ACK Delay Control for Improving TCP Throughput over Satellite Links *

Jing Wu [1] Yu Shi [2] Peng Zhang [1] Shiduan Cheng [1] Jian Ma [2]

[1] National Key Laboratory of Switching Technology and Telecommunication Networks
P.O.Box 206, Beijing University of Posts & Telecommunications, Beijing 100876, P.R.China
Fax: +86 10 62283412 E-mail: {wujing, chsd} @ bupt.edu.cn

[2] Nokia China R&D Center, No. 11, He Ping Li Dong Jie, Beijing 100013, P.R. China
Fax: +86 10 84222439 E-mail: {yu.shi, jian.janne.ma} @ nokia.com

Abstract

TCP has been proved to be working well in a variety of situations. However, in links of large bandwidth products, e.g., satellite links, TCP might underutilize the link and degrade the reliability. A number of revised algorithms have been proposed to improve TCP performance. In this paper, we propose a new algorithm to improve TCP throughpt without modifying TCP protocol. Since TCP uses the acknowledgments (ACKs) to adjust the sending rate, the basic idea of our scheme is to delay (or shape) ACKs traveling through a node where its forward connection is congested. The algorithm aims to fully eliminate packet loss caused by buffer overflow. We implement this scheme in the intermediate nodes connecting satellite links and make a simulation study on its effect. In addition, we test the proposed algorithm in real world. The results of both simulation and test show that the proposed scheme improves the TCP throughput by effectively controlling the acknowledgment flow.

1. Introduction

Internet plays an important role in the information infrastructure. Since satellites have distinctive advantages such as remote coverage, rapid deployment, distance insensitivity and immunity to terrestrial disasters, satellite networks will benefit the construction of global information networks [1]. However, satellite links possess some special characteristics such as higher transmission error and long propagation delay. In general, the propagation delay ranges from 25 ms to 250 ms for a one-way path. Satellite links have large bandwidth delay products and a significant impact on TCP performance. To achieve high performance, TCP is extended by adding some options, e.g., window scale option, timestamps option and selective acknowledgment (SACK) option [2][3]. Moreover, to guarantee the reliability, an additional scheme called Protect Against Wrapped Sequence Numbers (PAWS) is proposed to reject old duplicate segments that might corrupt an open TCP connection. These mechanisms have been proved to have great improvement on TCP performance and are necessarily required to carry TCP over satellite links [1][4].

Recently, a number of TCP mechanisms related to satellite have been summarized and generally discussed in [4][5]. These mechanisms focus on various aspects of TCP. For slow start algorithm, schemes of large initial window and byte counting are proposed to reduce the time needed to increase the window size to an appropriate level; For loss recovery, schemes of forward acknowledgement (FACK) and explicit congestion notification (ECN) are proposed to recover from loss quickly. Even spoofing and snooping that might damage the rule of TCP end-to-end control are presented. All these algorithms appear promising to improve TCP throughput in satellite networks, however, their efficiency and robustness need careful evaluation before they are exploited in real networks.

In this paper, we propose a traffic control scheme used in the routers adjacent to satellite links. The basic idea of this scheme relies on the fact that TCP uses ACKs to determine its traffic sending to the networks. Once congestion or overload is detected in the forward path, ACKs are delayed in the backward direction, which could slow down the source traffic rate. As a result, it is possible to quickly relieve the congestion and fully avoid packet loss. The original work was done in Nokia Research Center by Jian Ma etc. and presented in [6][7].

* The research is supported by Nokia Research Center.

In section 2, we explicitly present the background and motivation of our work. Section 3 presents the prototype of this scheme and discusses issues on its implementation. In section 4, we show the simulation model and the implementation of the scheme in the access router. We make a simulation study in section 5. Finally, the conclusions are given in section 6.

2. Motivation

Satellite networks yield huge fat pipes. In order to achieve good performance, TCP is extended to use larger maximum window size to match the fat pipe. On the other hand, TCP adjusts its window size according to a series of algorithms, e.g., slow start, congestion avoidance. These algorithms increase windows closely related to the round trip time (RTT). In a long delay environment, it takes a long time for TCP to increase window sizes to the appropriate level. As a result, TCP throughput might be significantly degraded during the period of slowly increasing window size. Table 1 illustrates the required large window to fully utilize the bandwidth on a MAN link (typically 5ms one-way delay), a LEO link (100ms one-way) and a GEO link (250ms one-way delay) respectively. The table also indicates how long a slow start takes to get to full link speed (assuming 1KB datagram) and how much data is transferred during the slow start phase. From the table, it shows that the links are underutilized for a relatively long time in links of large bandwidth delay products and consequently the throughputs are low in these cases.

Table 1. Summary of satellite and TCP interactions

	1.5Mb/s		
	MAN	**LEO**	**GEO**
Bandwidth delay product	15kb	300kb	375kb
Required large windows	No	No	Yes
Slow start time	0.01s	1.8s	5.6s
Slow start data (in bytes)	1.7K	76.7K	197.8K
Efficiency (%)	9.06	2.27	1.88
	45Mb/s		
	MAN	**LEO**	**GEO**
Bandwidth delay product	450kb	9Mb	22.5Mb
Required large windows	No	Yes	Yes
Slow start time	0.2s	3.5s	9.8s
Slow start data (in bytes)	115.9K	2.4M	6M
Efficiency (%)	10.3	12.2	10.88
	155Mb/s		
	MAN	**LEO**	**GEO**
Bandwidth delay product	155kb	31Mb	77.5Mb
Required large windows	Yes	Yes	Yes
Slow start time	1.9s	4.1s	11.3s
Slow start data (in bytes)	4.1M	8.3M	20.6M
Efficiency (%)	11.1	10.44	9.41

On the other hand, once packet loss occurs in a TCP connection, TCP will decrease its window size sharply and resume the slow start or congestion avoidance from a small window size. This will severely degrade TCP performance. In satellite networks, packet loss might happen in two cases: one case is that the packet is corrupted by the transmission error in the satellite links; the other is that buffer overflow occurs in the routers adjacent to the satellite links. Transmission error rates of satellite links are generally higher than those of terrestrial links. We will not focus on packet loss caused by transmission error. However, packet loss caused by buffer overflow might be eliminated. Therefore, novel traffic control scheme is really needed.

In this paper, we propose a scheme for delaying ACKs with the goal of fully preventing packet loss from buffer overflow [6],[7]. Since TCP is close-loop control based on feedback of ACKs, it is possible for congested nodes to avoid or alleviate congestion quickly by delaying ACKs. A typical deployment of delaying ACKs is shown in Figure 1. Apparently, the control loop is shortened by using this mechanism in the access routers. In the following parts, we call this mechanism Fast TCP.

It should be noted that our Fast TCP scheme differs from the scheme of spacing ACKs presented in [5]. Our scheme is implemented in routers rather than TCP sources. Its goal is to quickly relieve congestion while keeping the semantics of TCP. In addition, it is kept active at all time instead of only during the absence of ACKs.

3. Prototype of Fast TCP

Figure 2 shows the prototype of Fast TCP exploited in routers. For clarification, we separate the algorithm into three parts, i.e, congestion detection, ACK identification and delaying ACKs. Congestion detection is used to notify congestion before buffer overflow happens. ACK identification is required to separate ACK flow from normal data traffic. Delaying ACKs adjusts the rate of

ACK flow. Next, we discuss the implementation of these parts respectively.

3.1 Congestion detection

Router is usually designed on the basis of packet store and forward, which is in need of a huge buffer capacity to accommodate the burst of data flow. Congestion might be detected by means of supervising the queue length. Congestion is detected when the queue length exceeds a prefixed threshold. Moreover, some advanced traffic prediction methods might be employed.

3.2 ACK identification

To identify an ACK, a router should check the ACK bit in TCP/IP packet. However, ACK information might be piggybacked in data packets which are prohibited to delay. Thus, it is necessary to separate the ACK information from the data packets. We could clear the ACK bit in the data packets and generate new ACK packets by copying the ACK information from the data packets. By using Fast TCP, IP packets are monitored and ACKs are filtered. As shown in Figure 2, a separate buffer is employed to accommodate filtered ACKs.

3.3 Delaying ACKs

The flow of filtered ACKs is shaped according to the policy of delaying ACKs. In our method, the leaking rate is calculated explicitly, and ACKs are spaced according to the rate. On the other hand, because slow start scheme and congestion avoidance scheme increase window size in different ways, they do not have the same effect on delaying ACKs. For illustration, in slow start phase, upon receiving an ACK, window size increases by the size of one data packet and as a result two packets are released, thus, ACKs should be spaced with an interval larger than that in congestion avoidance phase. In [8], it is suggested that the intermediate nodes follow the window sizes of TCP sources in order to explicitly delay ACKs. However, in real networks, it is difficult for routers to keep pace with window sizes of TCP sources due to various implementation of TCP sources. Therefore, we propose to just consider slow start because it is more aggressive than congestion avoidance.

3.4 Implementation Challenge

In fact, there are some factors in real networks that might invalidate the employment of Fast TCP. Some of them are listed below.

IP networks are connectionless which means that ACK packets might travel along different paths from the forward data paths. It seems difficult for routers to determine whether data packets and relevant ACKs share the same path. However, in satellite networks, two routers are usually connected via a single satellite link. Thus, our Fast TCP scheme can be used. Particularly, this mechanism might be implemented in the routers as an enhanced policy and can be optionally enabled depending on the location of the routers deployed.

In TCP, delayed ACKs [10] allow data receivers to refrain from sending ACK for every incoming data segment. Although a delayed ACK can reduce the number of segments sent by the receiver, excessive delay on ACK can disturb the round-trip timing and inherent self-clocking of TCP. Because Fast TCP uses ACK flow to control data traffic in the forward direction, ACK flow disturbed by delayed ACK might affect the efficiency of Fast TCP.

Since multiple TCP connections might travel along the satellite link, it remains an open issue that whether Fast TCP controls ACKs of different TCP connections separately. We believe that classifying different types of TCP connections will produce an effective control because different TCP connections possess distinguishing traffic patterns and have different effects on networks. For example, TCP connections of file transfer service have greater impact on networks than those of telnet or rlogin. However, this requirement is hardly fulfilled due to its complexity and the increasing workload of routers. Therefore, we propose the general Fast TCP to control ACKs without identifying TCP connections.

In summary, although Fast TCP might be held back by the above issues, it truly brings about quite a few distinct advantages, e.g., reducing buffer requirement, smoothing source traffic and fully avoiding buffer overflow etc. These advantages will definitely contribute to the network performance and will be examined by simulation study in the next part.

4. Simulation Model and Implementation of Fast TCP

4.1 Simulation model and configuration

As shown in Figure 3, we use a simple network configuration in which a single TCP connection is established between a pair of workstations through two routers and one satellite link. We just consider one-way data transfer where TCP source delivers data packets and TCP destination replies by ACKs. It is assumed that all

links are symmetric links which have identical bandwidth and propagation delay in both directions. The satellite link is configured to 30 Mbps and its propagation delay is set to 130ms as in a Medium Earth Orbit (MEO) environment. The terrestrial links between the workstations and the adjacent routers are configured with the rate of 150 Mbps (i.e. five times than that of satellite link) and with the propagation delay of 1 ms. Thus, the Bandwidth-Delay Product (BDP = Bandwidth * RTT) of the channel between source and destination is approximately 1 M bytes.

The Maximum Segment Size (MSS) of TCP is defined as 536 bytes, the default value in most situations. In order to fill up the large pipe, we enlarge the Maximum Window Size to 1M bytes by using the window-scale option. We also set the initial threshold discriminating slow start phase from congestion avoidance phase to the same value as the Maximum Window Size. Thus, congestion might be aroused by aggressively increasing window in slow start phase. On the other hand, it should be mentioned that although algorithms of PAWS and SACK are essentially required in satellite networks, we omit them to simplify our implementation. We believe the results obtained in this paper could be naturally extended to the general cases with these algorithms. Additionally, to focus on our objective, we assume all the physical links including the satellite link to be ideal links without bit error.

4.2 Implementation of Fast TCP

The Fast TCP scheme is implemented in the routers adjacent to satellite link. It monitors data traffic in the forward channel and control ACKs in the backward channel. We use a simple method to detect congestion, which is that congestion is notified when the buffer occupancy exceeds a fixed threshold. Here, we set a small threshold of 50Kbytes. ACK identification is easy in our configuration because no data packets traverse in the backward path. A simple policy of delaying ACKs is given, that is, when no congestion occurs, ACKs leak at a normal rate, otherwise, at a fraction of the normal rate. We set the normal rate equal to the service rate of the data packet in the forward buffer. The fraction is set to half so that the leaking rate is halved when congestion happens.

5. Simulation Study [*]

5.1 Case 1 : large buffer capacity

In this case, we set the buffer size to 1M bytes which is large enough to avoid packet loss whether Fast TCP is working or not. However, it is possible for Fast TCP to avoid buffer overflow in smaller buffers and we will show it in the next case.

We first present the figures depicting congestion window size, forward queue length and backward queue length in the cases of with/without Fast TCP. As shown in Figure 4, there are aggressive bursts in the forward buffer in the case without Fast TCP. The bursts become larger and larger with the growth of window size, then reaches approximately half of the maximum window size. The bursts are obviously caused by slow start algorithm of TCP protocol. On the contrary, as shown in Figure 5 of the case with Fast TCP (note the scale), bursts in the forward queue length are truncated by Fast TCP to a very low level about 50K bytes, which is the fixed threshold. Therefore, in order to avoid packet loss, the forward buffer capacity in the case without Fast TCP should be much bigger than that in the case with Fast TCP. In fact, Fast TCP greatly decreases the forward buffer occupancy at the cost of a little increase of backward queue length.

Then, we show the forward queue length and the sequence number of data packets sent by TCP with Fast TCP enhancement in Figure 6. Clearly, when queue length of the forward buffer exceeds the fixed threshold, the TCP source slows down its sending rate. As a result, the queue length decreases. It can be seen that the forward queue length oscillates around the center of the fixed threshold so that it is kept within a small value. Apparently, the source traffic is smoother.

5.2 Case 2 : small buffer capacity

In this case, we use the routers with small buffer capacity. The buffer capacity is 250K bytes which is much smaller than that of the previous case.

The capacities of the buffers in routers are smaller than BDP. Normally it will cause low utilization of the bandwidth and packet loss will probably happen caused by buffer overflow and packet discard when Fast TCP does not work. By using Fast TCP, the probability of buffer overflow and packet discard is significantly reduced. The buffer capacity minus the fixed threshold is

[*] The simulation tool is OPNET.

large enough to accommodate the packets remaining at the links in the loop from Router 1 to Source and return to Router 1. As a result, the congestion window of the source is not reduced and the bandwidth utilization as well as throughput is improved. In Figure 7, the forward queue length with is compared to that without Fast TCP. It can be seen that Fast TCP reduces the bursts significantly and no overflow happens. In Figure 8, the total buffer occupancy with Fast TCP is compared to that without Fast TCP. It can be seen that backward queue length increases only a little. And the total buffer occupancy is smoother. In Figure 9, received segment number of destination with Fast TCP is compared to that without Fast TCP. It shows that Fast TCP improves the throughput. In Figure 10, received ACK number of source with Fast TCP is compared to that without Fast TCP. It can be seen that the arriving rate of ACKs from 14 to 14.6 second when Fast TCP is enabled is smaller than that when Fast TCP is disabled. The subtle shaping of ACK flow greatly change the behavior of forward data flow.

6. Test on Internet

In order to get more experience and see how the Fast TCP works in the real world, we have managed to implement it in UNIX user space and have gained some primary results so far. Our program serves as a router in which the Fast TCP mechanism can be enabled or disabled. Thus we can compare the performances.

6.1 Network topology and test environment

The topology is simple as shown in figure 11.

The traffic is generated by Netperf, a well-known network benchmark. TCP connection are established between source and destination. The two machines are running 4.4 BSD with Fast Retransmit and Fast Recovery Algorithms. (TCP is robust with FRR which the simulation tool has not implemented.)

And we do not send data from destination to source and thus avoid ACK piggyback. All the packets from source are data and should be put into a data FIFO queue. All the packets from destination are ACKs and should be put into an ACK FIFO queue.

The RTT (Round Trip Time) is about 600 milliseconds. Our program goes all out to catch packets and put them into the queue, but it forwards them at certain intervals. Thus, packets in buffer begin to accumulate and congestion occurs. During the test, we varied some parameters to see Fast TCP's performance in different environments, such as traffic load, MSS (Maximum Segment Size), test duration, router's forwarding rate, etc. Here, we give three scenes to demonstrate the principle of Fast TCP.

6.2 Case 1: single connection

This is a single connection test which lasts 50 seconds. The MSS is set to 256 bytes. The send and receive window size is set to 17K bytes. The router's forwarding rate is constant which equals 20 packets/second.

Delay algorithm:

interval=0.1 second

if buffer < threshold, then delay ack by interval;

if buffer >= threshold, then delay ack by 2*interval;

(If Fast TCP is disabled, every interval, we launch 2 packets, one from forward buffer, one from backward buffer.)

We disable Nagle algorithm in TCP (set TCPNODELAY in socket) as most HTTP servers and FTP servers do.

Table 2. Test result of single connection case

	No Fast TCP	Fast TCP
Max queue length (packets)	45	45
Threshold (packets)	N/A	42
Throughput (kbits/sec)	12.70	18.91

The buffer is counted by packet number, so 17K bytes sending buffer and 256 bytes MSS cause a buffer utilization less than 66. Some packets can be in Internet pipe. In our test, the queue length reached 55. Here we set the max queue length to 45 to make it congested. From this table, we can see that Fast TCP improved the throughput of a single TCP connection by 48.9%. In this case, Fast TCP is very successful at congestion control.

From figure12, we can see it completely avoids buffer overflow and packets dropping, while normal router drops 12 packets continuously because of the unawareness of congestion.

In figure 13, Fast TCP curve shows that the buffer occupation keeps around the threshold, which is 42, till the end of connection. High buffer occupation indicates that the data sender does not slow its sending rate.

When Fast TCP is disabled, there are packets dropped, which cause the data sender to slow down its sending rate. Thus the buffer occupation decreases to zero and recovers slowly.

In figure 14, when Fast TCP is disabled, the router waits for packets coming and even stops forwarding for some time because of the low buffer occupation. It takes about 12 seconds for the source to resend all the lost packets and recover.

Things are different in router with Fast TCP. It keeps forwarding packets and achieves higher throughput.

6.3 Case 2: multiple connections

This is a multi-connection test which lasts 15 seconds. The Maximum Segment Size (MSS) is set to 1440 bytes. The send and receive window size is set to 17K bytes. The router's forwarding rate is constant which equals 40 packets/second.

Delay algorithm:

Interval=0.05 second

if buffer < threshold, then delay ack by interval;

if buffer >= threshold, then delay ack by 2*interval;

(If Fast TCP is disabled, every interval, we launch 2 packets, one from forward buffer, one from backward buffer.)

We disable Nagle algorithm in TCP.

In our test, 4 TCP connections are established successively by UNIX shellscript. They compete for the shared bandwidth. The result table shows the throughput of each connection as well as the total one.

We found that in multi-connection, light congestion will not reduce the total throughput very much because some connections lose bandwidth while others pick them up. However, that will cause the unfairness. Figure 15 gives us a intuitionistic impression of improvement in both throughput and fairness. We have 3 similar views in figure 16,17 and 18 as those in figure 12,13 and 14, which show that the traffic is well controlled.

Table 3. Test result of multiple connection case

	No Fast TCP	Fast TCP
Max queue length (packets)	30	30
Threshold (packets)	N/A	25
Connection 1 Throughput (kbits/sec)	30.81	34.32
Connection 2 Throughput (kbits/sec)	22.93	36.51
Connection 3 Throughput (kbits/sec)	27.03	35.33
Connection 4 Throughput (Kbits/sec)	30.37	34.95
Total Throughput (kbit/sec)	111.14	141.11

6.4 Case 3: multiple connection

In the real world, Fast TCP might not avoid all of the buffer overflows because of the complexity of network traffic, but this technique can also be attractive if it can avoid some congestion, which will also contribute to network performance.

Here we give an example. The intention is to see the performance of the Fast TCP under some WWW traffic. On a UNIX machine, the Apache web server is running while we use Teleport Pro on another Windows 95 machine to retrieve web pages. Note that ACK piggyback is ignored. Teleport Pro is a software for offline browsing, which uses multi-thread technique to simultaneously transmit a number of files from a web server. We use it to establish many TCP connections easily.

The test lasts 150 seconds. More than 100 files are retrieved and the Fast TCP achieves better throughput. In the algorithm, the buffer threshold is set a bit higher to make buffer overflow happen. That means the router with Fast TCP also drops packets.

From figure 19 and 20, We can catch the following points:

1. There are many strong bursts during the process.
2. With Fast TCP, the router drops much less packets than it does without Fast TCP.
3. The bursts are smoothed by the control of the Fast TCP.

7. Conclusions

The proposed Fast TCP algorithm is a new enhancement to TCP traffic control. Fast TCP focuses on traffic control and congestion avoidance in access routers. It is better than some other existing enhancement in the following aspects:

Fast TCP is implemented in the intermediate nodes of networks, e.g. routers, gateways, switches, and does not need any amendment of terminal implementation of TCP. So Fast TCP can adapt to various versions of TCP and is easily deployed in the networks.

Fast TCP reduces the traffic control loop from end-to-end to the path from the source to the intermediate node where Fast TCP is implemented and back to the source. So the reaction of traffic control is quicker and consequently the control result is better.

The requirement of forward buffer occupancy is reduced at the cost of a small increase of backward buffer occupancy, and total buffer requirement is reduced. So by using Fast TCP, the intermediate nodes of networks with

small buffers can reach the same throughput as those of large buffers.

The queues in the intermediate nodes of networks are smoother. So the end-to-end delay as well as the delay jitter of TCP traffic is reduced. Applications such as Web Browser can benefit.

The difficulty of the proposed Fast TCP is that the intermediate nodes need to recognize ACK information encapsulated in IP packets, and some manipulation of IP packets may be required. The buffer management is also more complex. In addition, the threshold and delay time need to be adjusted to achieve better performance. The delayed ACK might accumulate and be timeout, which can have a bad impact on TCP traffic. Our further study is to develop a self-learning algorithm for Fast TCP.

References

1. I. F. Akyildiz and S. H. Jeong, "Satellite ATM Networks: A Survey," IEEE Communication Magazine, July 1997.
2. W. R. Stevens, "TCP/IP Illustrated, Volume 1: The Protocols," Addison-Wesley, Reading, Massachusetts, 1994.
3. M. Mathis, J. Madhavi, S. Floyd, A. Romanow, "TCP Selective Acknowledgment Options," Internet RFC 2018, October 1996.
4. C. Partridge, T. J. Shepard, "TCP/IP Performance over Satellite Links," IEEE Networks, Sept/Oct 1997.
5. M. Allan, "Ongoing TCP Research Related to Satellites," Internet draft, May 1998.
6. J. Ma, "Interworking Between TCP and ATM Flow Controls," ATM Forum/97-0960, Dec 1997.
7. J. Ma, "Performance Enhancement by Controlling ACKs in TCP/IP over Satellite Network," Inet'98, 1998.
8. P. Narvaez, and K.-Y. Siu, "An Acknowledgement Bucket Scheme for Regulating TCP Flow over ATM," Globecom'97, Nov 1997.
9. A. Koike, "TCP flow control with ACR information," ATM Forum/97-0758R1, Dec 1997.
10. R. Braden, "Requirements for Internet Hosts -- Communication Layers," Internet RFC 1122, Oct 1989.

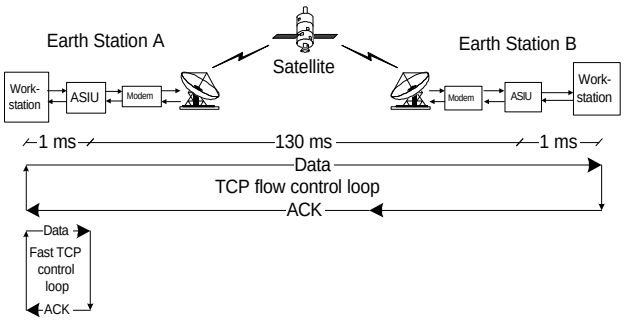

Fig 1. Flow control in satellite networks

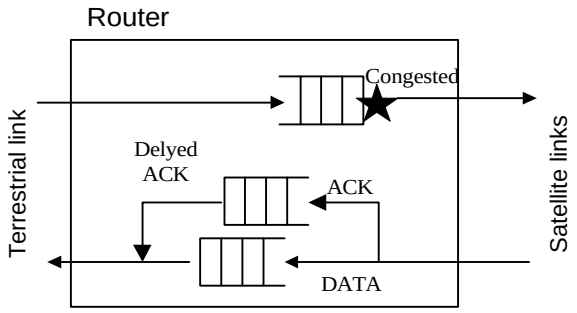

Fig 2. Prototype of Fast TCP

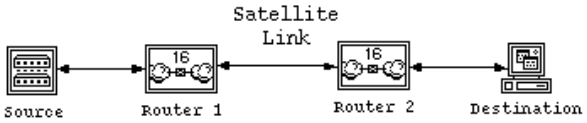

Fig 3. Simulation model

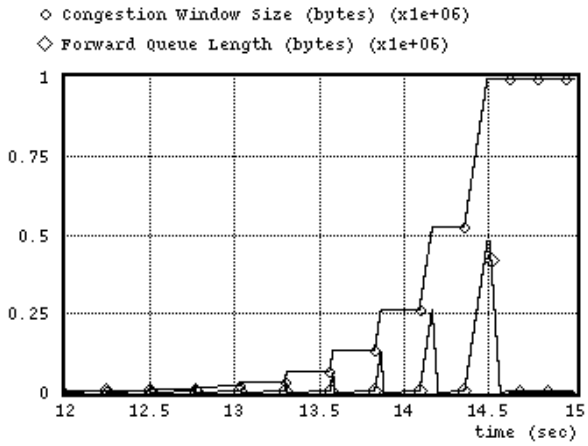

Fig 4. The case without Fast TCP algorithm

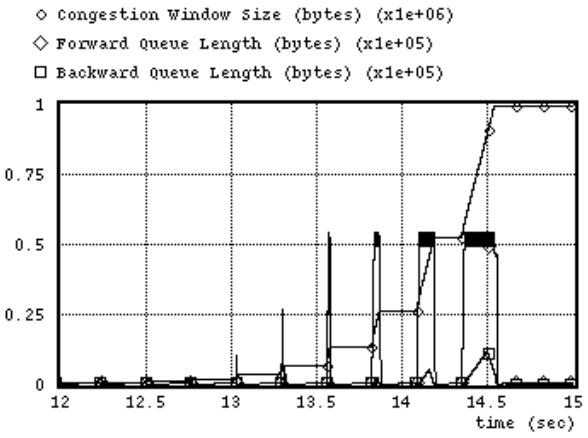

Fig 5. The case with Fast TCP algorithm

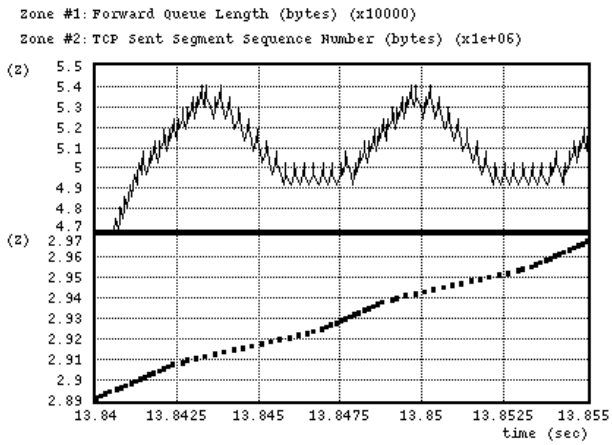

Fig 6. TCP sending data packets with Fast TCP

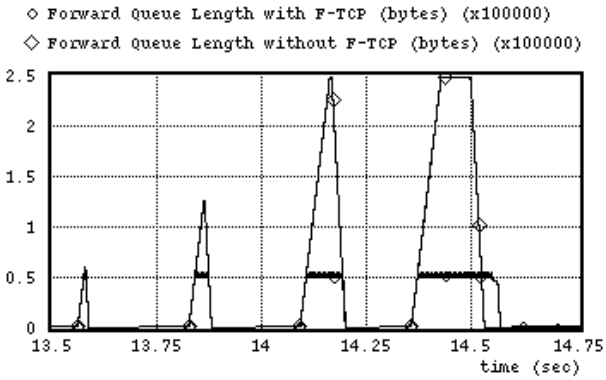

Fig 7. Forward queue length

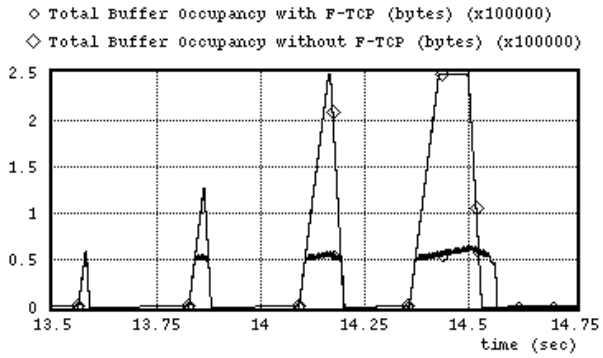

Fig 8. Total buffer occupancy

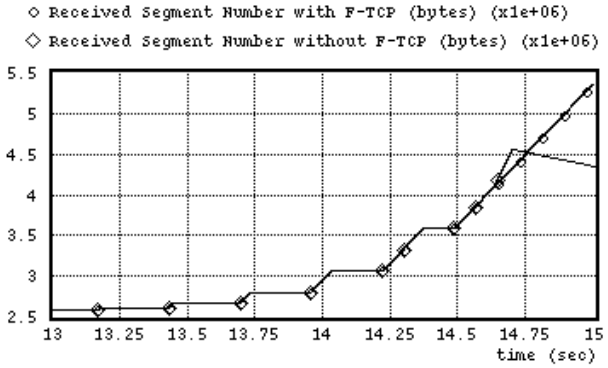

Fig 9. Received segment number of Destination

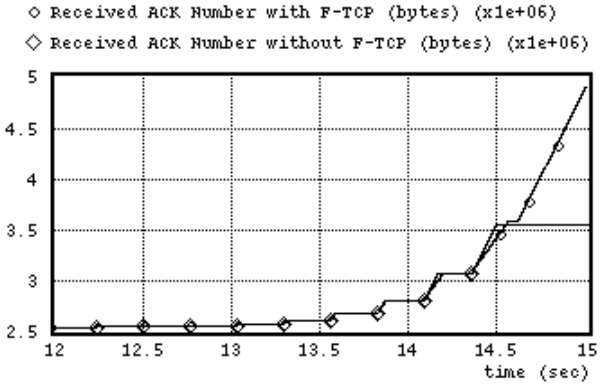

Fig 10. Received ACK number of source

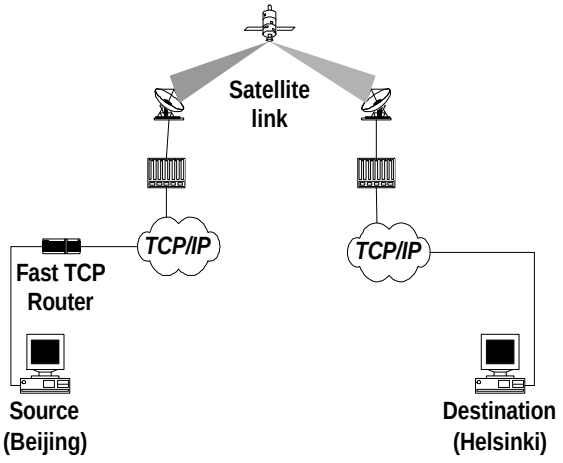

Fig 11. Network topology of test

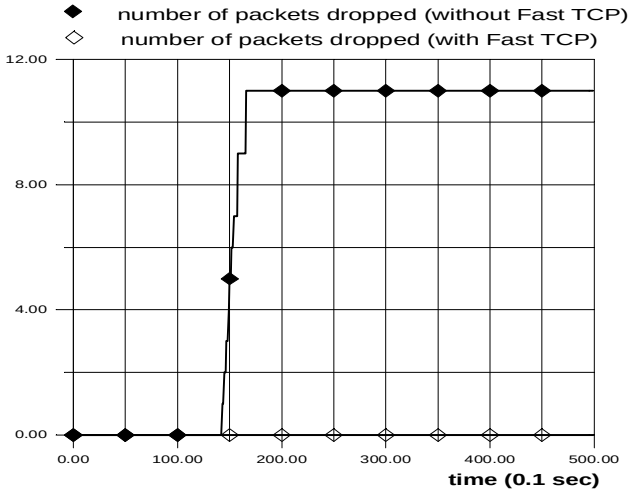

Fig 12 Number of packets dropped

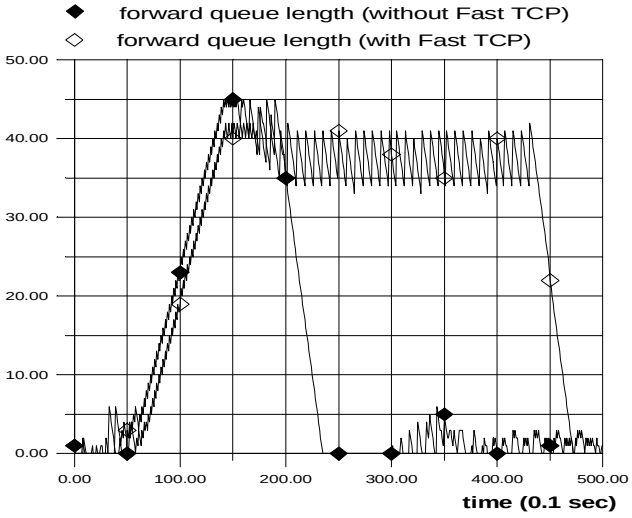

Fig 13. Forward queue length

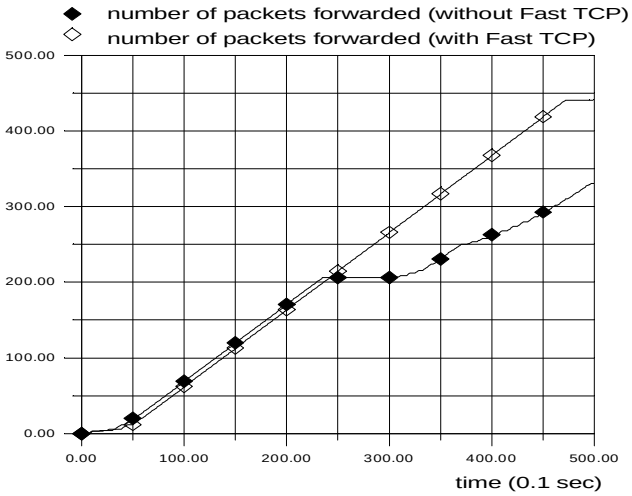

Fig 14. Forwarded packets number

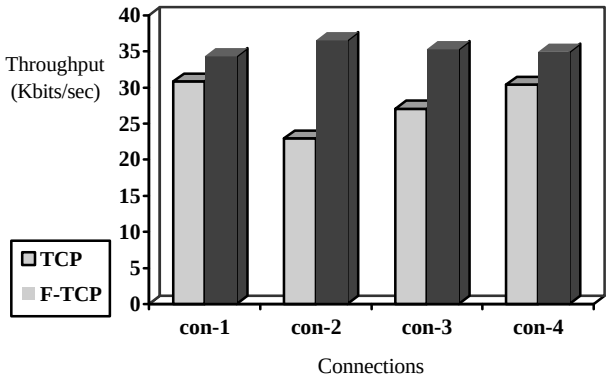

Fig 15. Throughput of each connection

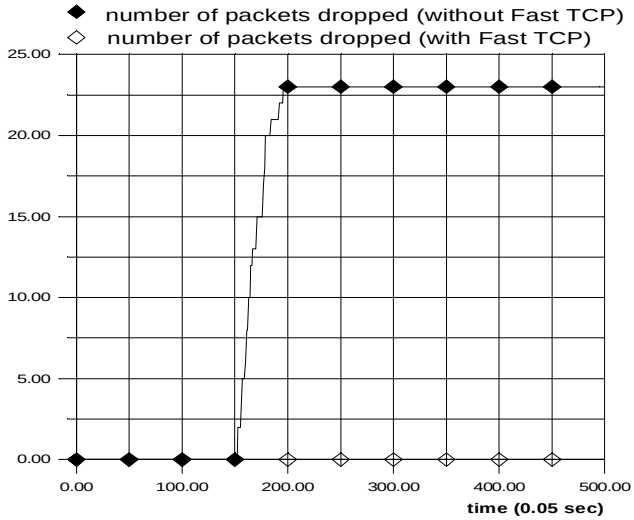

Fig 16. Number of packets dropped

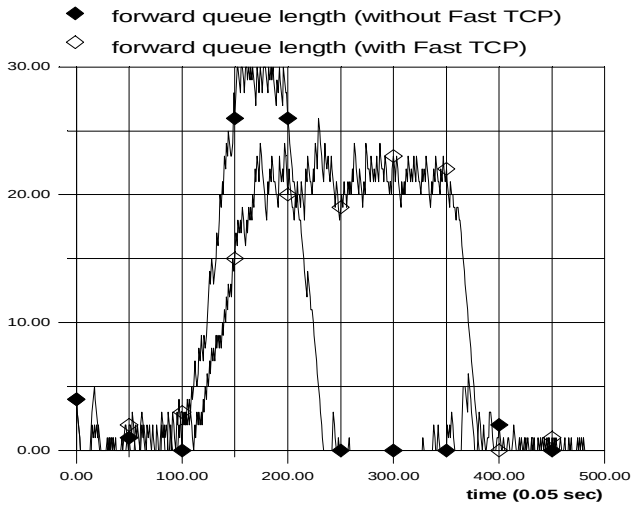

Fig 17. Forward queue length

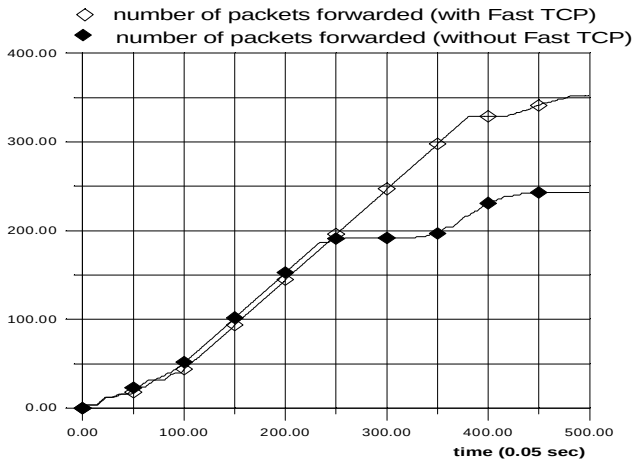

Fig 18. Forward queue length

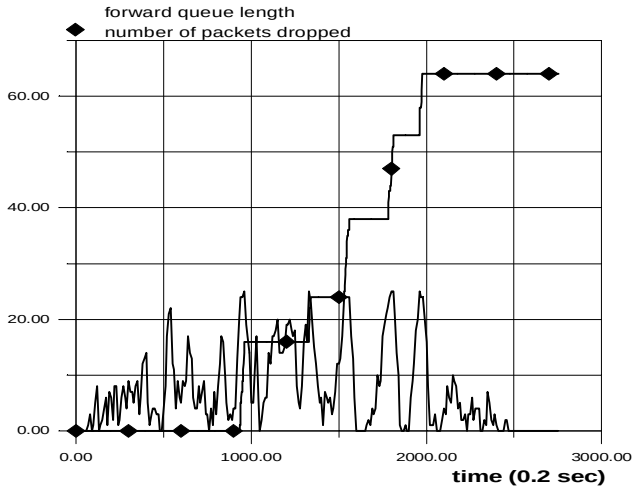

Fig 19. Forward queue length and number of packets dropped without Fast TCP

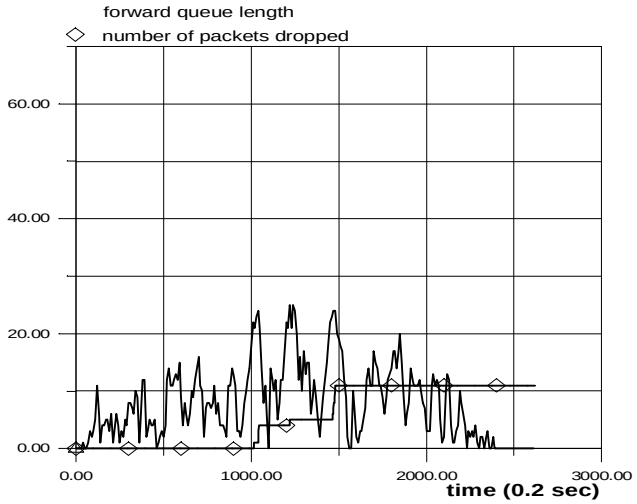

Fig 20. Forward queue length and number of packets dropped with Fast TCP