

Evaluation of Proactive Congestion Control on RED Gateway

Kuoling Fang
Alcatel USA, Inc.
Plano, TX
klfang@hotmail.com

Hee Yong Youn and Hyunseung Choo
School of Electrical and Computer Engineering
Sungkyunkwan Univ., Suwon, Korea
{youn,choo}@ece.skku.ac.kr

Chansu Yu
School of Engineering
Info & Comm Univ., Taejeon, Korea
cyu@icu.ac.kr

Abstract

In this paper we investigate the network performance of TCP hosts and gateways employing New Reno TCP using a parallel simulator. The gateway adopts the random early detection (RED) scheme and proactive congestion control mechanism to enforce fair sharing among competing TCP connections. The test network is modeled using PARSEC. The simulation reveals that the RED gateway drops only a small number of packets, and utilization of the bottleneck link on the RED gateway is less than 75%. The number of retransmission timeouts recorded at the TCP senders is also small. Moreover, increasing the bandwidth of the gateway input links does not substantially improve the network performance, while increasing the number of input links is very effective.

Key words: congestion control, New Reno TCP, parallel simulation, PARSEC, random early detection.

1. Introduction

The Internet has experienced explosive growth in the past few years. In the presence of more hosts and users with fast link connections, the traffic load has shown dramatic increase. Therefore Internet users often encounter some congestion. The need for Internet congestion control became apparent during mid-80's, and network collapse due to congestion had already been predicted [1]. The random early detection (RED) scheme [2,3] was introduced as a measure of proactive congestion control for achieving low delay and high throughput. Study on the network performance of the TCP hosts and gateways employing the RED approach (shortly RED gateways) using a parallel simulator is the main objective of this paper.

A significant amount of today's Internet traffic including WWW (HTTP), file transfer, email, and remote access, are carried by the TCP [4]. Therefore, traffic dynamics in the Internet are heavily influenced by the behavior of the TCP. Here the congestion

control mechanisms are implemented at the hosts in the form of flow control. In 1988, Jacobson [5] pioneered the study of the TCP congestion control mechanisms; slow-start, congestion avoidance, conservation of packets, and exponential timer backoff. The TCP was later augmented with the fast retransmit and fast recovery algorithm to improve the efficiency [6,7,8]. However, there still exists a limit for the end-systems in controlling congestion. RED gateway was thus introduced to complement the end-system congestion control mechanisms [2,9].

The PARSEC (PARallel Simulation Environment for Complex system) [10] is a C-based language developed for parallel discrete event simulation based on the process-interaction approach. Here an object is represented by a logical process. Interactions among the processes are modeled by time-stamped message exchanges among the corresponding logical processes. It offers good support for high-level (application level) simulations with message passing infrastructure and abstraction. Therefore it is an efficient tool for simulating a system consisting of a number of concurrent processes such as network and operating system.

In this paper we study the dynamics between the TCP hosts and RED gateway using PARSEC. The simulation results reveal that the RED gateway drops only a small number of packets for reasonable traffic condition and network structure. Utilization of the bottleneck link is less than 75%, and the number of retransmission timeouts recorded at the TCP sender is small. It was also identified that increasing the bandwidth of the gateway input links does not improve the network performance, while increasing the number of input links is very effective.

The rest of the paper is organized as follows. Section 2 gives a brief introduction to the TCP with an emphasis on congestion control and the UCLA PARSEC language. Section 3 describes the modeling of the test network using PARSEC. Section 4 presents the simulation results, and Section 5 concludes the paper.

2. TCP Congestion Control and PARSEC

2.1 New Reno TCP

The TCP was designed to operate reliably over almost any transmission medium regardless of the transmission rate, delay, corruption, etc. The current TCP implementations adapt to the transfer rates in the range of 100 to 10^7 bps and round-trip delays of 1 ms to 100 seconds. The studies on the TCP performance have shown that it can work well over a variety of Internet paths, ranging from 300 bit/sec dial-up modems to 800 Mbit/sec I/O channels [11].

There are four intertwined congestion control algorithms in the TCP; *slow start*, *congestion avoidance*, *fast retransmit*, and *fast recovery*. The underlying mechanisms for these congestion control methods are congestion window and acknowledgement (ACK) [12]. Different versions of the TCP vary with the employed congestion control algorithms [4]. The version of TCP adopted in our simulation is New Reno TCP [13,14]. The features for congestion control employed in it are outlined below.

- *Slow-start and congestion avoidance.*
- *Fast retransmission* with the congestion window reduced to half and the slow-start threshold set to the new congestion window size.
- *Fast recovery* with the inflation of the congestion window by the number of duplicated ACKs until a new ACK arrives. Exit of fast recovery only upon arrival of the ACK for the highest number segment transmitted.
- Response to a partial ACK inferring that the segment has been lost.
- Delayed acknowledgment as one ACK for every two segments received.

2.2 RED Gateway

The traditional approach for managing the gateway queue length is to accept packets until the preset maximum length is reached, and then drop subsequent incoming packets until the queue length decreases. There are two important drawbacks in this approach. First, it allows a single connection or a few connections to monopolize the queue space, which prevents the rest of the connections from getting the queue space. Second, it causes the gateway queue to be full or nearly full for a long period of time, which reduces the chance for accommodating bursty traffics.

An active queue management mechanism can provide several advantages for Internet connections. First, it reduces the number of packets dropped in the gateway by keeping the average queue size small,

which allows to absorb bursty packets. It also provides lower-delay service.

Random early detection (RED) is an active queue management algorithm proposed for gateway that provides the Internet with the advantages cited above. In contrast to the traditional queue management algorithm, the RED algorithm drops arriving packets with a probability. The probability increases as the estimated average queue size grows. The RED algorithm consists of two main parts; estimation of the average queue size and the decision on dropping incoming packets or not.

The effectiveness of the RED algorithm relies on the ability to correctly detect the upcoming congestion condition. It manipulates three parameters; minimum threshold, maximum threshold, and maximum probability for dropping the packets. The static characteristics of the network structure and dynamics of the connections sharing the bandwidth influence the effects of these parameters. The static characteristics include the topology and bandwidth of the links. In this paper the effectiveness of the RED approach is investigated considering these factors.

2.3 PARSEC

A PARSEC program is a collection of C functions and entity definitions [10]. An entity definition (or an entity type) describes a class of objects. Instances of an entity type are created to model the objects in the target system. For example, if an entity type 'TCP Sender' is defined to model a TCP sender for a network simulator, five instances of it are created to model five TCP senders. Every PARSEC program has a driver entity. This entity initiates the execution of the simulation program. Obviously, it serves the same purpose as the function 'main' in C. In our network simulator, the driver entity is also used to configure the simulated network. The entities communicate with each other via buffered message passing, and executions of the entities are interleaved by the PARSEC scheduler.

In PARSEC, a unique message buffer is associated with each entity for buffered message passing. Asynchronous send and receive are provided to deposit and retrieve messages from the message buffer, respectively. The definition of primitive 'message' has a similar syntax as declaration of a structure in C. The parameters may be viewed as the fields defined within a structure and referenced using operators as in C.

The PARSEC system is a major upgrade from the last version of Maisie with several improvements both in the syntax of the language and execution environment. The PARSEC language and its runtime system take care of the details of the simulator so that

users can concentrate on modeling the objects and interactions among them in the target system. Here the modeled system is assumed to consist of a set of processes that interact with each other at discrete points of time. Each interaction among them is modeled by message exchange.

3. Modeling of Gateway

As mentioned earlier, a PARSEC program is a collection of entities. Communication between the entities is allowed via message passing. In our simulation there are seven types of entities – Driver(E_D), Send TCP(E_S), Receive TCP(E_R), End-system queue(E_EQ), Gateway queue(E_GQ), End-system IP link(E_EL), and Gateway IP link(E_GL).

The E_R acknowledges every two segments with a timeout approach. The E_S implements the New Reno TCP. It contains all the flow control features mentioned in the previous section. The E_S is implemented as a state machine with the seven states outlined below.

- *Idle*: This is the initial state of the E_S.
- *Slow-start(SS)* : After receiving the start message, it enters this state from the Idle state. The congestion window size increases by one for each received ACK. It can also be reached from the OOS and RTO state.
- *Congestion avoidance(CA)*: When the congestion window size becomes greater than the slow-start threshold, the TCP entity enters this state. The congestion window increases by one for each round trip time. It can also be reached from the OOS, FR, and PR state.
- *ACK out-of-sequence(OOS)*: After receiving the first duplicated ACK in both the SS and CA state, the TCP entity enters this state.
- *Fast recovery(FR)*: After receiving the third duplicated ACK from the OOS state, the TCP entity retransmits the missing segment and enters this state. In this state, congestion window inflation is implemented.
- *Partial recovery(PR)*: After receiving the first partial ACK from the FR state, the TCP entity enters this state. The congestion window is deflated for each partial ACK.
- *Retransmission timeout(RTO)*: After the retransmission timer expires in any state except the Idle state, it retransmits the missing segment and enters this state.

Figure 1 depicts the relationship between the states.

The E_EQ resides in a host. It stores the segments in the queue received from the E_S. It also accepts the requests from the associated E_EL and transfers the segments in its queue to the E_GL. The E_EL resides in the hosts. It accepts the segments sent from the E_GL it is connected to, and delivers the received segments to the TCP entities. It sends a request to the associated E_EQ for new data transfer as soon as it becomes idle.

The E_GQ resides in the gateway. It stores the segments in its queue forwarded from other E_GL. It accepts requests from the associated E_GL, and transfers a segment at a time to the E_GL. The RED algorithm is implemented in the E_GQ. Both the E_EQ and E_GQ operate based on the FIFO discipline. The E_GL resides in the gateway. It accepts the segments sent from the E_EL, and forwards the segments to an E_GQ according to the end-system's IP address. The IP link entity includes the IP and the lower layers.

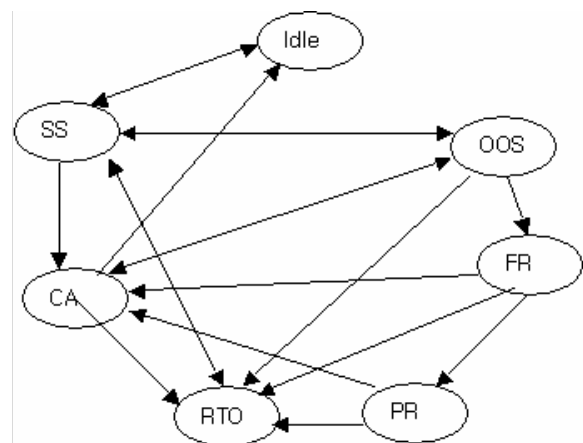


Figure 1. The state diagram for sending TCP entity.

The E_D assumes the same function as the function 'main' in a C program. The configuration of the network modeled is defined in it, and a number of entities of each type are created in the E_D. It starts an entity by sending it the parameter values and a set of enames (entity names). These enames are used by the entity to pass messages among the entities.

A host in a typical computer network consists of E_EL, E_EQ, E_S, and E_R. An example of a TCP sender is shown in Figure 2. The TCP entity is either E_S or E_R. Every IP link entity has a unique queue entity one-to-one associated to it. It is possible to have more than one IP link entity residing in one host if multiple output links are connected. It is also allowed to have multiple TCP entities to send data segments to the queue entity or to receive from the IP link entity as shown in the figure.

When an E_S has a data segment to send, it sends the segment to the E_EQ by specifying the receiving queue entity using its ename. Each message has a timestamp. The PARSEC runtime system stores messages in a queue by the order of the timestamps associated to them. The PARSEC scheduler checks available messages to activate an entity for execution. When a message moves to the head of the scheduler queue, the PARSEC runtime system delivers the message to the E_EQ and activates the entity into an active state. The queue entity stores the outgoing segments based on the FIFO discipline.

The interaction between the E_EL and E_EQ is modeled as request-reply messages. When the E_EL becomes idle, it sends a request message to the E_EQ asking it to send next segment. If there are data in the queue, the E_EQ sends the first segment of its queue to the E_EL. Otherwise, the reply is held until a new data comes in. Each E_EL entity can have at most one outstanding request message.

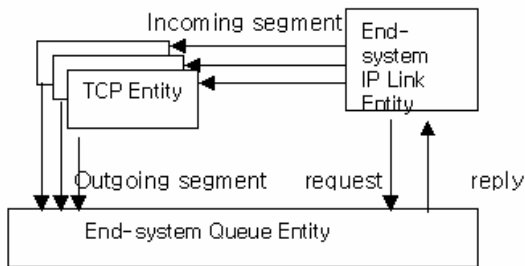


Figure 2. The structure of a TCP Sender.

The E_EL also has an array of enames of the TCP entities residing in the host and an ename of the corresponding E_GL. When the link entity receives an outgoing segment from the queue entity, it sends the segment to the corresponding E_GL. It also receives segments from the E_GL. After receiving a segment from the E_GL, it extracts the ename of the intended TCP entity and sends the segment to that entity.

A gateway consists of a set of gateway link and queue entity pairs. An example is shown in Figure 3. The E_D creates a set of entity pairs and starts these entities by sending a set of parameters and enames to them. Each Gateway IP link entity receives an array of enames of gateway queue entities corresponding to the terminating host IP addresses.

The E_GL-A in Figure 3 interacts with the associated E_GQ-A with the same request-reply messages as that on the host. When it receives a segment from its corresponding end-system, it forwards the segment to the E_GQ-B. The segment is then transferred to E_GL-B, which is connected to

other target end-system.

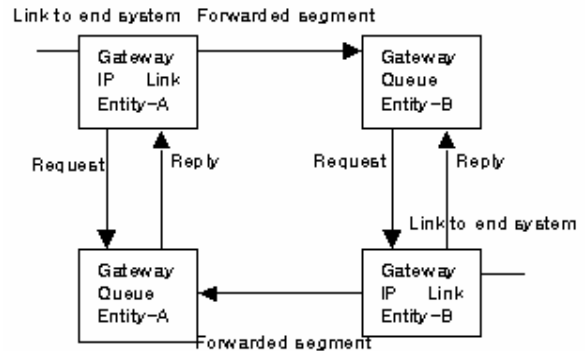


Figure 3. The structure of a gateway with two links.

4. Experiments

4.1 The Test Network

The network tested in this paper consists of five senders, one receiver, and one RED gateway as shown in Figure 4. This is a simple network, but sufficient for understanding the interrelationship among the network parameters. It has been thus commonly used by others [9].

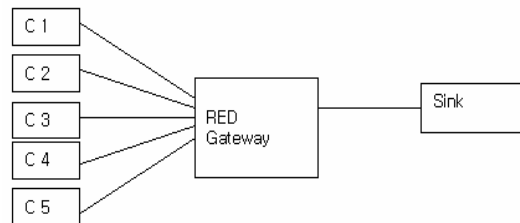


Figure 4. The test network.

The five TCP senders, C1-C5, in the figure are assumed to always have data to send and the congestion window can grow up to 128 segments. For the RED Gateway, the buffer size is 16 packets, the minimum threshold (min_th) is 4 packets, the maximum threshold (max_th) is 8 packets, the queue weight (Wq) is 0.002, and MaxP is 0.02. Here MaxP is the maximum probability of a packet to be dropped when the queue length (avg) is between min_th and max_th. The Maxp value of 0.02 is unrealistically low, but chosen for the purpose of comparison.

In the RED gateway algorithm [2], the current packet-marking probability (Pb) is calculated for each incoming packet using the formula, $MaxP \frac{avg - min_th}{max_th - min_th}$. Here avg is the exponentially weighted moving average of the queue length, which is obtained as $(1 - Wq) avg + Wq Q$. Q is the current size of the gateway queue. There exist three static factors related to network structure which

might cause network congestion. They are network topology, bandwidth-delay product, and speed mismatch.

- Network topology: TCP senders, C1 - C5, simultaneously send data through the RED gateway to overload the outgoing link.
- Bandwidth-delay product: A large value of this implies congestion. In order to study this, three different cases of link bandwidths - B1, B2, and B3 - are considered. In the B1 case, the TCP senders C1-C4 are connected to the RED gateway through the links of the bandwidth of 103.3 Mbps and transmission delay of 1 ms. Sender C5 has a link of the bandwidth of 31.0 Mbps and transmission delay of 16 ms. The bandwidths of all the links are doubled and quadrupled in the B2 and B3 case, respectively.
- Speed mismatch: The outgoing link from the gateway to Sink has the bandwidth of 51.7 Mbps and transmission delay of 2 ms. With this specification of the outgoing link, if data are sent at the peak rate, the buffer space of the RED gateway will be filled up quickly and then packets will be lost.

4.2. Simulation Results

Each of the three cases are run ten times and the numbers are added. Each simulation run time is set to 30 seconds. Table I summarizes the results on the total number of retransmission timeout.

Table I. The total number of retransmission timeouts.

Case	C1	C2	C3	C4	C5	Total
B1	0	0	0	0	0	0
B2	1	3	3	2	0	8
B3	3	3	2	31	0	39

Notice from the table that the total numbers of retransmission timeouts are surprisingly low. The B1 case experiences no timeout for all the senders. The B2 case of doubled bandwidth has a small number of timeouts. For the B3 case of quadrupled bandwidth of the B1 case, it becomes significantly larger than the other cases mainly due to C4. Otherwise, the rest of the TCP connections have a small number of retransmission timeouts. The total number of packets (in thousands) dropped in the RED gateway are presented in Table II. The number for the B2 case is almost twice that of the B1 case. There is no significant difference between the B2 and B3 case.

The total number of packets received at the receiving TCP entities, referred as goodput, are listed in Table III. The B1 case allows more goodput than

the other two cases as expected. However, the goodput of the B3 case is slightly higher than that of the B2 case. This is not an expected result. An explanation might be that the higher bandwidth of the B3 case does not cause congestion serious enough to lower the goodput but contribute to increase it by allowing more traffic. The table also lists usage percentage which is the ratio of the total number of segments received by all the TCP entities to the highest possible number of packets that the bottleneck link (the link between the Gateway and Sink) can transmit. For example, refer to the B1 case of Table III. Assume that a segment is 512 bytes with TCP data only, TCP header is 20 bytes, and IP header is 40 bytes. Therefore, the size of a packet is 572 bytes. Since the bandwidth of the bottleneck link is 51.7Mbps (6.4625MBps), maximally $6.4625 \times 10^6 / 572 = 11298.1$ packets can be sent per second. Finally the Usage % is obtained by $1856000 / (300 \times 11298) = 0.55$. It can be said that the available bandwidth is underutilized.

Table II. The total number of packets dropped.

Case	C1	C2	C3	C4	C5	Total
B1	59	122	131	126	9	447
B2	260	145	149	141	132	827
B3	268	139	157	154	155	873

Table III. The total number of segments received.

Case	C1	C2	C3	C4	C5	Usage
B1	651	230	258	239	478	0.55
B2	582	305	267	240	153	0.46
B3	573	333	334	318	124	0.50

In the study above the performance of the test network was checked while the link bandwidths are varied. The simulation data show that little improvement is achieved when the link bandwidths are increased. We next study what happens to the network performance if the number of links is increased instead of increasing the bandwidths. Interestingly, the performance shows a lot of improvement.

To see the effect of increased number of links, the number of the end systems is doubled while the bandwidth of each link remains the same. The test network configuration is thus changed into doubled number of input links to the RED gateway since the usage percentage was identified to be low from the previous experiment. In the new test network, there are eight input links of the bandwidth of 103.3 Mbps and transmission delay of 1 ms, and two links of the

bandwidth of 31.0 Mbps and transmission delay of 16 ms. The packet drop ratio can be as high as 10% for a TCP connection. The ratio of 1/50 used in the previous experiment is regarded unrealistic. In earlier simulators the default value for this ratio is set to 0.1. The doubled number of links will increase the bottleneck link usage percentage, and thus the impact of the maximum drop probability can be identified.

Table IV. The performances for different MaxP.

MaxP	0.1	0.08	0.06	0.04	0.02
Goodput	1267	1234	1089	1198	1217
Usage%	0.75	0.73	0.64	0.71	0.72
Drops	359	341	362	365	316
RTO	18	21	44	17	9

Table IV reveals the effect of different values of maximum drop probability(MaxP) on the performance metrics studied. Comparing the total number of packets received and the usage percentages, the highest and lowest MaxP of 0.1 and 0.02 display slightly better performance than the middle MaxP values. We observe similar results for other measures, too. The total number of packets dropped show no significant differences with varying MaxP. Intuitively the MaxP affects the total number of packets dropped, while the results show no straightforward relationship between them.

5. Conclusion

In this paper the dynamics of a test network of hosts connected through RED gateway has been studied using PARSEC. The simulation results show that the maximum usage percentages of the bottleneck link from the RED gateway to the Sink node is about 75%. Even though the dropped packets are included in the counting, the usage percentages are still below 0.75. It is obvious that there still exist some room to improve the TCP/IP performance. The next question is how high could the usage percentage be. In some test runs, the usage percentages becomes as high as 0.93. Besides, the RED gateway congestion control algorithms are able to keep the queue short. It seems that, with the deployment of the RED gateway, the flow control algorithms of the TCP can be more aggressive to increase the usage percentages as long as the algorithms are able to conform to the TCP requirements and standards.

The test results also show that the number of input links affects the usage percentages of the bottleneck link. Doubling the bandwidths of the input links does not significantly improve the usage

percentage. Some cases even show that the usage percentage decreases. One of the reasons for this is that there is only one user per each input link. These phenomena are similar to the effects of global synchronization in the real world. Each input link represents a significant percentage of traffic loads. The usage percentages go up as the number of input links increases. More investigation is needed to model the relationship between the network parameters and performance.

References

- [1] J. Nagle, "Congestion Control in IP/TCP," RFC 896, Jan. 1988.
- [2] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *Trans. on Networking*, Vol .1, No.4, pp. 397-413, Aug. 1993.
- [3] M. Gaynor, "Proactive Packet Dropping Methods for TCP Gateways," Oct. 1996, URL <http://www.eecs.harvard.edu/~gaynor/final.ps>.
- [4] J. Postel, "Transmission Control Protocol-DARPA Internet Program Protocol Specification," RFC 793, DARPA, Sept. 1981.
- [5] V. Jacobson, "Congestion Avoidance and Control," *Computer Communication Review*, Vol. 18, no. 4, pp. 314-329, Aug. 1988.
- [6] J. Hoe, "Startup Dynamic of TCP's Congestion Control and Avoidance Schemes," Master's Thesis, MIT, 1995. "<http://ana-www.ics.mit.edu/anaweb/ps-papers/hoel-thesis.ps>"
- [7] R. Morris, "TCP Behavior with Many Flows," *IEEE International Conference on Network Protocols*, Oct. 1997.
- [8] G.R. Wright and W.R. Stevens, "TCP/IP Illustrated, Volume 2: The Implementation," Addison-Wesley, 1995.
- [9] D. Lin and H.T. Kung, "TCP Fast Recovery Strategies: Analysis and Improvements," *Proc. IEEE INFOCOM '98*, pp. 263-271, March 1998.
- [10] R.A. Meyer, "PARSEC User Manual," Release 1.1, Aug. 1998, <http://pcl.cs.ucla.edu/projects/parsec>
- [11] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC 1323, May 1992.
- [12] S. Floyd, "Issues of TCP with SACK," ftp://ftp.ee.lbl.gov/papers/issues_sa.ps.Z, Jan. 1996.
- [13] B. Braden, ed., "Requirements for Internet Hosts Communication Layers," RFC 1122, Oct. 1989.
- [14] L.S. Brakmo, S.W. O'Malley, L.L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *Proc. of ACM SIGCOMM '94*, pp. 24-35, Aug. 1994.