A Novel TCP Congestion Control (TCP-CC) Algorithm for Future Internet Applications and Services

Haiguang Wang and Winston Khoon Guan Seah

Institute for Infocomm Research, 21 Heng Mui Keng Terrace
Singapore 119613
{wanghg, winston}@i2r.a-star.edu.sg

Abstract. In this paper, we present a novel congestion control algorithm for the Transmission Control Protocol (TCP) for the future Internet. Our assumption of future Internet is that, with the increasing quality of service (QoS) requirements, per-flow packet scheduling (per-flow here refers to per TCP or UDP connection) will replace the current first-come-first-serve algorithm used in routers. Based on the assumption, we design a new congestion control algorithm. In our TCP-CC algorithm, each connection adjusts the size of the congestion window according to the size of its packet queue at the bottleneck router. Thus, the queue size for each connection at the bottleneck router is within a controlled range. We show that congestion loss is effectively reduced compared to the current TCP congestion algorithm.

1. Introduction

TCP has been the dominant transport layer protocol of the Internet since 1970s. As a reliable end-to-end data transmission protocol, it is used by many applications such as WWW, FTP, Telnet, Email, etc. It enables the computers on separate networks to communicate and share data with each other. However, with the exponential growth of computer networks, congestion becomes a severe problem [1]. Congestion control algorithms are designed to reduce the packet drops at the bottleneck router.

Historically, TCP's congestion control algorithms are based on the notion that the network is a black box [1] [2]. A TCP sender increases its congestion window size (a value that determines the maximal size of data that can be pumped into the network by the sender) gradually to probe the available network capacity. When packet loss occurs, the size of the congestion window is decreased to relieve the network from congestion [3]. However, using packet loss as a primary indication of congestion may not always be a good scheme. Sometimes, it causes the packet buffer of routers to overflow unnecessarily even if there is only one TCP connection on the network. It is inefficient as the dropped packets have to be retransmitted and bandwidth is wasted. It may also affect the performance of the real-time traffic on the Internet [4]. Therefore, new congestion-control algorithms that avoid using packet loss as primary indication of congestion are required.

Another motivation for designing a new congestion control algorithm is the change in the nature of Internet applications. The increasing demand for QoS beyond best-effort makes the widely used FCFS packet scheduling algorithm insufficient for the future Internet [5] [6]. It does not allow a router to give some sources a lower delay than others nor prevent malicious sources from eating up excessive bandwidth. Protocols for supporting QoS such as the Resource ReSerVation Protocol (RSVP) [7] require the current FCFS packet scheduling algorithm to be replaced with per-flow ones [8] [9] [10].

In this paper, we propose a new TCP congestion control algorithm**,** which we refer to as **TCP-CC,** for networks that support per-flow packet scheduling. The congestion window of a connection is adjusted dynamically according to the size of its packet queue at the bottleneck router. Thus, queue size for each connection is controlled within a predefined range at the bottleneck router. Compared to the congestion control algorithm used by the current TCP implementations, which increases the congestion window until packet loss occurs, TCP-CC saves the expensive buffer space at the bottleneck router and reduces congestion loss. It is superior to TCP Vegas [11] as it can identify the direction of congestion. While similar to the one used in TCP Santa Cruz (SC) [12], it overcomes the unfairness problem inherent in TCP SC.

The paper is organised as follows. Section 2 discusses the previous work in congestion control for TCP. Section 3 describes our congestion control algorithm. Section 4 shows the results obtained from our test-bed and the performance improvements compared to other algorithms. Finally, Section 5 summaries the paper.

2. Previous Work

Congestion control has been an active research area for TCP since 1986 when the Internet was confronted with the congestion collapse for the first time. Many solutions with [13] [14] [15] or without [1] [11] [12] [16] [18] the involvement of routers have been proposed since then.

For router-based schemes, the router either drops packets when the size of the router buffer reaches a predetermined value (Random Early Detection (RED) gateway) [13] or notifies the sender by setting a flag in the packet (Explicit Congestion Notification (ECN)) [14]. This consequently adjusts the congestion window. The scheme proposed in [15] changes the size of receiver's advertised window such that the packet rate is slowed down when network enters the congestion state. The problem of router-based algorithms is that the router support is needed.

End-to-end congestion control algorithms adjust the packet rate according to some indication from the network. The earliest congestion control algorithm uses packet loss as an indication of congestion [1]. The congestion window is gradually increased until packet loss happens, then the window size is halved. This cycle of increase-drop-decrease period is repeated. The problem of this algorithm has been discussed in Section 1.

An improved congestion control algorithm, TCP Vegas [11], adjusts the congestion window according to the difference between the expected and actual throughput. The expected throughput is calculated from the minimal round-trip time (RTT) previously observed and the actual throughput is calculated according to the current RTT. When the differences are below α or above β, where α and β represent too little or too much data in flight, the size of congestion window is increased or decreased accordingly. Similar algorithms have been proposed by [16] [18].

The problem of using RTT in congestion control is the traffic fluctuation on return link, which may lead to erroneous decisions in congestion window adjustment. To solve this problem, TCP-SC [12] introduced the concept of relative delay, which is the increase and decrease in delay that packets experience with respect to each other as they propagate through the network. It estimates the number of packets of a

connection in the bottleneck router queue using the relative delay and the congestion window is adjusted when too little or too many packets are in the queue. However, in a multiple connection case, bandwidth is shared unfairly among the connections.

The above congestion control algorithms do not consider the features of per-flow packet scheduling. A congestion control algorithm for per-flow packet scheduling has been proposed in [16]. Back-to-back pair packets are sent out and the inter-arrival time of the acknowledgement packets (ACK) are measured to estimate the bandwidth assigned to this connection and the sending rate is adjusted according to the estimated bandwidth. While it has the advantages of per-flow packet scheduling, the problem of this algorithm is that the variation of the ACK's delay on the return link may adversely affect its performance.

3. TCP congestion Control Algorithm

Our TCP congestion control algorithm is based on the fundamentals of queuing theory and applies the concept of relative delay and Back-to-Back pair packets in queuing-length detection at the bottleneck router.

3.1 Assumptions

The basic assumption of our algorithm is that per-flow packet scheduling algorithm is used in the routers to provide QoS support. Per-flow packet scheduling algorithms have been introduced in [8] [9] [10] and Deficit Round Robin (DRR) [10] is implemented in a product [20]. Our assumption of the future Internet is as follows.

- For a given duration during which congestion occurs, there exists only one bottleneck router on the link between the sender and receiver.
- DRR is used in packet scheduling at the bottleneck router.
- Each TCP connection is treated as a flow.
- When congestion happens, the router drops packets of the flow with longest queue.
- The sender always has data to send.

3.2 The New Algorithm

Similar to TCP-SC [12], the critical point of our TCP-CC algorithm is in calculating the packet-queue-length (PQL) (including the packet that is being served) of a connection at the bottleneck router. According to the Little's theorem:

$$N = \lambda T \tag{1}$$

where N is the average number of customers in a system, λ is the arrival rate, and T is the mean time each customer spends in the system. For a computer network, we can consider the router as the server and the packets are the customers, and if the sender knows the packet arrival rate λ and T, then it can estimate the value of PQL at the bottleneck router. Assuming the packet arrival rate from a connection is λ_{t_j} and the mean time a packet spends at the bottleneck router is T_j when the j^{th}

packet reaches the bottleneck router. Then, we can estimate PQL of this connection at the bottleneck router, as follows:

$$N_{t_j} = \lambda_{t_j} * T_j \tag{2}$$

where N_{t_j} is the PQL at time t_j. For a connection, λ_{t_j} can be derived as:

$$\lambda_{t_j} = \frac{\#\ pkts \quad received}{R} \tag{3}$$

where R is the duration between the arrival times of the first and last packets in the period for which λ_{t_j} is being computed.

Therefore, if we can get the value of T_j, then we can know the PQL of this connection. One way is to let the router attach the time in the packet. However, this method may consume expensive computing resource at the router.

The concept of relative delay introduced by TCP-SC can tell us the additional delay experienced by j^{th} packet compare to the i^{th} packet. If we know the delay, T_i, of the i^{th} packet, then we can derive the delay of the j^{th} packet from (4) and (5).

$$T_j = T_i + D^F_{j,i} \tag{4}$$

$$D^F_{j,i} = (R_j - R_i) - (S_j - S_i) \tag{5}$$

where j is greater than i, $D^F_{j,i}$ is the relative delay of the j^{th} packet compare to the i^{th} packet, R_j and R_i are the arrival times of the j^{th} and i^{th} packets at the receiver side, and S_j and S_i are the sending times of the j^{th} and i^{th} packets at the sender side.

The problem lies in determining the value of T_i. Back-to-Back pair packets used in [16] can be used in estimating the value of T_0 as follows. Firstly, after the connection is established, the size of congestion window is set to 2. Thus the Back-to-Back pair packets, namely, packet 0 and 1, are sent out. The bandwidth assigned to this connection at the bottleneck router can then be calculated as follows:

$$b_1 = pkt\_size / R_1 - R_0 \tag{6}$$

where b_1 represent the bandwidth assigned to this connection when packet 1 is served by the bottleneck router. As the time interval between packet 0 and 1 are served is very short, we assume the bandwidth assigned to this connection does not change too much, that is $b_0 \cong b_1$. As packet 0 is the first packet of this connection, no packet is in the queue of this connection at the bottleneck router when it reaches. Thus, T_0 can be estimated as follows.

$$T_0 = \frac{pkt\_size}{b0} \approx \frac{pkt\_size}{b1} = R_1 - R_0 \tag{7}$$

Since R_0 and R_1 are known values, we can get the estimated value of T_0 using (7) and get T_j using (4) and (5), and finally get the value of N_{t_j} through (2).

The congestion window is adjusted according to (8) after each window of data is received. It is similar to the schemes used in TCP Vegas and SC.

$$cwnd = \begin{cases} cwnd + 1 & if\ N_{t_i} < Q_m - \alpha \\ cwnd & if\ Q_m - \alpha < N_{t_i} < Q_m + \beta \\ cwnd - 1 & if\ N_{t_i} > Q_m + \beta \end{cases} \tag{8}$$

cwnd is the size of congestion window. Q_m is the expected value of PQL at the bottleneck router for this connection. Q_m - α and Q_m + β represent the lower and upper bound of the PQL at the bottleneck router.

4. Performance Test

In this section, we examine the performance of our new algorithm, TCP-CC, and compared its performance with TCP Reno [3], Vegas [11] and SC [12]. We first show the performance results for a basic configuration with a single connection case. Then, we show the performance results for multiple-connection case. Finally, we examine the influence of traffic on the reverse link. We have implemented TCP-CC, TCP Vegas and TCP-SC in our test-bed.

3.1 Configuration of Test-bed

We set up a test-bed as Fig.1 shows. The test-bed consists of 4 computers with Red Hat Linux 7.0 (kernel version: 2.4.10). The computers are connected with 100 Mbps Ethernet links. We have developed a per-flow router emulator with a DRR packet scheduling algorithm. It simulates the bottleneck router with 100 Mbps input link and 1.5 Mbps output link. The size of buffer is set to 22.5 KB (15 data packets) at the router. The last packet in the longest queue will be dropped when congestion happens. Another network emulator, NISTNet [21], is run on the computer labeled as the non-bottleneck router. It is used to simulate the delay at the non-bottleneck router on the Internet. We assume each packet will be delayed 50 ms at the non-bottleneck router. The bandwidth delay product (BDP) is equal to 19.8 KB, or 13.5 data packets. In the test, the receiver fetches a file with a size of 5 MB from sender through a FTP client.

3.2 Performance Results for One Connection Case

Fig.2(a) and (b) shows the growth of the TCP Reno's congestion window and packet queue at the bottleneck router. As TCP Reno uses packet loss as primary congestion indication, it keeps on increasing the size of congestion window until the congestion loss happens. Then, the congestion window is decreased and starts the next round of window increase-drop-decrease period. Even for the one-connection case, packets are dropped periodically, and cause the congestion window and packet queue at bottleneck router to vary in a see-saw oscillation. The delay variance caused

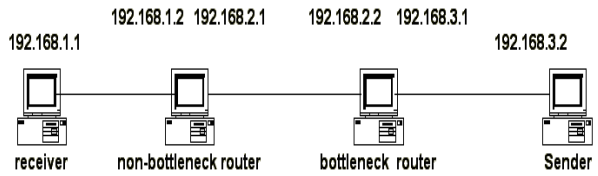

Fig.1: Configuration of Test-bed.

by this oscillation may affect the performance of real time and interactive applications [12].

Fig.3(a) and (b) shows the variation of the congestion window and the packet queue at the bottleneck router of our new algorithm. We set the value of α to 1.5 and β to 2.5. The congestion window varies in a very small range from 13 to 15, and the packet queue at bottleneck router varies from 1 to 3 packets. These figures show the main advantage of TCP-CC, that is, the bandwidth of the bottleneck router is almost fully used without causing any network congestion and packet loss, and the queue size at the bottleneck router varies within a controlled range. We have gotten similar results for TCP Vegas and SC in terms of the variation of congestion window and queue size at bottleneck router. Table 1 shows the throughput of TCP Reno, Vegas, SC and TCP-CC. TCP Reno, SC and TCP-CC have similar throughput that is better than TCP Vegas. Although congestion loss happens for Reno, the size of the congestion window is still greater than the value of BDP. For TCP-SC and TCP-CC, after the connection has started for a while, the pipe is kept full. TCP Vegas oscillates at the beginning of the connection and keeps the window size below the value of BDP for a certain duration. Therefore, the throughput is a little lower than that of the others.

We can also observe from Table 1 that 20 packets have been dropped due to the congestion for TCP Reno. For TCP Vegas, SC and TCP-CC, no packets were dropped since packet loss was not used as primary indication of congestion.

3.3 Performance Results for Multiple Connection Case

In the following, we study the performance of different algorithms in multiple connection cases. In the study, we start multiple connections one after another with an interval of 5 seconds. Each connection downloads 5 M bytes of data from server.

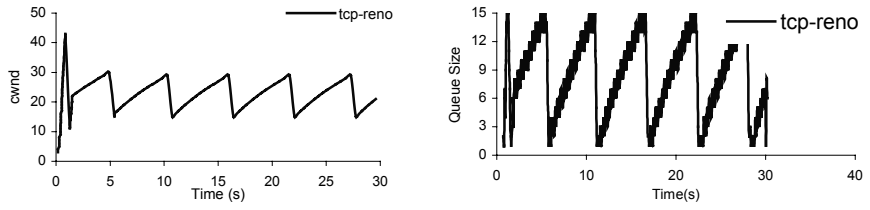

Fig. 2 TCP Reno (1 connection): (a) CWnd -(b) Bottleneck Queue.

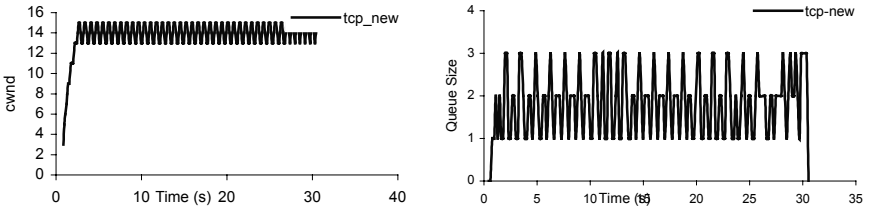

Fig. 3 TCP-CC (1 connection): (a) CWnd (b) Bottleneck Queue.

Table 1: Performance for TCP Reno, SC, Vegas and TCP-CC (1 connection).

Protocol	Throughput	Utilization (%)	Congestion loss
TCP Reno	1.40 Mbps	89.3	20 packets
TCP Vegas (2, 4)	1.36 Mbps	63.3	0
TCP-SC	1.40 Mbps	89.3	0
TCP-CC	1.40 Mbps	89.3	0

We name the connections with their sequence number. For example, we name the first connection as *conn1*, the second as *conn2*, etc.

Table 2 shows the performance results for different algorithms in the four-connection case. The throughputs of TCP Reno, Vegas and TCP-CC are the same as the buffers at the bottleneck router always have packets waiting to be sent. The throughput of TCP-SC is lower than that of the others. The reason is that, in the multiple connection case, the algorithm used by SC for queue size estimation is inaccurate. This further causes the congestion windows to be adjusted inappropriately. For example, as Fig.4(a) shows, for *conn1*, even when the congestion window is decreased to 2, the estimated queue size is still greater than 2. Thus, the *cwnd* is kept at 2 even when all other connections have completed their download and this causes the network to be underutilized. The reason is that SC computes the current PQL by summing the PQL of the previous window and the PQL introduced by the current window. However, with the change of network status, the old PQL has expired and cannot represent the corresponding value of PQL in the new network status, thus causing the inaccuracy in PQL estimation. Fig.4(b) shows the queue size of *conn1* of TCP SC. It oscillates between 0 and 1 for a while and then drop to 0 as the *cwnd* remained at 2.

TCP SC also has the fairness problem in multiple connection case. As Fig.4(a) shows, the values of *cwnd* for the connections are difference. Some are as high as 16 while some are as low as 2. The differences cause the bandwidth to be shared unfairly among the connections. Although connection 1 started first, it gets a bandwidth of 0.22 Mbps (1/7 of the total bandwidth) and is the last one to complete the download.

Compared to TCP-SC, our new algorithm does not have such problems. As Fig.5(a) shows, the values of *cwnd* for the connections are always close to each other. It is adjusted when the network status changes, for example, connections open or close, and tries to keep the PQL between α and β. Fig.5(b) shows the PQL for the *conn1*. Most of time, it varies in the range of 1 to 3, which is close to our objective. Other connections also have similar trends in PQL variation and the bandwidth is shared fairly among different connections.

Table 2: Performance for 4-connection case

Protocol	Throughput	Utilization (%)	Fairness
TCP Reno	1.43 Mbps	95.3	fair
TCP Vegas (2, 4)	1.43 Mbps	95.3	fair
TCP Santa Cruz	0.88 Mbps	89.3	unfair
TCP-CC	1.43 Mbps	95.3	fair

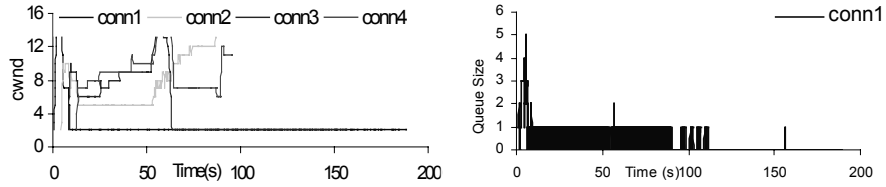

Fig. 4. TCP SC (4 connections): (a) CWnd (b) Queue Size of connection 1.

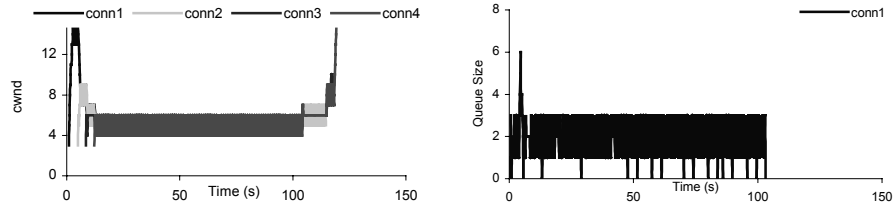

Fig. 5. TCP-CC (4 connections): (a) CWnd (b) Queue Size of connection 1.

Table 3 shows the packet drop rate for different algorithms (TCP-SC is not included as it has the fairness problem). TCP Reno drops more packets than other algorithms. Our TCP-CC is the best with no packet dropped in 4-connection case. When the number of connections increases to 6, the maximum number of packets expected to queue at the bottleneck router is 2.5 * 6 = 15 packets, which is equal to the maximal size of the available buffer at the bottleneck router. Thus, congestion loss happens when some connections do not adjust their window on time. However, the number of packets dropped by TCP-CC is less than one third of the packets dropped by Reno and Vegas. For an 8-connection case, the network is heavily congested. Our algorithm still performs better than Reno and Vegas with a 30% less in packet drop rate. This suggests that TCP-CC can obviously reduce the congestion loss and improve the performance of loss-sensitive applications.

4.4.1 Traffic on Reverse Link

Table 4 shows the performance results of different congestion control algorithms with traffic on the reverse link. The client first starts a connection that fetches data from the server, and subsequently starts another connection which sends data to the server on the reverse link.

Results in Table 4 shows that the performance of TCP Reno and TCP-CC is better than TCP Vegas with a percentage increase of 33.3% and 30.4% in throughput respectively. The reason is because the ACK delay on reverse link causes TCP Vegas

Table 3: Congestion Loss in Multiple-connection Case.

Protocol	4 connections	6 connections	8 connections
TCP Reno	350 (2.4%)	1092 (5.1%)	1963 (7.0%)
TCP Vegas (2, 4)	0 (0%)	1017 (4.8%)	1916 (6.6%)
TCP-CC	0 (0%)	315 (1.47%)	1286 (4.5%)

Table 4: Throughput Comparison with Traffic on Reverse Link.

Protocol	Throughput
TCP Reno	1.36 Mbps
TCP Vegas (2, 4)	1.02 Mbps
TCP-CC	1.33 Mbps

to incorrectly infer the increase of RTT as congestion on the forward link. Thus, *cwnd* is reduced and kept below BDP.

For both TCP Reno and TCP-CC, although there is slight performance degradation due to the delay of the ACK on the reverse link, the degradations are trivial (2.9% and 5%) with TCP Reno performing a little better than TCP-CC (2.3%). However, the cost of the better performance is an over-aggressive *cwnd*-increase algorithm that results in more congestion losses, as shown in Table 3. This violates the objective of congestion control — reducing the congestion loss.

5. Conclusion

In this paper, we have proposed a new congestion-control algorithm (TCP-CC) for the future Internet in which per-flow packet-scheduling algorithm is expected to replace the FCFS algorithm used by current routers. Compared to other congestion control algorithms, our new algorithm has the following advantages:

- Compared to TCP Reno, TCP-CC adjusts the congestion window based on the queue size of the connection at the bottleneck router instead of the packet loss and this reduces congestion loss significantly.
- Compared to TCP Vegas, TCP-CC achieves better throughput, less congestion loss and is more accurate in identifying the direction of congestion.
- Compared to TCP-SC, TCP-CC has sound fundamentals and solves the fairness problem in multiple connection scenarios.

Our future work is to study TCP-CC in a more complex network environment, which includes the characteristics of web traffic, the variations in packet size etc.

References

[1] V. Jacobson, Congestion avoidance and control, *Proc. SIGCOMM '88*, Stanford, CA, Aug 1988.

[2] V. Jacobson, Modified TCP congestion control avoidance algorithm, *Message to end2end-interest mailing list*, Apr 1990.

[3] W.R. Stevens, *TCP/IP illustrated, volume 1* (Addison-Wesley, 1996).

[4] H. Sawashima, Y. Hori, H. Sunahara and Y. Oie, Characteristics of UDP packet loss: effect of TCP traffic, *Proc. INET'97*, Jun 1997.

[5] S. Keshav and R. Sharma, Issues and trends in router design, *IEEE Communications Magazine*, May 1998, Vol.36, No.5, pp. 144-5.

[6] S. Shenker, Fundamental design issues for the Future Internet, *IEEE Journal of Selected Areas in Communication*, Vol. 13, No. 7, pp. 1176-1188, Sep 1995.

[7] L. Zhang, S. Berson, S. Herzog and S. Jamin, Resource reservation protocol, *RFC2205*, Sep 1997.
[8] A. Demers, S. Keshav and S. Shenker, Analysis and simulation of a fair queuing algorithm, *Proc. ACM SIGCOMM'89*, Vol. 19, pp. 1-12, Oct 1989.
[9] J. Nagle, On packet switches with infinite storage, *IEEE Transaction on Communications*, Vol. 35, Apr 1987.
[10] M. Shreedhar and George Varghese, Efficient fair queuing using deficit round-robin, *IEEE/ACM Transactions on Networking*, Vol. 4, No. 3, Jun 1996.
[11] L.S. Brakmo, S.W. O'Malley and L.L. Peterson, TCP Vegas: new techniques for congestion detection and avoidance, *Proc. ACM SIGCOMM'94*, pp. 24-25, Oct 1994.
[12] C. Parsa and L. Aceves, Improving TCP congestion control over Internets with heterogeneous transmission Media, *Proc. IEEE Conference on Network Protocols (ICNP '99),* Toronto, 1999.
[13] S. Flyod and V. Jacobson, Random early detection gateways for congestion avoidance, *IEEE/ACM Transactions on Networking*, Vol. 4, pp. 397-413, Aug 1993.
[14] V. Jacobson and S. Floyd, TCP and explicit congestion notification, *Computer Communication Review*, Vol. 24, pp. 8-23, Oct 1994.
[15] L. Kalampoukas, A. Varma and K. Ramakrihnan, Explicit window adaptation: a method to enhance TCP performance, *Proc. IEEE INFOCOM'98*, pp. 242-251, Apr 1998.
[16] S. Keshav, Congestion Control in Computer Networks *PhD Thesis, published as UC Berkeley TR-654* , September 1991.
[17] Z. Wang and J. Crowcroft, Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm, *Computer Communication Review*, Vol. 22, pp. 9-16, Apr 1992.
[18] Z. Wang and J. Crowcroft, A new congestion control scheme: slow start and search (Tri-S)", *Computer Communication Review*, Vol. 21, pp. 32-43, Jan 1991.
[19] G. Hasegawa and M. Murata, Survey on fairness issues in TCP congestion control mechanism, *IEICE Transactions on Communications*, E84-B6:1461-1472, Jun 2001.
[20] Cisco 12016 Gigabit Switch Router, available from http://www.cisco.com/warp/public/cc/cisco/mkt/core/12000/12016.
[21] NISTNet network emulator, available from http://snad.ncsl.nist.gov/itg/nistnet/.