (50)	50 (50)	50 (50)	50 (50)	50 (50)
24KB	20 (20)	40 (40)	80 (80)	87 (89)	87 (88)	87 (87)	87 (88)	87 (87)
32KB	20 (21)	40 (40)	80 (82)	89 (95)	94 (106)	98 (107)	102 (115)	99 (118)
40KB	20 (21)	40 (41)	80 (80)	98 (103)	99 (103)	134 (137)	139 (144)	137 (146)
48KB	20 (20)	40 (42)	80 (81)	99 (102)	104 (102)	150 (159)	161 (170)	161 (168)
52KB	20 (21)	40 (41)	80 (80)	98 (98)	99 (105)	152 (160)	168 (165)	162 (170)

Original AAA (modified AAA) ▒▒ : Deadlock region

Table 1. Throughput for different send-receive buffer sizes (in KB per simulation sec) using AAA.

S/R	4KB	8KB	16KB	24KB	32KB	40KB	48KB	52KB
4KB	*Area 1*		*Area 2*					
8KB								
16KB	*Area 3*		*Area 4*			*Area 5*		
24KB								
32KB								
40KB			*Area 6*			*Area 7*		
48KB								
52KB								

Table 2. Seven areas of send-receive buffer size combinations.

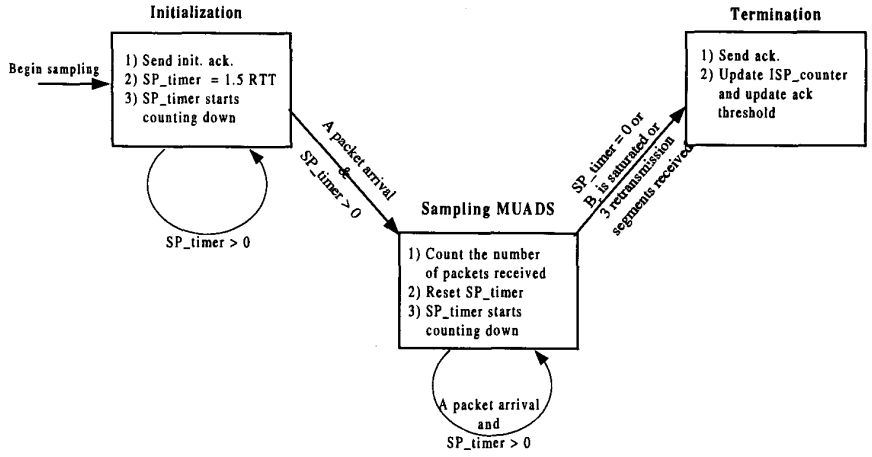

Fig. 1. State diagram of an AAA receiver during packet samplings.

if (| $MUADS_{current} - MUADS_{previous}$ | < *MSS*)
    ISP_t = 3 ISP_t*
    acknowledgment threshold unchanged
else
    ISP_t = 10
    else if ($MUADS_{current} > MUADS_{previous}$)
        acknowledgment threshold = 0.35 * $MUADS_{current}$

modified to

if (| $MUADS_{current} - MUADS_{previous}$ | < *MSS*)
    ISP_t = 4 ISP_t*
    acknowledgment threshold unchanged
else
    ISP_t = 0.5 * *ISP_t*
    else if ($MUADS_{current} > MUADS_{previous}$)
        acknowledgment threshold = 0.35 * $MUADS_{current}$

Fig. 2. A revised mechanism for computing threshold values for AAA.

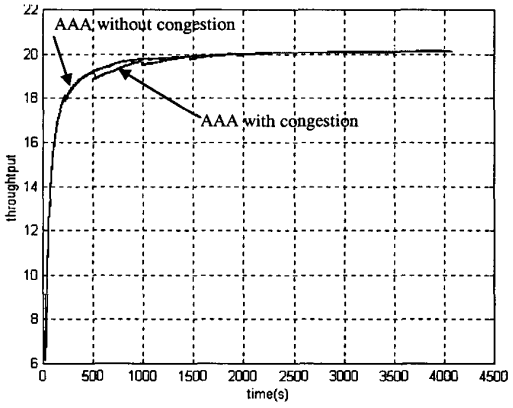

Fig. 3. Throughput for a 4-KB sender and a 4-KB AAA receiver (deadlock-free region).

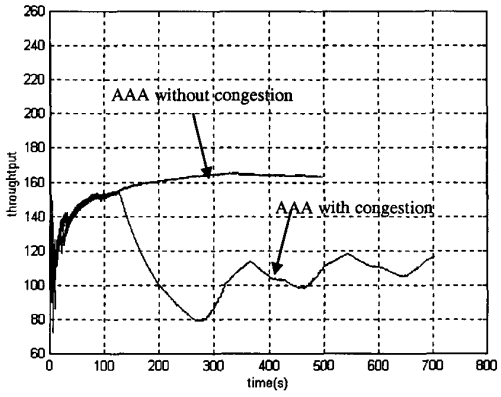

Fig. 4. Throughput for a 52-KB sender and a 52-KB AAA receiver (high-risk deadlock region).

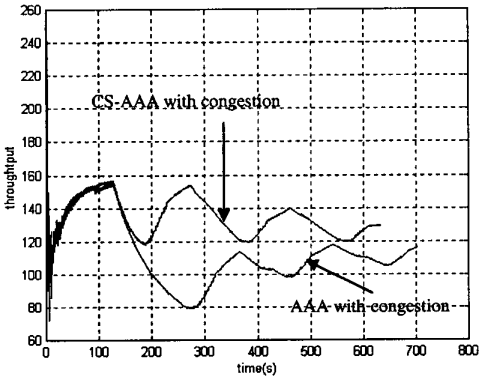

Fig. 5. Throughput for a 24-KB sender and a 24-KB AAA receiver (low-risk deadlock region).

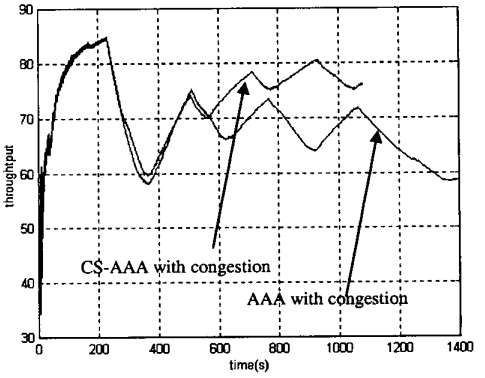

Fig. 6. Throughput for a 48-KB sender and a 24-KB AAA receiver (low-risk deadlock region).

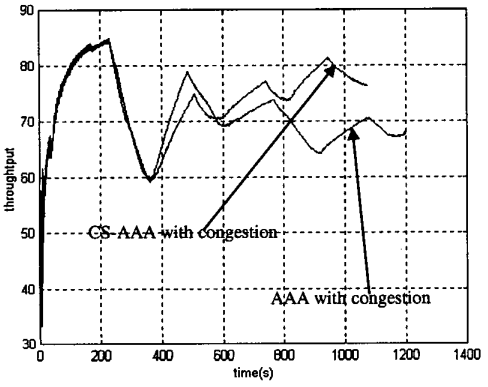

Fig. 7. Throughput for a 52-KB sender and a 52-KB receiver w/o CS-AAA (high-risk deadlock region).

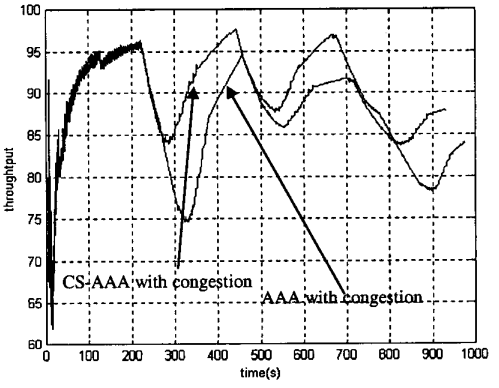

Fig. 8. Throughput for a 24-KB sender and a 24-KB receiver w/o CS-AAA (high-risk deadlock region).

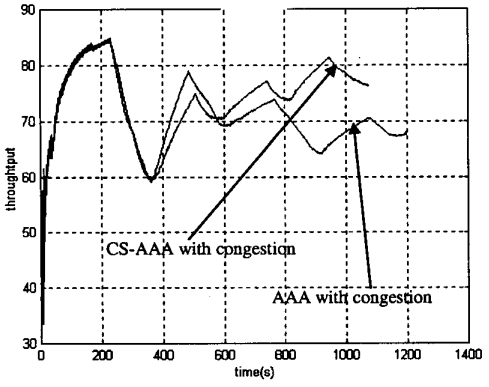

Fig. 9. Throughput for a 24-KB sender and a 48-KB receiver w/o CS-AAA (low-risk deadlock region).

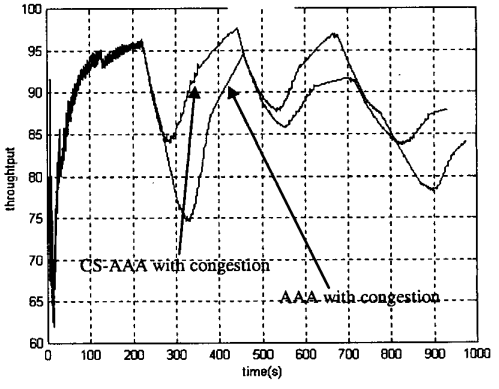

Fig. 10. Throughput for a 48-KB sender and a 24-KB receiver w/o CS-AAA (high-risk deadlock region).