Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications Las Vegas, Nevada, June 1999, pp. 2143-2149

Behavior of Competing TCP Connections on a Packet-Switched Ring: A Study Using the Harvard TCP/IP Network Simulator

H.T. Kung and S. Y. Wang
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138, USA

Abstract

This paper presents simulation results on the behavior of TCP connections when they compete on a packet-switched ring. These results were generated by the Harvard TCP/IP network simulator, which uses real-life TCP code to support high-fidelity simulation. There are two parts to the paper. The first part is a high-level description of the methodology for constructing the Harvard TCP/IP network simulator. The second part describes an application of the simulator in studying the behavior of competing TCP connections on a packet-switched ring. Simulation results show that TCP connections achieve various degrees of fairness under different packet scheduling methods used in the nodes of the ring. Under the FIFO scheduling, the connections exhibit serious unfairness. Under per-source queueing, per-source-destination-pair queueing, and 2-level queueing, the connections exhibit improved levels of fairness.

Keywords*: Network simulation, TCP/IP, performance evaluation, ring network*

1 Introduction

Network simulators implemented in software are flexible tools for researchers to develop, test and diagnose a variety of network protocols. It is well-known, however, that network simulators also have limitations. Developing a complete network simulator, including associated application programs and network tools, is a large effort. Due to limited development resources, typical network simulators have the following drawbacks:

¥ Simulation results are usually not as convincing as those produced by real hardware and software equipment.

¥ These simulators are not extensible. They lack the application programming interface (API) that allows application programs to be developed and run on nodes in a simulated network.

This paper proposes a simple simulation methodology for alleviating these drawbacks. It simulates a packetÕs traversing multiple network nodes by letting it re-enter the UNIX kernel of the simulation host multiple times. Based on this methodology, we have developed the Harvard TCP/IP network simulator [1] that simulates a network of BSD UNIX hosts and routers. This network simulator has two good properties: 1) It makes direct use of the real-life BSD UNIX TCP/IP protocol code, existing network application programs, and existing network tools. Thus, while being able to use existing software, it is a high fidelity simulator in the sense that it generates more accurate simulation results than a traditional TCP/IP network simulator that may abstract a lot away from real-life TCP/IP implementations. 2) It provides the BSD UNIX systemÕs API (i.e., the standard UNIX system call interface) on every node in a simulated network and allows application programs to be developed and run on any node in a simulated network. Since a developed application program is a real UNIX program, our simulator has an important advantage that a programÕs simulation implementation can also be its real implementation on a UNIX machine. Thus, when the simulation study is finished, we can quickly implement the real system by reusing the simulation implementations.

We used the Harvard TCP/IP network simulator to study the behavior of competing TCP connections on a packet-switched ring. We chose to study the ring topology because it is widely used by SONET rings as transport networks. A packet-switched ring is composed of layer-3 routers as ring nodes. We studied the behavior of TCP connections when they compete with each other on a packet-switched ring for available bandwidth under different traffic configuration patterns and packet scheduling methods. In this paper, we report on observed performance of these connections, and performance effects of using different packet scheduling algorithms.

Sections 2 and 3 present the methodology for constructing the Harvard TCP/IP network simulator. Section 4 reports on the observed behavior of competing

TCP connections on a packet-switched ring. Section 5 concludes the paper.

2 Simulator Architecture Overview

The architecture of the Harvard TCP/IP network simulator differs from traditional ones in the way they integrate the various component programs that implement the following functions:

1. Links with various delays and bandwidths
2. Routers that forward IP packets
3. Hosts that use TCP/IP protocol to send and receive packets
4. Application programs that generate network traffic

Unlike traditional approaches such as ns [2], our simulator architecture does not combine these four parts together to form a single monolithic program. Instead, it has separate and independent parts for 1, 2, 3 and 4. When these parts run concurrently in a BSD UNIX environment, together their executions simulate a network and generate network traffic.

In the rest of this section, we describe the key ideas and techniques that have made our simulator architecture feasible.

2.1 Simulating Single-Hop Networks

Tunnel network interface, available in most BSD UNIX environments, is a pseudo network interface that does not have a real physical network attached to it. The functions of a tunnel network interface, from the kernelÕs point of view, are no different from those of a normal Ethernet or FDDI network interface. A network application program can send out its packets to its destination host through a tunnel network interface or receive packets from a tunnel network interface, just as if these packets were sent to or received from a normal Ethernet interface.

As depicted in Figure 1, we can simulate a network configuration in which two hosts are connected together by two one-way links of any bandwidth, delay and other characteristics (e.g., packet corruption, packet dropping, reordering, and duplication). This can be done simply by writing an application program (we call it Òvirtual linkÓ object) that plays the role of a link. This application would open a tunnel network interfaceÕs special file in /dev and then execute a loop in which it reads a packet from the special file, waits the linkÕs propagation delay time plus the packetÕs transmission time on the link, and then writes this packet to the special file. The application continues this loop forever.

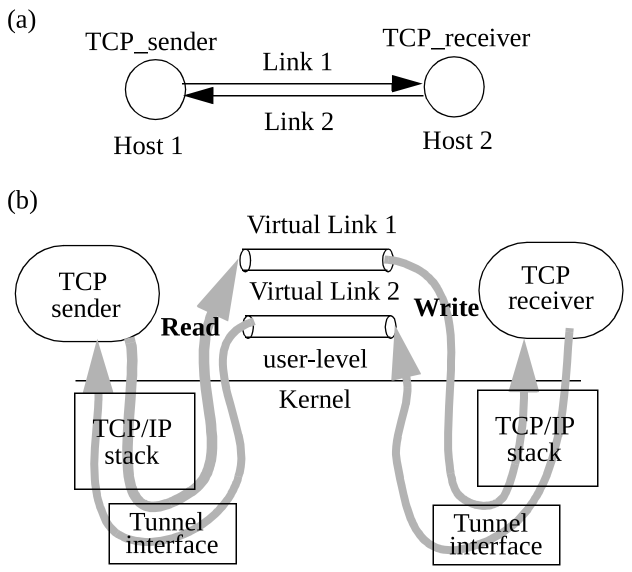

Figure 1: A network (a) is simulated using tunnel network interface (b).

2.2 Simulating Multi-Hop Networks

The single simulation host will act as both hosts and routers when simulating a multi-hop network. Using the network of Figure 2 (a) as an example, we illustrate the operation of the simulation host when sending a packet across the network from node 1 to node 2. As depicted in Figure 3, it involves a sequence of leaving and entering the kernel operations. That is, to forward a packet along the path from the TCP_sender to the TCP_receiver in Figure 2 (a), the packet will leave the kernel along link 1 and then re-enter it, leave the kernel along link 3 and then re-enter it, and finally, leave the kernel along link 5 and then re-enter it.

For routing packets, the simulation host needs to maintain a routing table. Since the simulation host simulates all the routers and hosts in a simulated network, using the union of the routing tables in all the nodes it should be able to route packets correctly. However, together these routing tables contain conflicting information. For example, in the network of Figure 2 (a), when forwarding a packet destined to host 2, host 1 will choose link 1 as its next hop, router 1 will choose link 3, and router 2 will choose link 5. If all these conflicting pairs (destination IP address, next hop) are stored in the single simulation hostÕs routing table, the kernel will be confused and will not be able to choose the correct next hop for forwarding a packet.

Our solution is to use special address-remapping and route-setup schemes. The basic idea is that, by remapping the destination IP address of a packet before it arrives at and

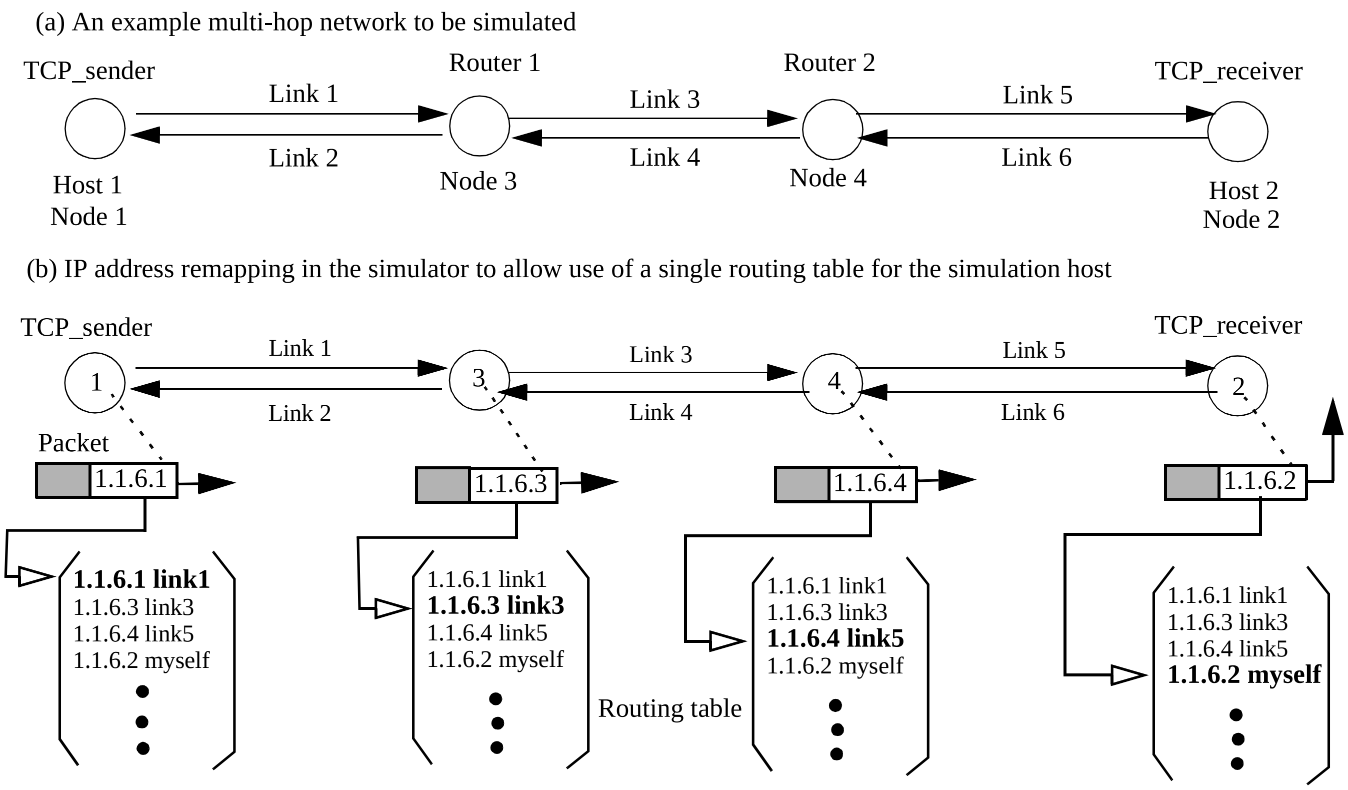

Figure 2: Simulation of a multi-hop network, and IP address remapping. The original destination IP address in the packet destined to node 2 is 1.1.6.2.

is forwarded by a router, a single routing table can be used in the simulation host without the conflicting problem. Because traffic may be bidirectional (e.g., TCP traffic), address-remapping needs to be applied to both the source and destination addresses of a packet. We call the mapped version of an IP address on node i as this IP addressÕs **ÒAs-Seen-By-Node(i)Ó** address. Since before a packet arrives at a router it is transmitted along a link, it is natural that in our simulator the virtual link objects implement the address remapping. Figure 2 (b) illustrates the address-remapping idea.

3 Required Supports from Kernel and Applications

3.1 Kernel Modifications

Disabling IP and UDP/TCP Checksum Tests. Because in simulation the source and destination IP addresses of a packet will change every hop (see Section 2.2). the checksums in the IP and UDP/TCP headers of the packet are incorrect, and should not be checked.

Each Node Has Its Own Virtual Clock. In the real world, each machineÕs clock may be different from othersÕ due to clock drift. We simulate this phenomenon to avoid multiple TCP connections to drop packets, time out, and Òslow-startÓ in a lock-step manner. In our simulator, we maintain a virtual clock for each node in the simulated network, and TCP slow and fast timers are triggered based on the virtual time of the simulated network rather than the real time.

3.2 Application Modifications

Associate the ApplicationÕs TCP Sockets with the ID of the Node Where the Application Will Run. The TCP socketÕs retransmit timer should be triggered based on the virtual clock of the node on which the application is running.

Convert the Destination Address to the ÒAs-Seen-By-Node(i)Ó Address. The application normally would need to convert its packetsÕ addresses before they are delivered to and routed by the kernel. The conversion can be avoided if the destination address provided to the application program is already the ÒAs-Seen-By-Node(i)Ó address of a packetÕs destination address (assuming that the application is running on node i).

Use Simulated NetworkÕs Virtual Time. When an application program reports data related to time, it should use the simulated networkÕs virtual time, rather than the real

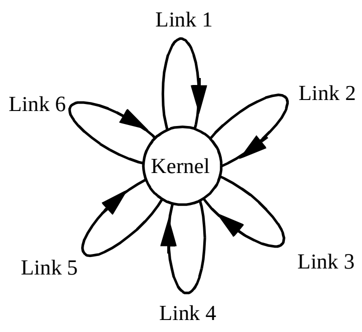

Figure 3: Routing a packet along a route in the simulated network of Figure 2 (a) is a sequence of leaving and entering the kernel operations.

time. Examples include Òping,Ó which reports a packetÕs round-trip time, and Òftp,Ó which reports the throughput of a file transfer.

3.3 SimulatorÕs Event Scheduler

Our system has only one object type to simulate, i.e., virtual link objects that simulate links. Hosts and routers are ÒsimulatedÓ or ÒrunÓ by the BSD UNIX kernel of the simulation host. Therefore, we do not need to create corresponding objects for them in the simulation system. The simulator is a discrete-event dynamic system. The unit of its virtual time can be set to any value as small as we would like (e.g., nanosecond) to simulate high speed links. All events can thus be precisely scheduled and triggered based on the virtual time in the simulated network. For this reason, simulation results are not affected by other activities on the simulation host (e.g., disk I/O and network I/O).

4 Behavior of Competing TCP Connections on a Packet-Switched Ring

Taking advantage of its high fidelity property, we have used the Harvard TCP/IP network simulator to perform detailed study of packet scheduling methods on a packet-switched ring. In particular, we study their effects on the behavior of competing TCP connections. The scheduling methods we have considered include FIFO, per-source queueing (PSQ), per-source-destination-pair queueing (PSPDQ), and two-level queueing (2LQ).

Under a separate effort, we have also constructed in our lab testbed networks to study TCP behavior on a ring. Our experimental results obtained from the testbed networks are consistent with the simulations result reported in this paper.

4.1 Simulation Configuration

The simulated topology is a 16-node uni- or bi-directional ring. For presentation clarity, only an 8-node ring is shown in Figures 4 and 5. The simulated traffic patterns include one-to-many, many-to-one and many-to-many as depicted in Figure 4. The buffer size in each node is 100 packets. The link between any pair of adjacent ring nodes is 100 Mbps, and has a delay of 5 ms. The performance metrics which we focus on are the link utilization on the ring and the fairness of achieved bandwidths of competing TCP connections on the ring.

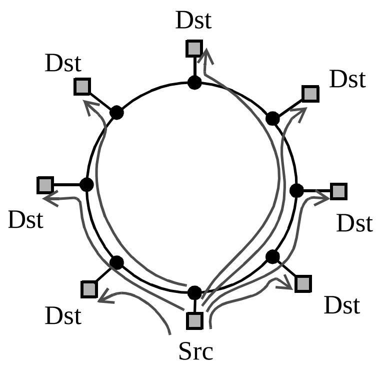

Figure 4: One-to-many traffic pattern on a bi-directional ring. An arrow represents a greedy TCP connection. When these arrows are reversed, the traffic pattern becomes many-to-one. Many-to-many traffic pattern is obtained by having one TCP connection for any pair of two nodes.

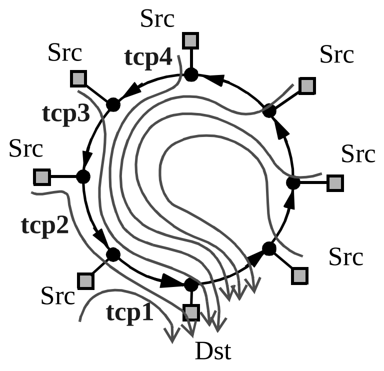

Figure 5: A many-to-one traffic pattern on a uni-directional ring.

4.2 FIFO Scheduling

We report on two cases when FIFO scheduling is used at the ring nodes. The first case concerns a many-to-many traffic pattern on a bi-directional ring. For a ring of N nodes, the many-to-many pattern is the union of N copies of the one-to-many pattern of Figure 4, each with a different source node. The second case concerns a many-to-one traffic pattern on a uni-directional ring as depicted in Figure 5.

The overall findings about FIFO scheduling are that a high link utilization can be achieved by having a sufficiently large buffer at each ring node, and sufficiently many TCP connections on the ring. However, fairness can be a problem. For example, the ratio of the maximum achieved bandwidth to the minimum achieved bandwidth can be as high as 10 among competing TCP connections.

Case 1:

As shown in Figure 6, the more congested nodes a TCP connection needs to traverse, the less bandwidth it can achieve. Note that since the traffic pattern on the ring is many-to-many and each TCP connection is greedy, each node on the ring is fully loaded and is a bottleneck router. The simulation shows that each node on the ring has almost the same packet drop rate. The reason for the reported uneven bandwidth distribution of Figure 6 is that, the more bottleneck nodes a TCP connectionÕs packets need to go through, the higher packet drop rate they will experience before they arrive at their destination node. As a result, TCPÕs fast retransmit mechanism or time-out will be triggered more often, which greatly reduces the maximum bandwidth a TCP connection can achieve.

Although the performance results of Figure 6 may also suggest that the longer a TCP connection is (or the larger a TCP connectionÕs round-trip time is), the less bandwidth it can achieve, it turns out that this is not the main reason for the unfairness phenomenon. We have confirmed this by simulation. That is, when we purposely set the delays of the links on the ring to a negligible value so that each TCP connectionÕs RTT is almost the same, the unfairness phenomenon still exists.

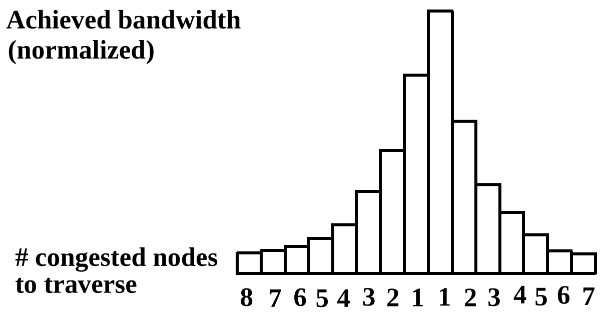

Figure 6: Normalized achieved bandwidths for case 1 of FIFO scheduling. Each bar represents the achieved bandwidth of a TCP connection.

Case 2:

As shown in Figure 7, a TCP connection which needs to compete with a larger group of other TCP connections achieves less bandwidth. This phenomenon is caused by a property which we call the Òsmall-large unfairnessÓ of TCP traffic. In the rest of this section, we will first discuss the Òsmall-large unfairnessÓ in detail, and then explain why it causes unfairness problems for the ring network.

4.2.1 ÒSmall-Large UnfairnessÓ of TCP Traffic

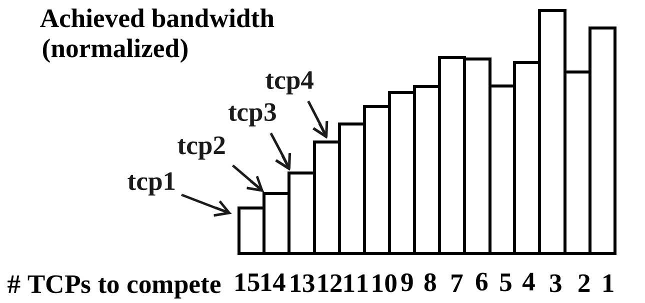

Figure 7: Normalized achieved bandwidths for case 2 of FIFO scheduling. Each bar represents the achieved bandwidth of a TCP connection.

The Òsmall-large unfairnessÓ of TCP traffic refers to the fact that a small group of TCP connections is ÒdominatedÓ by a large group of TCP connections when contending for shared bandwidth. This means that the TCP connections in the small group tend to time-out often and, consequently, the small group cannot achieve its fair share of the bandwidth.

Depicted in Figure 8 is a simulation study which demonstrates the Òsmall-large unfairnessÓ property. The router has two input ports and one output port. A group of C1 TCP connections enters the router from the top input port and another group of C2 TCP connections enters the router from the bottom input port. In total, C1 + C2 TCP connections together contend for the output portÕs link bandwidth and its buffer space. While fixing C1 + C2 to be 40, we varied the ratio of C1 to C2 and measured the resulting ratio of the achieved aggregate bandwidth of the top group to that of the bottom group. The simulation result shows that when 3C1 = C2 = 30, the top group only achieves an aggregate bandwidth of 9% and the bottom group achieves 91%. In contrast, however, the ideal bandwidths for the top and the bottom group should be 25% and 75%, respectively, as TCP generally can achieve per-flow level fairness. These results show that the small group of TCP connections are dominated by the large group of TCP connections, and cannot achieve its fair share of the bandwidth. The same unfairness problem happens when we reversed the ratio of C1 to C2 from 1:3 to 3:1.

We can explain the reason for the Òsmall-large unfairnessÓ of TCP traffic as follows: when the output portÕs buffer is full and when there are two incoming packets entering the router at the same time to contend for the last buffer slot, one of them will be randomly selected to be dropped. Hence the number of dropped packets arriving from the top input port is roughly the same as that arriving from the bottom input port. Note that, in a large group of TCP connections, the number of TCP connections which happen to escape packet dropping is higher than that in a

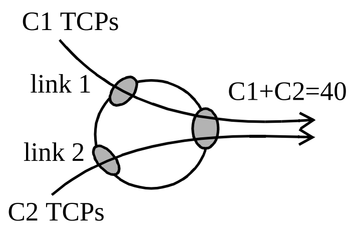

Figure 8: A small group of TCP connections is dominated by a large group of TCP connections. The percentages in the table represent the link utilizations.

	Link1	Link2
3C1=C2=30	9%	91%
C1=C2=20	50%	50%
C1=3C2=30	91%	9%

small group of TCP connections. The large group thus will have more TCP connections ÒaliveÓ than the small group after the competition. These live TCP connections will keep using the bottleneck link, while the other TCP connections are timed out or slowed down.

4.2.2 ÒSmall-Large UnfairnessÓ on the Ring

The Òsmall-large unfairnessÓ property shown in Figure 8 can readily explain the uneven bandwidths achieved by the competing TCP connections on the ring, that are reported in Figure 7.

Note that from Figure 5, each node on the ring, except the one to the right of Dst, has exactly the same input-port and output-port structure as that shown in Figure 8. That is, they all have two input ports -- one from the upstream node and the other from the host, and they all have one output port -- the one going to the downstream node. Figure 9 shows this structure, and the measured packet drop rates associated with links and TCP connections.

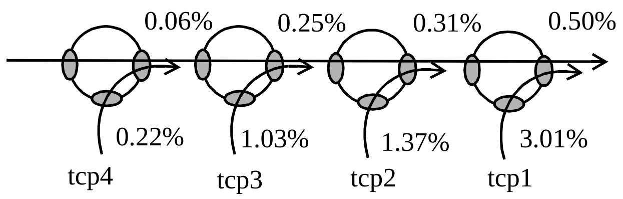

Figure 9: A uni-directional ring is composed of a series of routers shown in Figure 8. The numbers at the top are the measured packet drop rates of the associated links on the ring. The numbers at the bottom are the measured packet drop rates of the associated TCP connections.

The fact that tcp1 in Figure 7 achieves the least bandwidth can be explained by the fact that it experiences the highest packet drop rate (3.01%) among all TCP connections on the ring. The reason why it experiences the highest packet drop rate in turn can be explained by the Òsmall-large unfairnessÓ property. Observe that tcp1 needs to compete with a group of 15 other TCP connections coming from the other input port. Tcp2 achieves the second least bandwidth because it needs to compete with a group of 14 other TCP connections. The numbers of other TCP connections to compete for tcp3, tcp4, ... , are 13, 12, ... , respectively. Thus, the Òsmall-large unfairnessÓ property explains why the achieved bandwidths for tcp1, tcp2, tcp3, etc. decrease.

4.3 Per-Source-Destination-Pair Queueing (PSDPQ)

To solve the unfairness problem of Figures 6 and 7, we can use per-source-destination-pair queueing (PSDPQ). That is, every node on the ring has a dedicated queue for each pair of source and destination nodes and schedules packets for transmission using the round-robin method over the queues. The overall finding about PSDPQ is that the utilization of links on the ring can be high, and per-source-destination-pair (PSDP) fairness can be achieved.

However, the resulting PSDP fairness may not be appropriate for transport networks which require per-source node fairness. (Consider, e.g., the situation when each source node pays the same subscription fee to an ISP for the same bandwidth.) Figure 10 shows a scenario under PSDPQ in which source1 achieves four times more bandwidth than source2, just because source1 happens to have four PSDPs on the bottleneck link while source2 has only one.

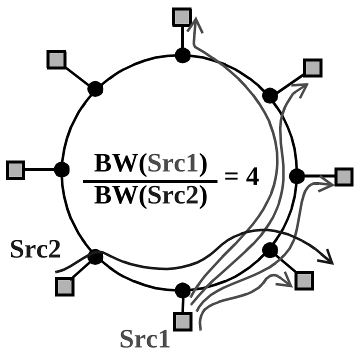

Figure 10: PSDP queueing and round-robin scheduling result in PSDP fairness. However, in this scenario, source1 achieves fours times more bandwidth than source2. This may not be appropriate for transport networks that require per source fairness.

4.4 Per-Source Queueing (PSQ)

The overall finding about per-source queueing (PSQ) on the ring is that (1) the utilization of links on the ring can be high, and (2) it can achieve per-source fairness (not per-flow or per-source-destination-pair fairness). Also, it does not exhibit the unfairness problem of FIFO scheduling.

When an outgoing link from the ring to a host is a bottleneck, we need to implement PSQ on the outgoing link as well. Otherwise the FIFO scheduling used on the outgoing link will play an undesirable role of allocating bandwidth. The results in Figure 11 show the difference between using and not using PSQ on the outgoing link when it is a bottleneck link.

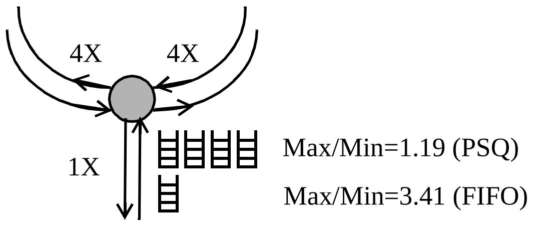

Figure 11: When the speed (4X in this example) of links on the ring is higher than the speed of outgoing links (1X) from the ring, PSQ on the outgoing link reduces unfairness.

Note that while PSQ can achieve per-source fairness, it cannot control the bandwidth allocation among flows which originate from the same source node but are destined to different destination nodes. Figure 12 shows a scenario exhibiting this unfairness problem.

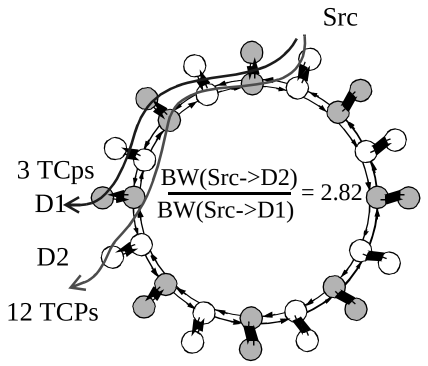

Figure 12: Under PSQ, a destination with a larger number of terminating flows originated from the same source achieves a larger bandwidth. The background traffic is the many-to-many load as defined in the caption of Figure 4.

4.5 Two-Level Queueing (2LQ)

In two-level queueing (2LQ), per-destination queueing is implemented on top of per-source queueing at each ring node. This avoids the PSDP problem described in the end of Section 4.3, and also solves PSQÕs problem that it may not be fair between destinations with different numbers of terminating flows. Figure 13 shows that under 2LQ this PSQ problem is basically solved. The utilization of links on the ring are generally high when 2LQ is used.

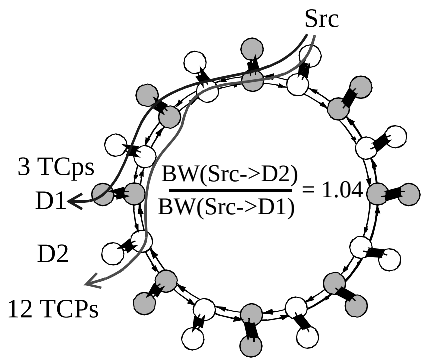

Figure 13: When 2LQ is used on the ring, it can fairly allocate a source nodeÕs achieved bandwidth among flows destined to different destination nodes regardless of their numbers of connections.

5 Conclusions

In this paper, we first present the methodology used to develop the Harvard TCP/IP network simulator. The Harvard TCP/IP network simulator is both extensible and high-fidelity [1]. We have used it to study many problems, including the behavior of competing TCP connections on a packet-switched ring.

In the second part of this paper, we report on simulation results of the behavior of competing TCP connections on a packet-switched ring. We have identified the Òsmall-large unfairnessÓ property of TCP traffic under FIFO scheduling, and used it to explain the unfair bandwidth allocation among competing TCP connections on a uni-directional ring. We have also studied the effects of other scheduling methods, including PSPDQ, PSQ and 2LQ. The simulation results show that 2LQ can fairly allocate a ringÕs bandwidth among competing source nodes. At the same time it can also fairly allocate a source nodeÕs achieved bandwidth among flows destined to different destination nodes regardless of their numbers of TCP connections.

Acknowledgment

This research was partially supported by Nortel, Sprint, Air Force Office of Scientific Research Multidisciplinary University Research Initiative Grant F49620-97-1-0382, and National Science Foundation Grant CDA-94-01024.

References

[1] S.Y. Wang and H.T. Kung, ÒA Simple Methodology for Constructing Extensible and High-Fidelity TCP/IP Network Simulators,Ó IEEE INFOCOMÕ99, March 21-25, 1999, New York, pp. 1134-1143.

[2] S. McCanne, S. Floyd, ns-LBNL Network Simulator. (http://www-nrg.ee.lbl.gov/ns/)