NASA/TM—2004-212938

Evaluation of Swift Start TCP in Long-Delay Environment

Frances J. Lawas-Grodek and Diepchi T. Tran
Glenn Research Center, Cleveland, Ohio

October 2004

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the Lead Center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA Access Help Desk at 301–621–0134

- Telephone the NASA Access Help Desk at 301–621–0390

- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076

NASA/TM—2004-212938

Evaluation of Swift Start TCP in Long-Delay Environment

Frances J. Lawas-Grodek and Diepchi T. Tran
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center

October 2004

Acknowledgments

We would like to thank Robert Dimond for his assistance in setting up our single-flow and multiple-flow testbeds. We also greatly appreciated the guidance provided by Joseph Ishac and Mark Allman for the modification of the BBN source code, and also Will Ivancic for getting our project funded.

Available from

NASA Center for Aerospace Information
7121 Standard Drive
Hanover, MD 21076

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22100

Available electronically at http://gltrs.grc.nasa.gov

Evaluation of Swift Start TCP in Long-Delay Environment

Frances J. Lawas-Grodek and Diepchi T. Tran
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Summary

This report presents the test results of the Swift Start algorithm in single-flow and multiple-flow testbeds under the effects of high propagation delays, various bottlenecks, and small queue sizes. Although this algorithm estimates capacity and implements packet pacing, the findings were that in a heavily congested link, the Swift Start algorithm will not be applicable. The reason is that the bottleneck estimation is falsely influenced by timeouts induced by retransmissions and the expiration of delayed acknowledgment (ACK) timers, thus causing the modified Swift Start code to fall back to regular transmission control protocol (TCP).

Introduction

The slow start algorithm (ref. 1) is used at the beginning of a transmission control protocol (TCP) connection to probe a network capacity and to establish a self-clocking process. Slow start probes the network by transmitting packets at 1.5 to 2.0 times the rate of acknowledged data in each round trip until the congestion window has reached the slow start threshold or a dropped packet has occurred. With this growth pattern in a high-bandwidth-delay product (BDP) environment, TCP may take several seconds to transmit at the maximum rate allowed in the network. For small transfers, as in command strings or small files, TCP may have completed sending all the data before reaching the full capacity of the network. Also, the number of packets being released into the network exponentially increases in large bursts during the slow start phase, causing the buildup of a queue in bottleneck routers. In a network with a high BDP, these router queues may be smaller than the maximum TCP window, which leads to the dropping of packets and poor overall performance.

With a goal of solving these startup problems, BBN Technologies designed an algorithm called Swift Start (ref. 2), which is a combination of packet-pair and packet-pacing techniques. Packet pair is based on the spacing between two initial acknowledgments (ACKs) from data that have been sent in immediate succession. The algorithm provides an estimation of the current network bandwidth, thus allowing a TCP connection to obtain the network capacity much faster than slow start. Packet pacing is a technique that uses an estimated round-trip time (RTT) to spread out packets across an RTT, avoiding bursts of packets that are released to the network, and to establish an initial self-clocking process.

Since limited testing had been done by BBN Technologies, the goal of the project reported herein was to fully evaluate the Swift Start code in a testbed environment using variables of file size, bottleneck link, delay, and fraction of capacity estimation in a single-flow transfer. The code was examined under the influence of different rates of user-datagram-protocol (UDP) background traffic inserted in the link. In the initial testing, we found and corrected some coding flow problems with the capacity estimation and pacing section in the BBN code. By testing the modified code, the results showed that capacity estimation helps to improve the throughput in small and moderate file transfers of the single-flow testing, but the code needs further changes to perform better in long file transfers and in a multiple-flow environment. The pacing-only tests showed dropped packets later in the connection as compared with regular TCP, which proved that the bursts of packets sent in TCP caused the network routers to be overloaded sooner

than with pacing only. On the other hand, although packet pacing reduced the large buffering requirement in routers, there were higher latencies in packet transfer and more retransmissions in the pacing-only tests as compared with TCP tests. Therefore, consistent with the result found in reference 3, the average throughput of the pacing-only technique in the test was about equal to or lower than regular TCP throughput.

The following sections of this report discuss the problems found in the Swift Start code, the testing of the code and the modifications made to it, the testbed configurations, test variables, and results of the single- and multiple-flow tests. The conclusions and recommendations for future work may help to improve the performance of Swift Start.

Appendix A contains a glossary of terms used herein and appendix B presents the calculated values of the congestion control window.

Swift Start Implementation Problems

The Swift Start code was obtained from BBN Technologies (ref. 2) and consisted of modifications to the TCP stack of a FreeBSD version 4.1 operating system kernel. However, after compiling the modified FreeBSD kernel with the Swift Start changes, it was noted that there were some coding logic problems with the capacity estimation and pacing sections.

For the capacity estimation, the code originally captured the synchronize-sequence-numbers-flag (SYN) packet as the first data packet, and also the SYN ACK packet as the first ACK, which consequently led to the incorrect calculation of the time interval between the acknowledgments (called delta). Since the delayed ACK option was used and an initial TCP window of four packets was set, the delta was supposed to be the time difference between the acknowledgement of the first two packets (th_ack_1) and the acknowledgment of the last two packets (th_ack_2). Instead, the original code computed the delta as the time difference between the SYN ACK and the th_ack_2, which then made the delta value much larger than the correct value. Because of this miscalculation, the capacity estimation window (ce_cwnd), which equaled (SegSize × RTT/delta) (ref. 2), was estimated to be less than the current send congestion window (snd_cwnd), and the code was written to use the value of snd_cwnd for the second RTT. As shown in figure 1, only six packets are paced out in the second RTT instead of 216 packets per the theory for a 5-Mbps bottleneck link with a 500-ms delay and a packet size of 1448 bytes.

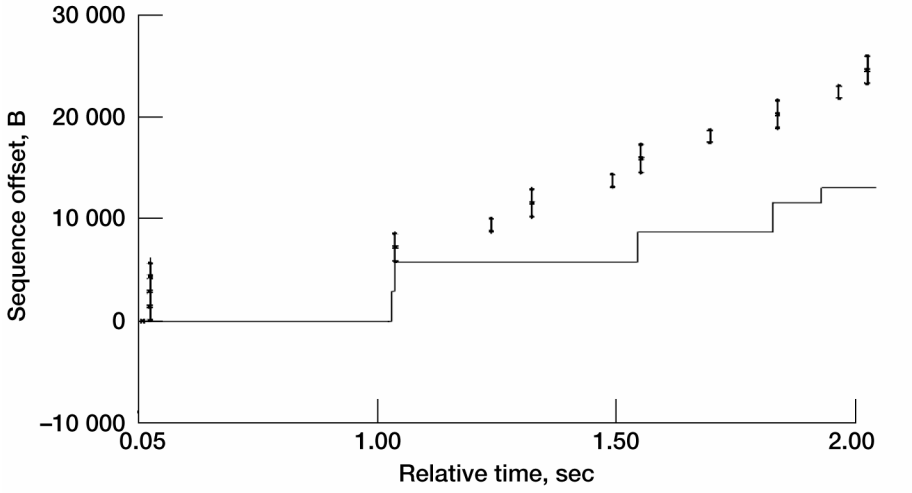

Figure 1.—Underestimated capacity due to too large delta.

To correct this problem, the code was modified to capture the first data packet as the start sequence instead of the SYN packet. This modification helped to correctly compute the delta as the difference between the time stamps of the actual th_ack_1 and th_ack_2. The RTT was then calculated as the time interval between the first packet sent and the th_ack_1 received. To obtain a more accurate estimation of capacity, we also changed the method of calculation to use a byte-based count derived from the packet sequence number instead of using packet counting as in the original code[1]. In addition, code was added to abort the capacity estimation if there were any retransmissions and/or if the delay timer expired after getting the th_ack_1 but before the arrival of the th_ack_2. If either of these two conditions had occurred, there would not have been a correct estimation of the bottleneck bandwidth. Furthermore, we changed the code to abort the pacing if the ce_cwnd was estimated as less than or equal to the current snd_cwnd because pacing just the last three of six packets in the second RTT[2] would not make a significant difference between slow start and pacing.

Besides the problem with capacity estimation, there were also problems with the pacing performance. First, the pacing process was starting too early before the ce_cwnd was computed. Because of this start, after the th_ack_1 was received, the pacing code used the current snd_cwnd to pace the first three packets of the second RTT. As a result, there was a timing gap between the first two packets and the third packet, as shown in figure 2.

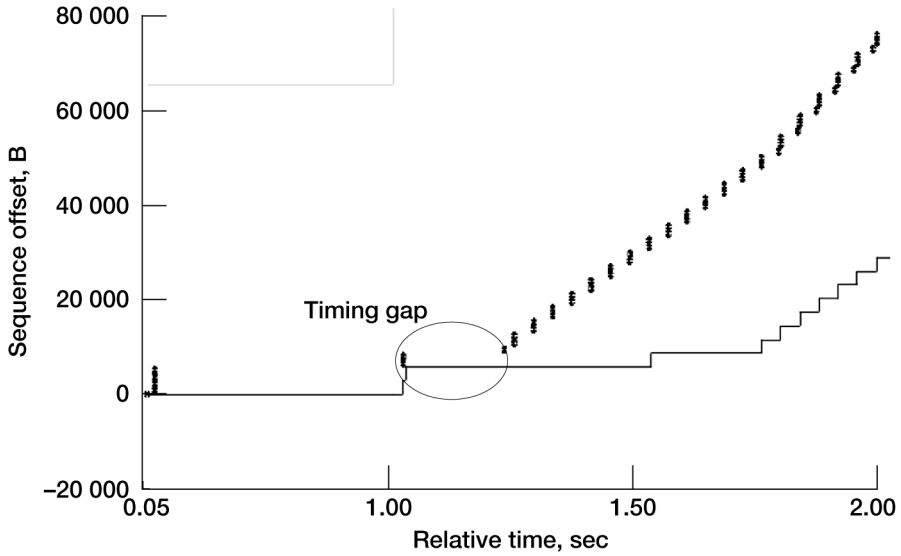

Figure 2.—Premature pacing.

Second, after sending all the packets as calculated in the ce_cwnd, the pacing did not stop and slow start was not invoked. To correct these problems, we modified the code to start pacing only when the calculated ce_cwnd had been defined and to stop pacing when a number of packets up to the value of the ce_cwnd had been sent. In addition, at the end of pacing, we decreased the snd_cwnd to account for the number of ACKed packets to accommodate for the ACKs that arrived during the pacing. Without this adjustment, TCP would have sent a burst of packets at the end of the pacing.

[1]The original BBN code used a calculation for ce_cwnd based on a segment size of 1448 bytes, the size of one packet. However, during the delta interval, two packets are sent (2896 bytes), not 1448 bytes as indicated in the BBN code. For the inadequate size of 1448 bytes of data, the ce_cwnd will be calculated to be smaller than expected, thus underestimating the bottleneck bandwidth. Accordingly, we changed the code to use a byte-based count method for a more accurate calculation of ce_cwnd.

[2]The first three packets were already sent in response to the th_ack_1 before the ce_cwnd was calculated.

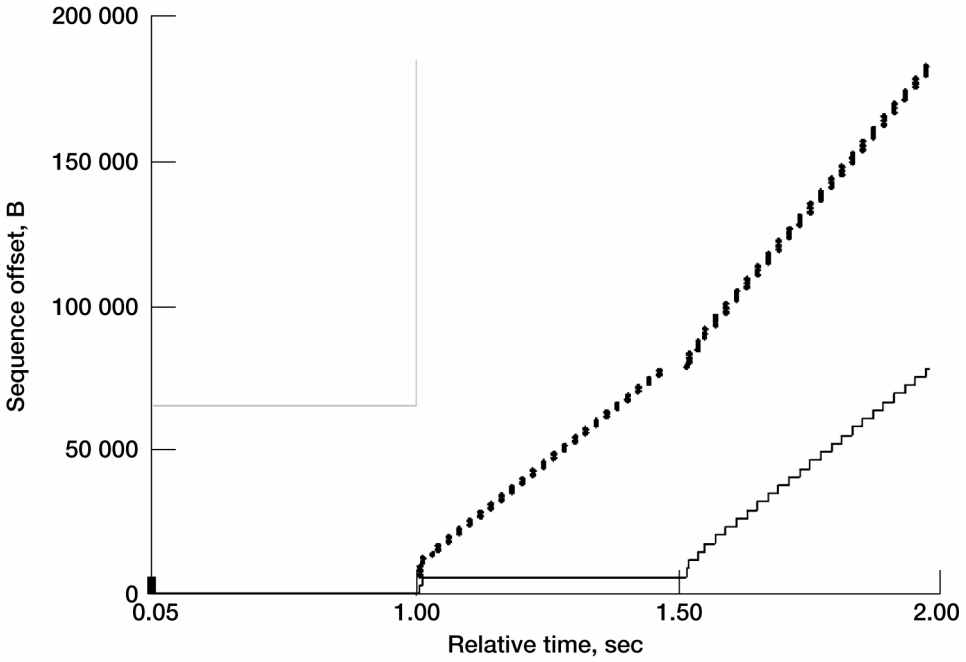

Figure 3.—Corrected capacity estimation and pacing.

Figure 3 shows the first few seconds for a 1-MB file transfer at a 5-Mbps bottleneck with a 500-ms delay after all the aforementioned modifications to the capacity estimation and pacing had been made. In the plot, the packets are paced nicely in the second RTT and then slow start became involved at the beginning of the third RTT.

Test Results

In both single- and multiple-flow testing, the FreeBSD version 4.1 kernel was used with Swift Start patches applied, and different ports were assigned to switch between the TCP (Reno) and Swift Start tests.

Single-Flow Testing

Single-flow testbed configuration.—As shown in figure 4, the testbed environment for the single-flow tests consisted of two separate networks, a terrestrial and a space network. The two networks were interconnected via several unique asynchronous transfer mode (ATM) virtual circuits passing through an Adtech SX/14 channel simulator. The SX/14 allows the insertion of time delays and random bit errors in the network flows. The networks on each side of this Adtech channel consisted of a CISCO 7100 router and a CISCO 2900 catalyst Ethernet switch. The catalyst switches served as the local area networks (LANs) connecting to transfer senders, receivers, or analyzers for the tests. In addition, on the sending side of the network, a Fore ASX−200BX ATM switch was added into the network before the Adtech channel simulator to assure a constant bit rate (CBR) because the CISCO routers could only guarantee an unspecified bit rate (UBR), not a constant one.

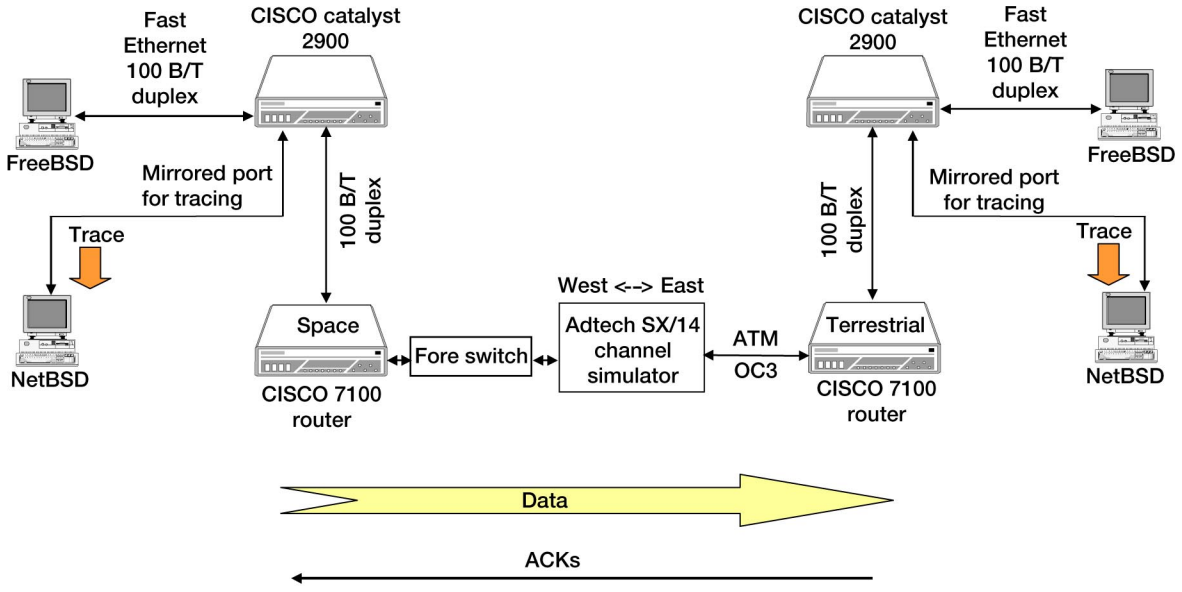

Figure 4.—Testbed for single-flow testing.

Hosts on the CISCO switches were connected either to an active port to allow the system to participate in a traffic flow or to a mirrored port to allow the system to analyze traffic to and from a specific host. In addition, hosts on each LAN were configured with two physical interfaces: the first was used for external access during the tests to avoid impeding traffic flows that were being analyzed; the second was used only for the actual test flow that was being measured and analyzed.

The receiving and sending hosts were set up with the FreeBSD version 4.1 operating system, whereas already established hosts with the NetBSD version 1.5 operating system were used as the monitoring and analyzing machines for the traffic flows.

The BBN Swift Start source code kernel patch was applied to the FreeBSD sending and receiving machines. In addition to applying BBN's patches to the TCP stack of the FreeBSD kernel source and defining new kernel options related to this capacity estimation and pacing algorithm, the kernel option "HZ" was set to 1000 (1 ms). This option allowed a finer timer granularity, which would then affect the calculation of delta. Other FreeBSD options and variables that changed from their default are presented in table 1.

TABLE 1.—OPTIONS AND VARIABLES FOR FreeBSD KERNEL SOURCE

Kernel	
NMBCLUSTERS, byte	32 768
MAXMEM, byte	400 × 1024
Sysctl	
kern.ipc.maxsockbuf, byte	7 340 032
debug.bpf_bufsize, byte	32 768
net.inet.tcp.sendspace, byte	3 125 000
net.inet.tcp.recvspace, byte	3 125 000
net.inet.tcp.local_slowstart_flightsize, packets	4

The NetBSD monitoring machines on each side ran the commonly used tcpdump (ref. 4) software to capture packet information about the test flows and then tcptrace (ref. 5) and xplot (ref. 6) to analyze and display the packets. Bottleneck links of 5 and 20 Mbps were set up using a UBR-limited circuit on the ATM interface of the sending CISCO router and a CBR-limited circuit on the Fore switch. The CBR-

limited circuit could not be used solely for the tests because it had a strict limitation on the speed of the data transfers and caused errors during reassembly of the ATM packets. As a result, hardly any packets were able to pass through this CBR link without being retransmitted in this condition. For the 500-kbps bottleneck link, a UBR-limited circuit was still used in the sending CISCO router, but the CBR in the Fore switch on the receiver side was set to a full pipe flow to avoid the problem in reassembly of the ACKs on the return link for this slow link.

Table 2 presents the variables for the Swift Start code and the regular TCP.

TABLE 2.— TEST VARIABLES FOR SWIFT START AND TCP

Delay, ms	100, 500
Bottleneck rate, Mbps	0.5, 5, 20
File size, MB	
For bottlenecks of 5 and 20 Mbps	50
For bottleneck of 500 kbps	20
For bottlenecks of 5 and 20 Mbps and 500 kbps	1
Gamma (Swift Start tests only)	1,4,8

Evaluation of capacity estimation.—As indicated in the BBN report (ref. 2), although the packet-pair approach was used in the Swift Start code for estimating the bottleneck bandwidth, BBN was aware of some problems that might occur with this algorithm in real networks, that is, ACK compression (ref. 7) and extra time introduced by the delayed ACK timers.

The packet-pair performance was evaluated by itself by first running single-flow tests through the network without any other traffic. Based upon our own internal logging, most of the estimated values of ce_cwnd were acceptable. It was noted during the tests that the estimations of the RTTs increased as the speed of the bottleneck link decreased. Since the packets needed extra time to send through the slower bottleneck link (e.g., for a 500-ms delay), the RTTs calculated in the Swift Start code appeared to be about 3 to 4 ms, 6 to 7 ms, and 50 to 52 ms longer as compared with the RTTs directly defined in the hardware for the bottleneck links of 20, 5, and 0.5 Mbps, respectively. Furthermore, since a 1-ms clock was used and the RTT, delta, and ce_cwnd were defined as integers in the tests, the estimation values of the RTT and delta were computed in milliseconds whereas the values of ce_cwnd were rounded off to the next lower integer.

The test variables presented in table 2 were used to perform 15 tests for each file size. Observe that the ce_cwnd was estimated 90 times (15 test runs with 2 file sizes and 3 gamma values) at each delay for each bottleneck link. Table 3 summarizes the number of ce_cwnd estimations and their percentage of the estimated ce_cwnd in comparison with the calculated values presented in appendix B. The percentage of estimated values as compared with the calculated values is computed by the following formula:

$$\frac{\text{Estimated ce\_cwnd} \times 100}{\text{Calculated ce\_cwnd}} \tag{1}$$

Percentage values greater than 100 indicate that the ce_cwnd has been overestimated for that test; otherwise, they are underestimated as compared with the calculated ce_cwnd values. The third column (number of test runs) gives the number of tests that have estimated ce_cwnd values at the indicated percentage as compared with the calculated values. For instance, with a 20-Mbps bottleneck link at a 500-ms delay, there are 70, 4, and 16 test runs that have estimated ce_cwnd values of 116.50, 116.70, and 58.30 percent, respectively, as compared with the calculated value.

TABLE 3.—PERCENTAGE OF TIMES CAPACITY ESTIMATION CONGESTION WINDOW (CE_CWND) IS MET

Bottleneck link rate, Mbps	Delay, ms	Number of test runs	Percent of estimated ce_cwnd
20	500	70	116.50
		4	116.70
		16	58.30
	100	79	119.30
		5	59.70
		6	59.10
5	500	60	94.00
		20	93.80
		9	117.50
		1	78.20
	100	56	98.20
		27	99.20
		1	81.90
		5	123.90
		1	125.10
0.5	500	38	104.20
		6	104.40
		14	102.10
		1	111.60
		1	106.60
	100	23	142.80
		1	141.90
		3	139.90
		2	139.00
		1	143.70

As shown in the preceding table, there are 90 ce_cwnd estimations each for bottleneck links of 20 and 5 Mbps. However, there are only 60 and 30 estimations for a bottleneck link of 500 kbps at delays of 500 and 100 ms, respectively. Sixty runs are for the gamma values of 1 and 4 at a 500-ms delay, and 30 runs are for a gamma of 1 at a 100-ms delay[3]. With this slower link, when the gamma is set at 8 for a 500-ms delay and the gamma is set at 4 and 8 for a 100-ms delay, the estimated ce_cwnd values are smaller than the values of the current snd_cwnd. As a result, as indicated in the preceding section, these estimated values were not used. In these cases, the code used the values of the current snd_cwnd and then normal slow start was invoked since pacing was aborted.

For each delay, the majority of the estimations were more accurate as the speed of the bottleneck link became lower, except for the 100-ms delay at 500 kbps. For the slower bottleneck link, the time interval between the first two ACKs (delta) was larger compared with that of the faster bottleneck link, and the estimated ce_cwnd values were less sensitive to the variation of this delta. For instance, at a 20-Mbps link, when the estimated delta equaled 1 ms, the value of the estimated ce_cwnd could be doubled as compared with that of ce_cwnd when the delta equaled 2 ms. However, for the 5-Mbps link, the ce_cwnd

[3]Gamma is a configurable value that is used to help protect against an overestimate from the packet-pair algorithm. A higher value of gamma will give a smaller estimate of the bottleneck bandwidth. The greater the certainty of the actual speed of the bottlenecks in the network, the lower the value of gamma that can be set.

with a delta of 4 ms was only 1.25 times greater than that with a delta of 5 ms. As a result, the variation among the estimated values in this 5-Mbps bottleneck link was not as large as compared with that of a 20-Mbps bottleneck. As shown in table 3, there is a bigger gap between the percentage values of the estimated ce_cwnd (116 and 58 percent at a 500-ms delay) for a 20-Mbps bottleneck link as compared with those gaps in the slower bottleneck link (117 and 78 percent in a 5-Mbps bottleneck link and 111 and 102 percent in a 500-kbps bottleneck link at a 500-ms delay).

Except for the 100-ms delay at a 500-kbps bottleneck link, the percentage of the majority of the estimated ce_cwnd values is in the range of 98.2 to 119.3 percent as compared with the calculated ce_cwnd values. For the 100-ms delay at a 500-kbps link, the extra approximately 50 ms in the RTT estimation (as mentioned earlier in this section) make the estimated ce_cwnd about 140 percent of the calculated values. Also, note from table 3 that with this speed of bottleneck link, these extra 50 ms in the RTT do not have as great an impact on the estimated ce_cwnd values at a 500-ms delay as they do at a 100-ms delay.

Performance comparison of Swift Start and regular TCP.—As expected, since the estimated bottleneck bandwidths (ce_cwnd) define the number of packets to be sent in the second RTT for the Swift Start tests, the Swift Start tests sent out more packets in the first few RTTs than did the regular TCP, but dropped packets occurred earlier during the Swift Start transfers than they did in the regular TCP tests. This condition is an advantage for a short-file-size transfer, as discussed later in this section. Table 4 shows the Swift Start versus TCP time and number of packets sent for the three bottleneck links. As seen in the table, Swift Start sent many more packets during the first few seconds of the transfers as compared with the slow start performance of TCP (e.g., note the 5- and 20-Mbps bottleneck links under a 500-ms delay). This result is in agreement with the goal that Swift Start be designed to improve performance in a high-bandwidth-delay product environment.

TABLE 4.—COMPARISON OF NUMBER OF BYTES SENT AT SPECIFIC INTERVALS BY SWIFT START AND TCP

Test condition	Swift Start			TCP
	Gamma			
	1	4	8	
Delay, 500 ms				
Bottleneck link, Mbps	20	20	20	20
Time, sec	2	2	2	2
Number of bytes sent	3 091 481	887 625	457 569	47 785
Bottleneck link, Mbps	5	5	5	5
Time, sec	4	4	4	4
Number of bytes sent	2 481 873	1 339 401	697 937	194 033
Bottleneck link, kbps	500	500	500	500
Time, sec	25	25	25	25
Number of bytes sent	2 098 100	[a]1 953 352	Same as TCP	1 998 241
Delay, 100 ms				
Bottleneck link, Mbps	20	20	20	20
Time, ms	900	900	900	900
Number of bytes sent	2 337 073	1 585 561	981 744	262 088
Bottleneck link, Mbps	5	5	5	5
Time, sec	3	3	3s	3
Number of bytes sent	2 423 953	2 143 941	[a]1 938 873	2 079 329
Bottleneck link, kbps	500	500	500	500
Time, sec	25	25	25	25
Number of bytes sent	[a]2 183 585	Same as TCP	Same as TCP	2 186 481

[a] Fewer number of packets sent than in regular TCP tests during same amount of time.

In table 4, note that the Swift Start test byte values that are marked with the footnote “a,” had sent fewer numbers of packets than were sent in the regular TCP tests during the same amount of time. In these cases, the ce_cwnd was estimated to be six packets, which is equal to the number of packets that

slow start sent out in the second RTT. However, since the first three packets were burst out in response to the first ACK, only the remaining three packets of these six were paced, and the last packet was paced near the end of the RTT. This action caused a delay in sending compared with the stacked-up packets during the slow start of the regular TCP. Those tests with the estimated ce_cwnd equal to less than the current snd_cwnd (five packets) were performed by regular TCP; no pacing was performed in these cases.

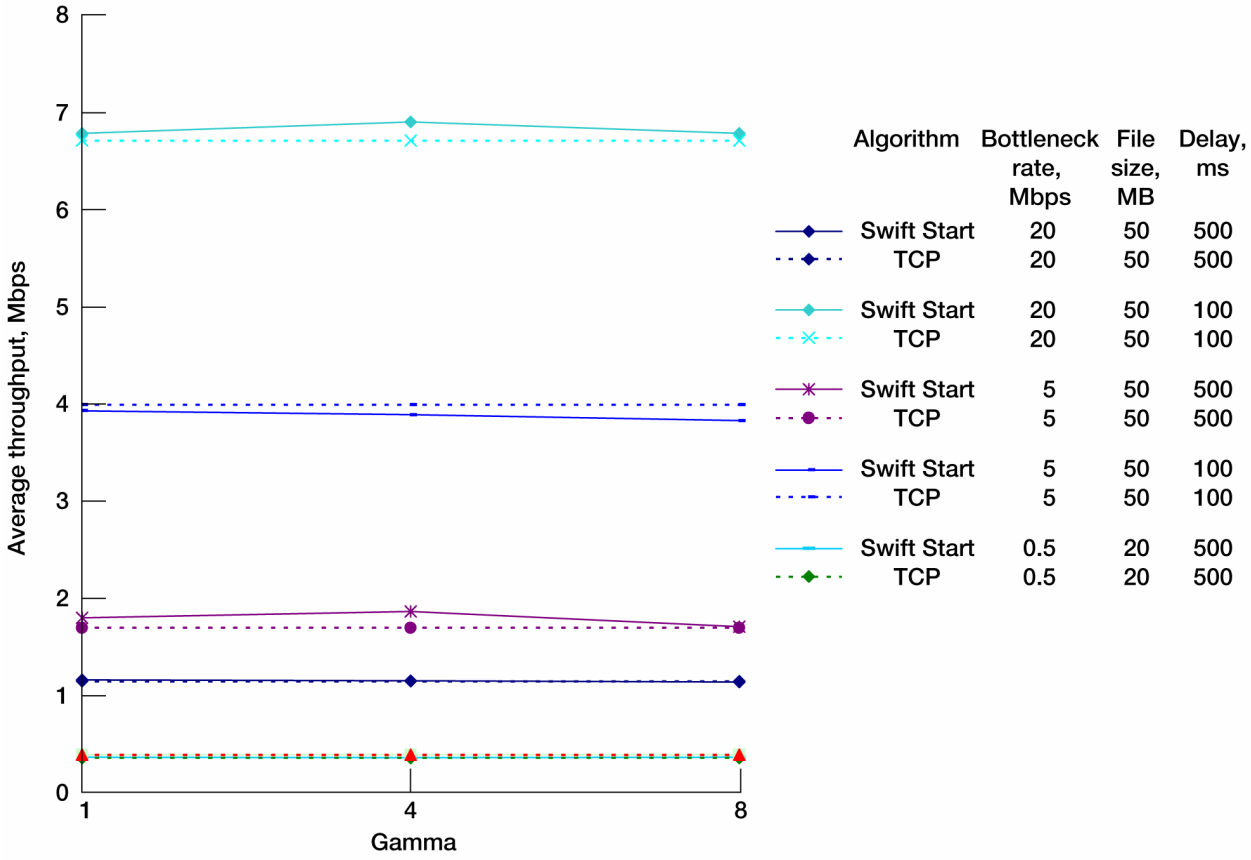

Figure 5.—Throughput of 20- and 50-MB file sizes versus gamma.

Figure 5 compares the throughput of Swift Start and TCP. The throughput of Swift Start does not indicate a significant advantage over TCP when the file size is defined large enough such that the router queue overflows, thus causing some dropped packets and the transfers continue to process for a time in the recovery phase. Files of 50 MB were used for 5- and 20-Mbps bottleneck links in these tests, and files of 20 MB were used for a 500-kbps bottleneck link. With these large file sizes, the Swift Start throughput was a little higher than the TCP throughput when the bottleneck link rate was set to 20 Mbps for a 100-ms delay and was also higher at a 5-Mbps bottleneck for a 500-ms delay at all tested values of gamma (1, 4, and 8). For the 20-Mbps bottleneck link at a 500-ms delay, the throughput of the Swift Start tests was slightly better than that of TCP only when the gamma values were set to 1 and 4. On the other hand, the TCP throughput was a little higher than that of the Swift Start at a bottleneck link rate of 5 Mbps with a delay of 100 ms. With the small bottleneck link of 500 kbps, the throughput of Swift Start and TCP was very similar for all values of the tested gamma values and propagation delay.

However, for the smaller file size of 1 MB, when the queuing overflow condition had not yet been reached, the throughput of Swift Start exceeded that of the TCP in 5- and 20-Mbps bottlenecks with a 500-ms delay because there were more packets sent in the first few RTTs of the Swift Start tests, as described at the beginning of this section. In the 5-Mbps bottleneck at a 100-ms delay, the Swift Start

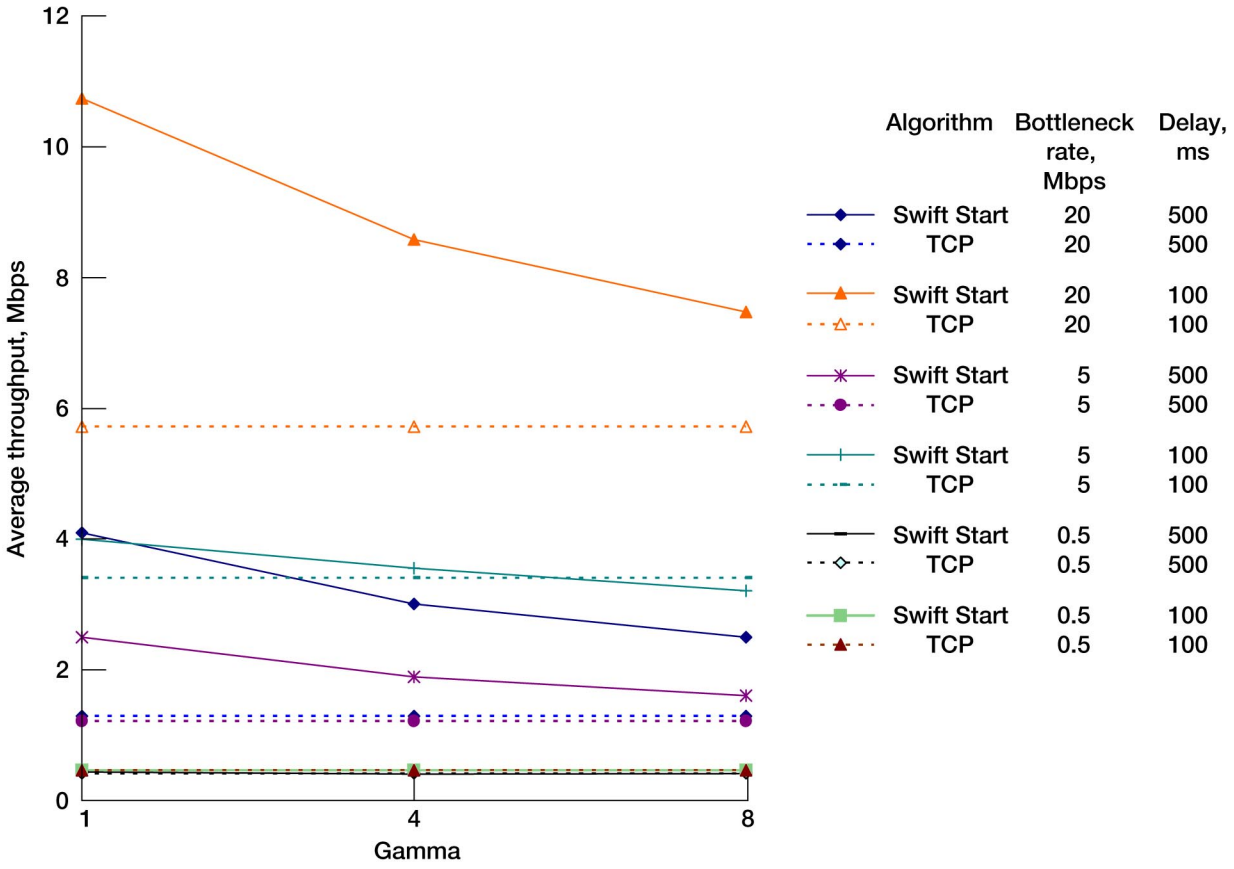

Figure 6.—Average throughput of 1-MB file size versus gamma.

throughput was slightly better than that of the TCP when the gamma values were 1 and 4. For the slower bottleneck link of 500 kbps, the throughput of this smaller file was about the same in the Swift Start and TCP tests. Figure 6 shows the average throughput versus the three settings of gamma for a 1-MB file in the Swift Start and TCP tests.

Multiple-Flow Testing

Observations.—The multiple-flow tests began using the same single-flow testbed with the bottleneck link set to 20 Mbps at a 500-ms delay. One 50-MB TCP flow was transferred followed by one Swift Start flow of the same size, and then the order was reversed so that the Swift Start flow started first followed by the regular TCP flow. All performed between one pair of sender and receiver machines. In both tests, the throughput of the flows was a little lower as compared with that of their single-flow tests. However, the ce_cwnd under Swift Start was still estimated as in single flow, which indicates that the network was not heavy congested. Therefore, there was no advantage seen for the Swift Start throughput over TCP with this setup.

With the same testbed at a 5-Mbps bottleneck link and a 500-ms delay and using a UDP rate-based flow of 5 Mbps via iperf (ref. 8) as background traffic sent from sender to receiver, a Swift Start flow overestimated the RTT (getting ~1600 ms). However, the delta was estimated to be the same as it was for the single-flow tests without any traffic. These estimations of RTT and delta led to a larger estimated bandwidth for the bottleneck link. Also, in this setup, sometimes the Swift Start flow would time out at the beginning of the transfer with the result that no capacity estimation or pacing was invoked since the estimation was false. Additional testing was performed for a background traffic of UDP rate-based transfers at 4 Mbps in both directions between the sender and receiver, but the RTT was still over-estimated in the Swift Start tests.

Configuration for multiple flow.—Based upon the observations of the preceding section, it was suspected that there is a deep queue inside the router. Ping tests were run to observe the RTT, but the queuing continually increased even past 2000 ms. It is believed that the reason for a large quantity of packets being queued is the way the router manages its system buffer pool; that is, the router creates more buffers if the interface buffers are full. As the result, a new testbed that allows us to control the buffer was created for the multiple-flow tests using the same pair of sender and receiver FreeBSD machines as used in the single-flow tests. Both machines were connected to a third FreeBSD machine that ran dummynet (ref. 9) between the first two machines. Dummynet was chosen as the network simulator as it could specify the speed of the bottleneck, delay, and queue size for the link. The same sender and receiver machines on each side of the dummynet were also used to capture traffic information for analyzing the packet flow instead of using entirely separate monitoring machines such as those in the single-flow tests. Figure 7 shows the testbed configuration for the multiple-flow tests and table 5 gives the variables for the multiple-flow tests.

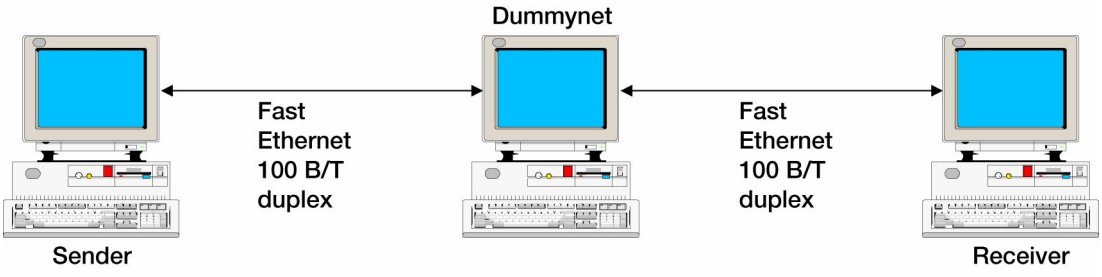

Figure 7.—Testbed for multiple-flow tests.

TABLE 5.—VARIABLES FOR MULTIPLE-FLOW TESTS

Delay, ms	500
Bottleneck rate, Mbps	5
Bottleneck queue, slots	5, 20, 50
Background traffic, Mbps UDP	2, 4
File size, MB	50
Gamma[a]	1,4,8
Kernel	Capacity estimation and pacing[b] and pacing only

[a]Only for tests with capacity estimation and pacing kernel.
[b]Same kernel as that used in single-flow tests.

Multiple-Flow Test Results

Testing with capacity estimation plus pacing kernel.—Using the test variables given in table 5, 15 tests at a 500-ms delay of Swift Start or TCP were performed back to back for each queue size with background traffic of 2 or 4 Mbps. Although a UDP background flow was created in both forward and return links via iperf, some ACK compressions still occurred, resulting in overestimated ce_cwnd values. In addition, when using a bottleneck queue of 5 slots, the delayed ACK timer expired before the delta could be calculated, which caused the Swift Start algorithm to abort and the transfer to fall back to the regular TCP slow start, per written in the kernel code.

Also, when the 2- and 4-Mbps UDP background traffic flows were inserted in the link with a bottleneck bandwidth of 5 Mbps, it was expected that Swift Start would give the estimation for the available bandwidths of 3 and 1 Mbps, respectively. However, it is likely that the capacity estimation algorithm in the Swift Start code was estimating the total bandwidth of the bottleneck link and not the available bandwidth remaining after the link became congested.

In general, all the bandwidth values were overestimated when gamma was set to 1 for both rate settings of the UDP background traffic (2 and 4 Mbps). With a gamma of 1, the majority of the estimates of bandwidth values were in the range of 3.3 to 4.6 Mbps and 3.8 to 4.6 Mbps with 2- and 4-Mbps background traffic, respectively. For gammas of 4 and 8, most of these estimated values were about 1 to 1.4 Mbps and 0.3 to 1 Mbps for both UDP background traffic. Figures 8 to 13 show the estimated bandwidth values with the two background traffic rates. As shown in these figures, in each set the estimated values of bandwidth that are much higher than the majority are caused by the ACK compressions. For example, the dissimilar estimated rate at 23 Mbps in figure 9 is caused by ACK compression.

By using a 2-Mbps background UDP traffic, the average throughput of the Swift Start tests was slightly higher than that of the TCP tests for most of the settings, except when the bottleneck queues were set to 5 and 20 slots and the gamma was equal to 1. For these settings, the average throughput of the Swift Start tests was a little lower than the TCP tests since the overestimated bandwidth described above had a greater impact on the throughput of these smaller queue tests as compared with the bigger queue tests of 50 slots. With a background UDP traffic of 4 Mbps, the average throughput of the Swift Start tests was slightly lower than the TCP throughput for queue sizes of 5 and 50 slots but was a little higher than that of TCP with a queue size of 20 slots. Figures 14 and 15 show the average throughput of Swift Start and TCP with 2- and 4-Mbps UDP background traffic, respectively.

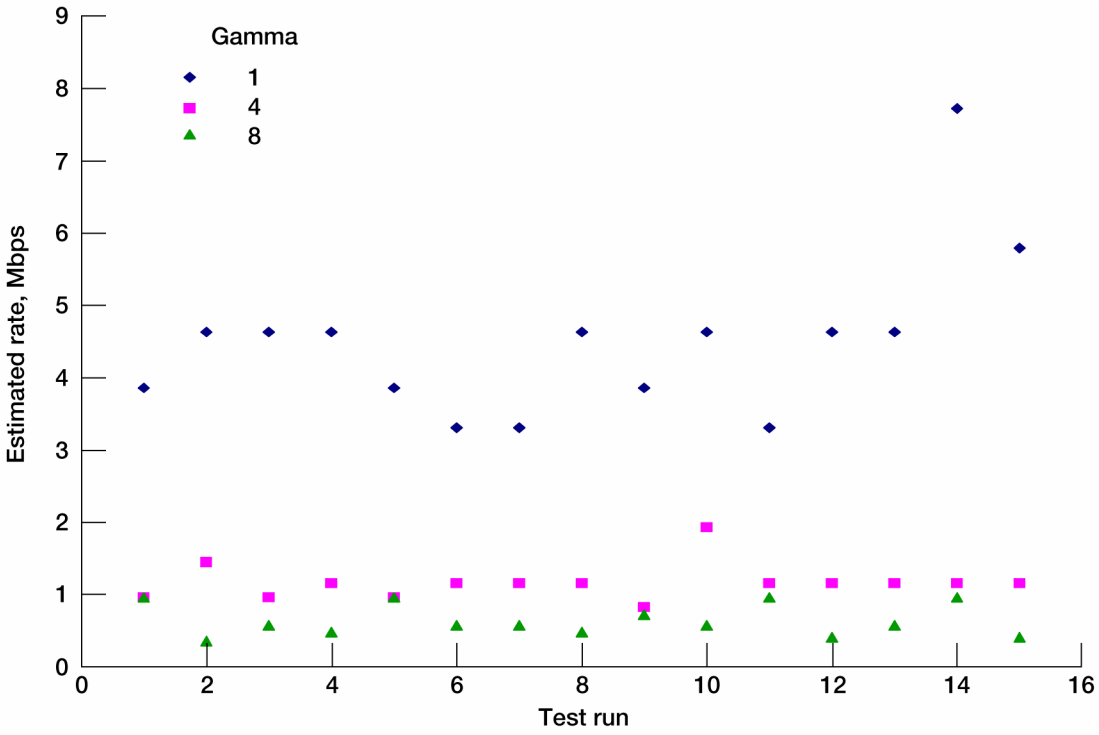

Figure 8.—Estimated bandwidth rates with 2-Mbps UDP background traffic and queue of 50 slots.

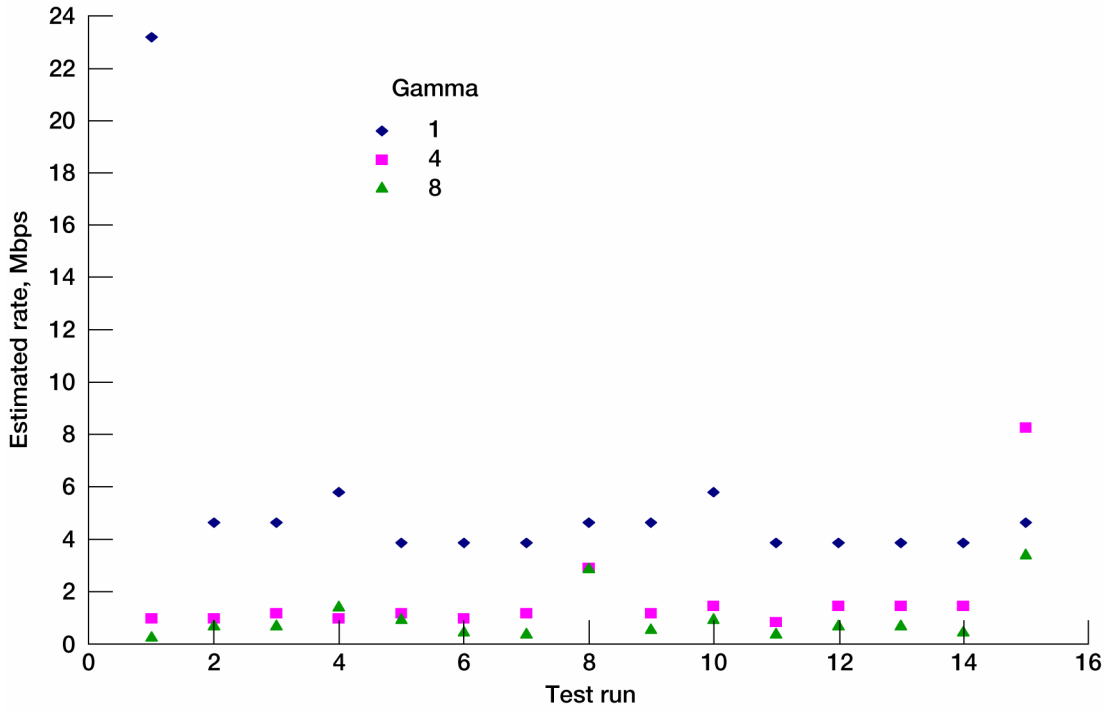

Figure 9.—Estimated bandwidth rates with 4-Mbps UDP background traffic and queue of 50 slots.

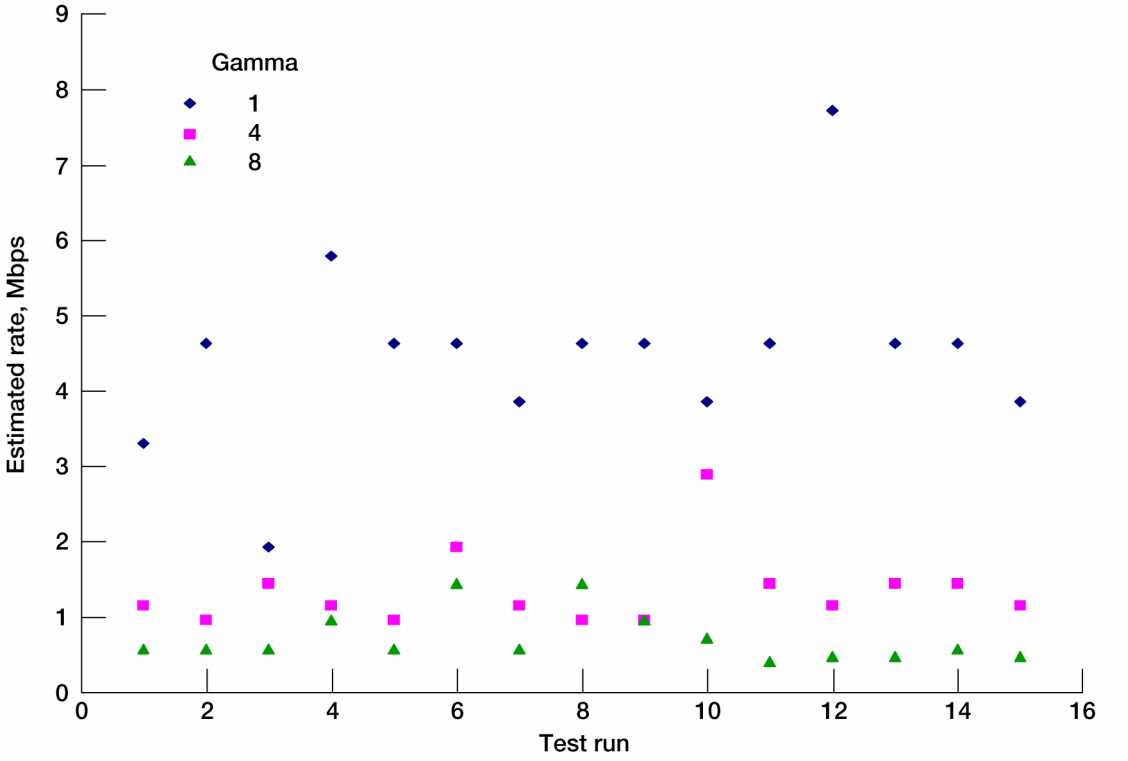

Figure 10.—Estimated bandwidth rates with 2-Mbps UDP background traffic and queue of 20 slots.

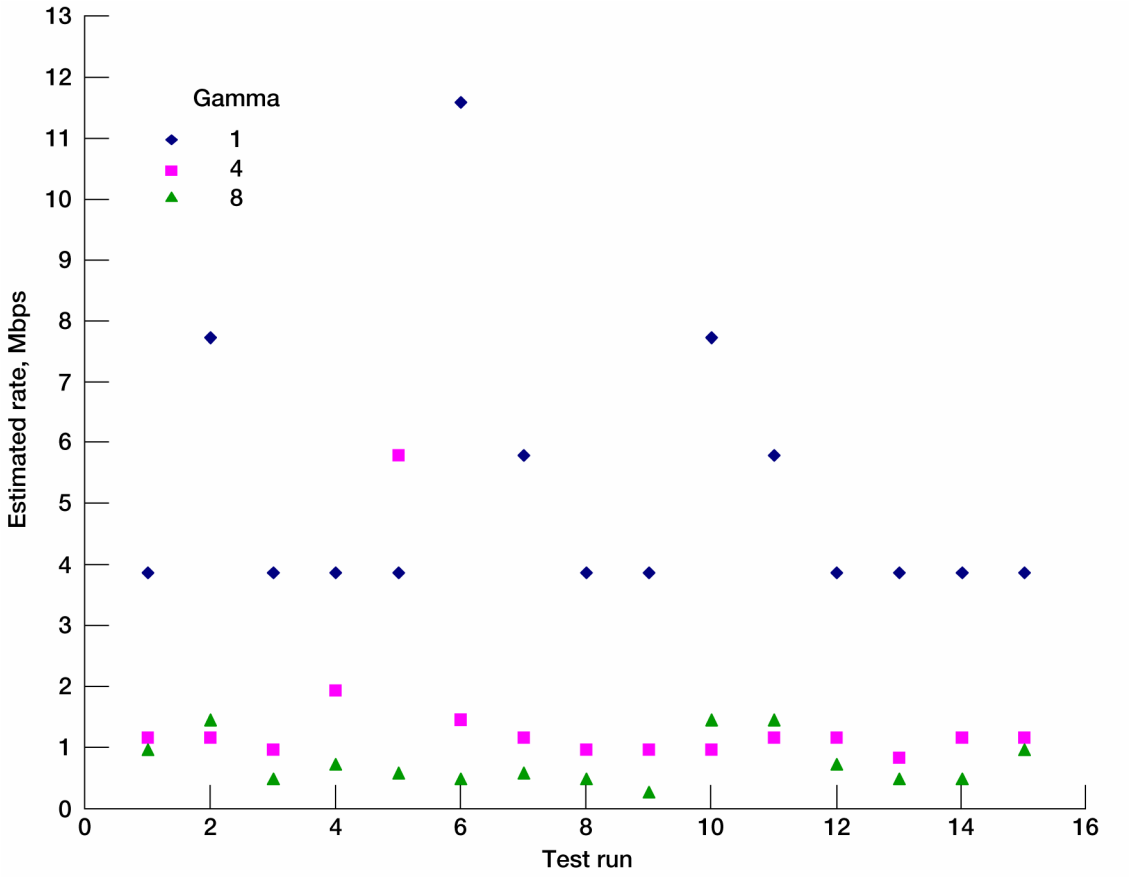

Figure 11.—Estimated bandwidth rates with 4-Mbps UDP background traffic and queue of 20 slots.

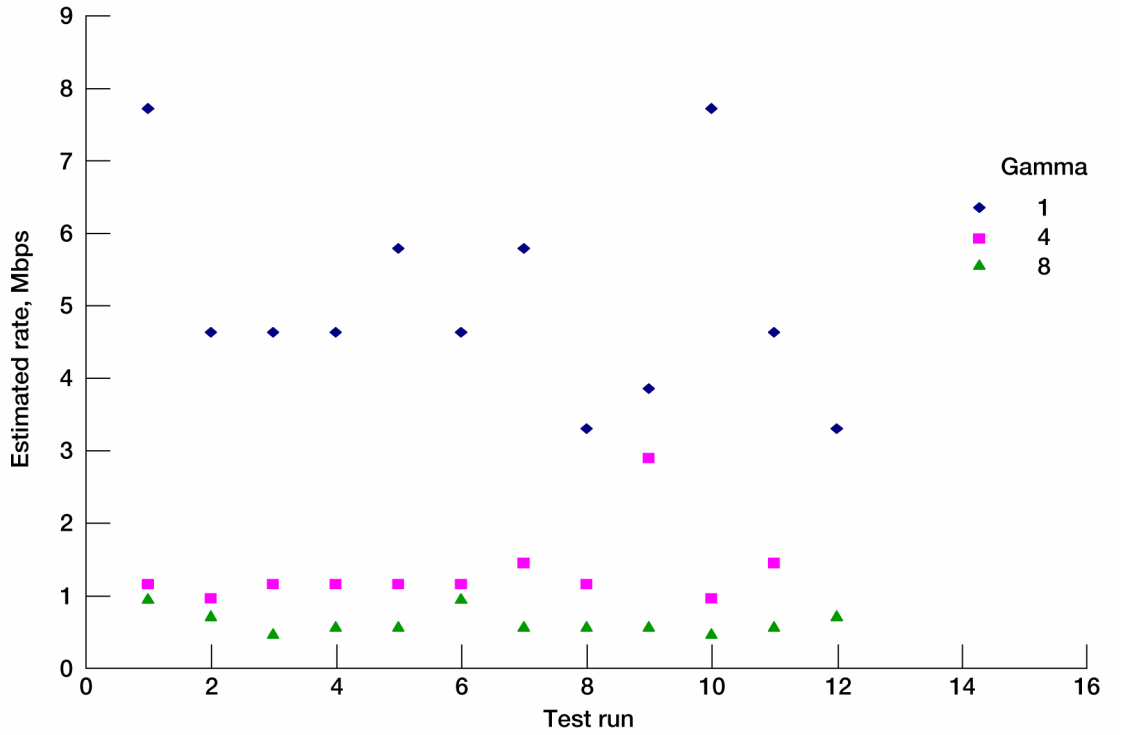

Figure 12.—Estimated bandwidth rates with 2-Mbps UDP background traffic and queue of 5 slots.

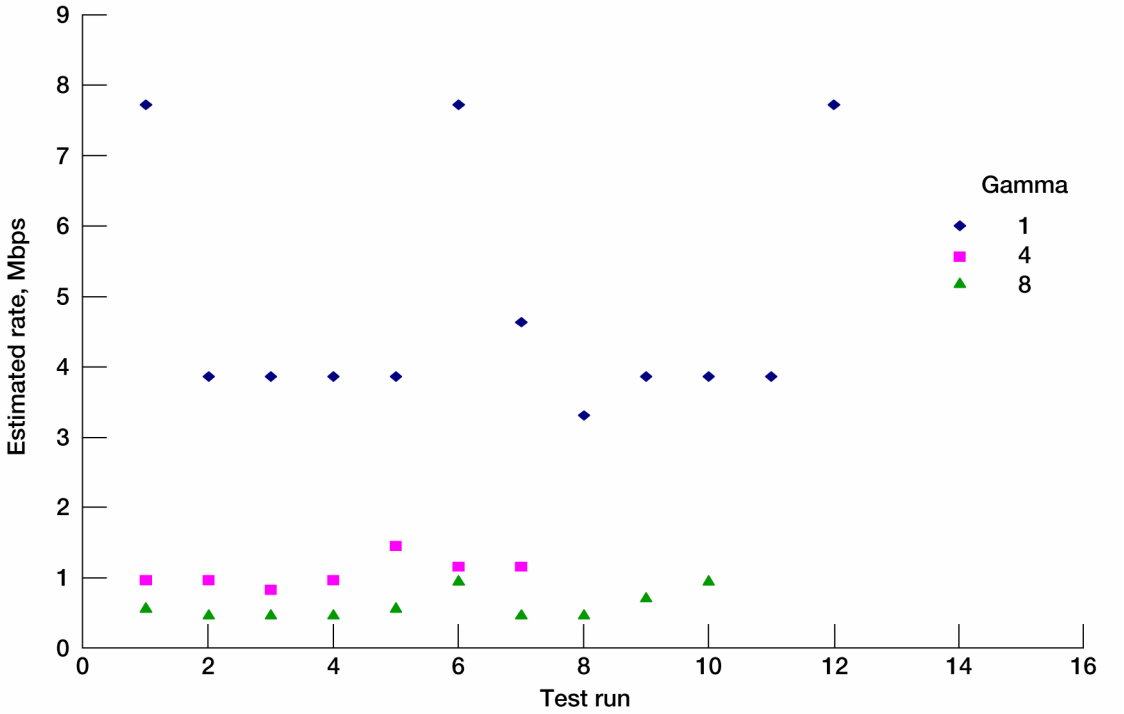

Figure 13.—Estimated bandwidth rates with 4-Mbps UDP background traffic and queue of 5 slots.

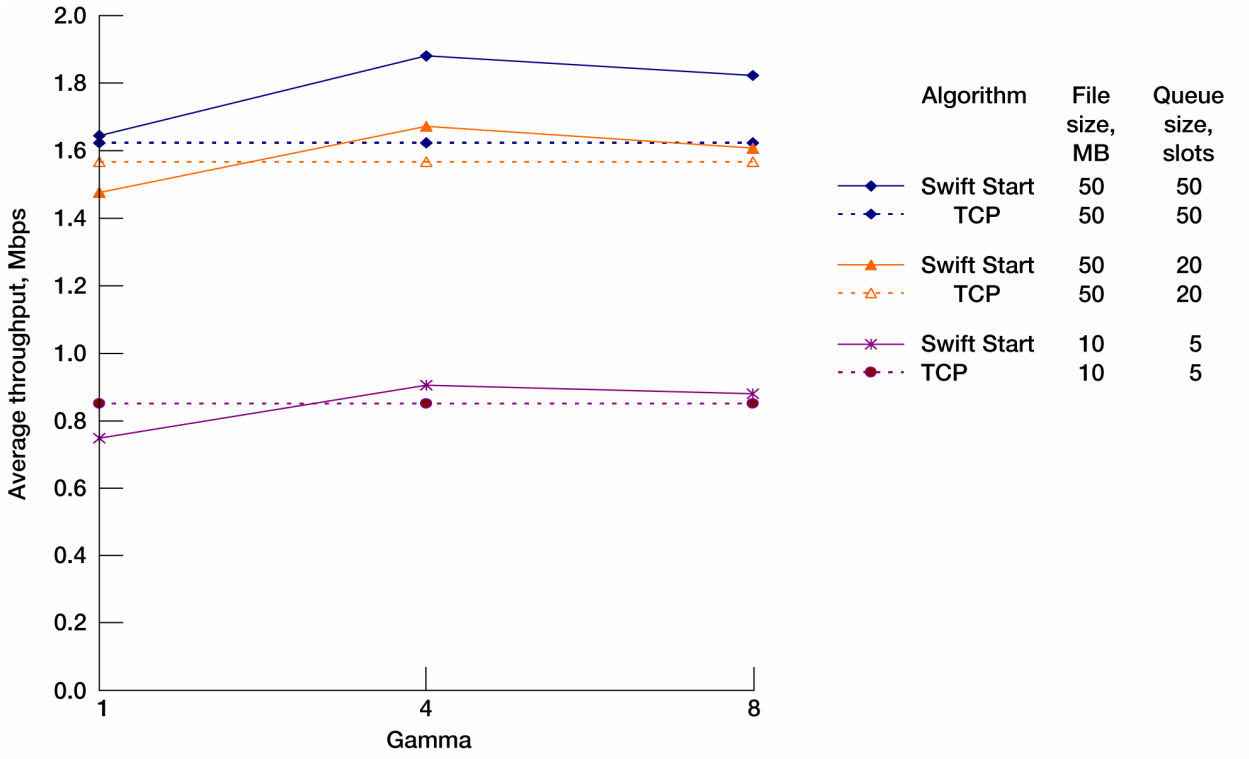

Figure 14.—Throughput versus gamma with 2-Mbps UDP background traffic for 50- and 10-MB file sizes.

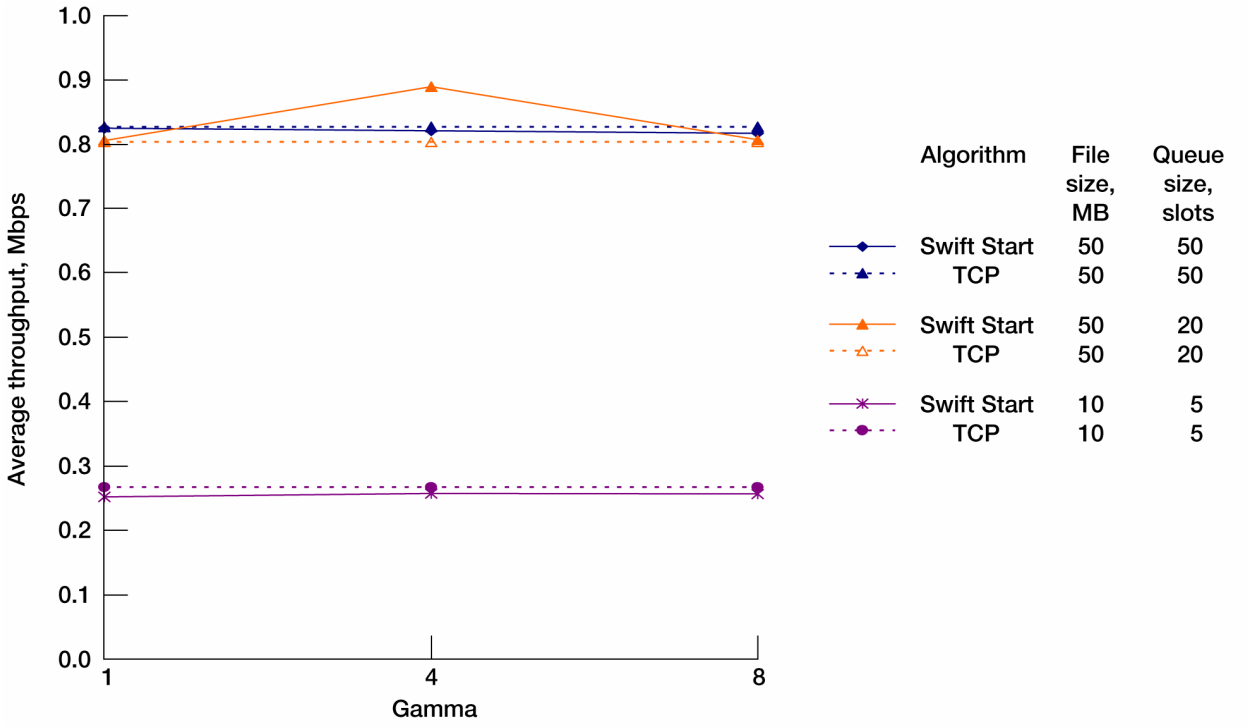

Figure 15.—Throughput versus gamma with 4-Mbps UDP background traffic for 50- and 10-MB file sizes.

Testing with pacing only.—In this kernel, capacity estimation was not used, and six packets were purposely paced out (same number of packets sent in slow start) in the second RTT. However, because of time constraints, we were unable to achieve a smooth pacing of the three pairs of packets (pacing sending a pair at a time in a 2-2-2 pattern) during the RTT. To obtain this even pacing, a considerable amount of modification to the pacing algorithm in the Swift Start code would have been necessary. Eventually, a pacing pattern flow of 2-1-2-1 packets was used for this test.

Throughout the tests using this pacing-only kernel, we observed that because of the bursts in slow start, the TCP tests dropped packets earlier in a connection as compared with the Swift Start tests. However, since the congestion window of Swift Start at the time the dropping occurred was larger than that of TCP, there were more overshoot packets in Swift Start tests as compared with TCP. That would be one of the reasons why there were more retransmitted packets in the Swift Start tests than there were in TCP, particularly for queue sizes of 20 and 50 slots. This retransmission resulted in the Swift Start average throughput being a little lower and about the same as compared with the average throughput of the TCP tests with 2- and 4-Mbps UDP background traffic, respectively. Figure 16 shows the average throughput of Swift Start and TCP using the pacing-only kernel.

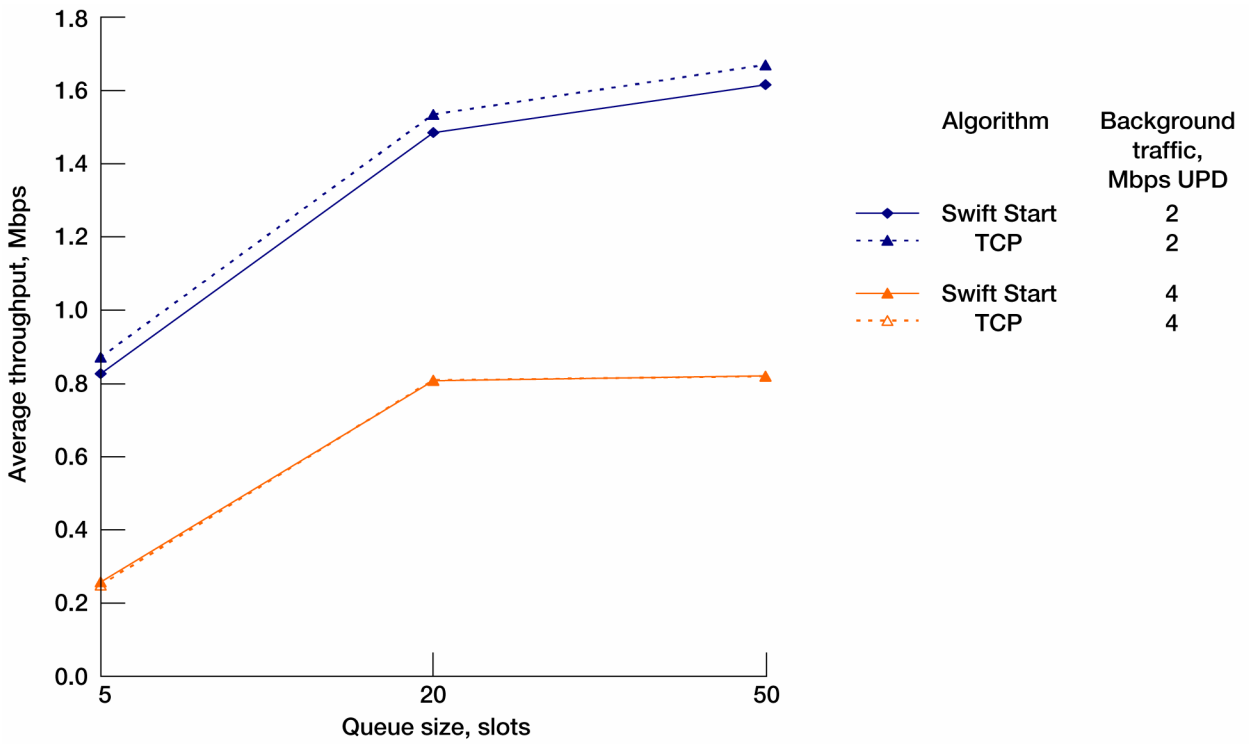

Figure 16.—Average throughput versus bottleneck queue using pacing-only kernel.

Recommendations

As mentioned previously, changes will need to be made to the Swift Start code that was provided by BBN Technologies in order to correct some flaws in the capacity estimation and pacing routines. Also, additional code is needed to stop the use of the estimated cwnd and to revert to the use of regular TCP in some conditions where the estimated ce_cwnd values would not be accurate, such as when the delayed ACK timer expired and retransmissions occurred before the delta was defined.

For large-file-size tests, after some dropped packets occurred followed by a recovery period, the advantage of Swift Start early in the connection was diminished because of the additive increase and multiplicative decrease algorithm of TCP. In this case, the cooperation of a less conservative method to

dropped-packet algorithms, such as scalable-TCP (ref. 10) or high-speed-TCP (ref. 11), in addition to the Swift Start algorithm may improve performance.

Several suggestions for handling multiple-flow conditions follow:

1. In light of the fact that the network usage fluctuates, continuously probing and estimating the available bandwidth would be helpful to correctly estimate the current network bandwidth.
2. A less conservative reaction to dropped packets may help to improve the Swift Start performance in a congested network.
3. With regard to the finding that all the ce_cwnd values with a gamma of 1 were overestimated, a gamma value of 4 or higher should be used in a heavy traffic environment (such as in the 4-Mbps background UDP traffic in the test).

Conclusions

The Swift Start algorithm was tested in single-flow and multiple-flow testbeds under the effects of high propagation delays, various bottlenecks, and small queue sizes. The conclusions are as follows:

For the capacity estimation feature, it was found that in the single-flow tests, the percentage of the majority of the estimated ce_cwnd values was acceptable. In addition, the fine granularity of the clock introduced a limit on the speed of the bottleneck link that could be tested. With the 1-ms granularity, it could not be tested for a bottleneck link rate higher than 23.168 Mbps (with a packet size of 1448 bytes) because the delta value would be smaller than 1 ms at these bottleneck links. Setting the clock granularity was tried at 0.5 ms, but right after pacing ended, the sender did not send out the packets in response to a few acknowledgments (ACKs) as soon as they arrived. Instead, the sender held off on these packets and sent them out in a burst when the next ACK came back.

For end-to-end performance based on the capacity estimation, in the single-flow tests the packets were sent faster at the beginning of Swift Start tests than in slow start tests. However, in the large-file-size tests, after some dropped packets occurred followed by a recovery period, the advantage of Swift Start early in the connection was diminished because of the additive increase and multiplicative decrease algorithm of transmission control protocol (TCP). The Swift Start throughput in this case did not show a significant benefit over regular TCP. On the other hand, Swift Start worked well in the tests with shorter file sizes when the buffer queue did not become overloaded. The average throughput of Swift Start in these small-file-size tests exceeded that of TCP in higher bandwidth-delay-product (BDP) scenarios, such as bottleneck link rates of 20 and 5 Mbps at a delay of 500 ms, and was slightly better or the same as the TCP throughput in the lower BDP cases. The result of the small-file-size tests was in line with the goal that Swift Start be designed to perform better under high BDP conditions as indicated in the previous report by BBN Technologies.

During the multiple-flow tests with the 2- and 4-Mbps user datagram protocol (UDP) background traffic inserted in the network, some ACK compression occurred, causing the ce_cwnd values to be overestimated. Also, with the small queue size of 5 slots, the delayed ACK timer expired before the th_ack_2 arrived in some tests, which then triggered the code to fall back to regular TCP handling. The result of the multiple-flow tests showed that there was no significant advantage in using Swift Start over TCP. When there was no capacity estimation due to the bursting behavior of slow start, TCP had dropped packets earlier in a connection as compared with the Swift Start pacing-only tests. However, there were more retransmitted packets in the Swift Start pacing-only tests than there were with TCP. Therefore, the Swift Start throughput was a little lower and about the same as TCP with 2- and 4-Mbps UDP background traffic, respectively.

In summary, because Swift Start was designed to attack the initial ramp-up problem of TCP, it held the promise of improving file transfers that do not overflow the network buffer (as in the 1-MB file transfers). However, the development of additional algorithms is needed to address the continuous estimation of bandwidth for long flows.

Appendix A
Glossary

ACK	acknowledgment
ATM	asynchronous transfer mode
BDP	bandwidth-delay product
BSD	Berkeley Software Distribution
CBR	constant bit rate
ce_cwnd	estimated congestion window for capacity estimation
delta	time interval between th_ack_1 and th_ack_2
gamma	configurable value used to obtain a fraction of estimated congestion window
LAN	local area network
RTT	round-trip time
SegSize	size of a segment
snd_cwnd	congestion window that determines the number of packets sent
SYN	synchronize sequence numbers flag
TCP	transmission control protocol
th_ack_1	ACK responses to first two packets of four initial packets
th_ack_2	ACK responses to last two packets of four initial packets
UBR	unspecified bit rate
UDP	user datagram protocol

Appendix B
Calculated Values of Congestion Control Window

The values of the congestion control window are calculated based on the settings of the bottleneck link rates and delays.

TABLE 6.—CALCULATED VALUES OF CONGESTION CONTROL WINDOW FOR EACH BOTTLENECK LINK

Test condition	Gamma		
	1	4	8
	Calculated value, bytes		
Bottleneck link rate, 20 Mbps Delay, ms			
500	1 250 000	312 500	156 250
100	250 000	62 500	31 250
Bottleneck link rate, 5 Mbps Delay, ms			
500	312 500	78 125	39 062.5
100	62 500	15 625	7 812.5
Bottleneck link rate, 0.5 Mbps Delay, ms			
500	31 250	7 812.5	3 906.25
100	6 250	1 562.5	781.25

References

1. Jacobson, V.: Congestion Avoidance and Control. Symposium Proceedings on Communications Architectures and Protocols, Stanford, CA, 1988, pp. 314–329.
2. Partridge, Craig, et al.: A Swifter Start for TCP. BBN Technical Report No. 8339, 2002.
3. Aggarwal, A.; Savage, S.; and Anderson, T.: Understanding the Performance of TCP Pacing. Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, 2000, pp. 1157–1165.
4. tcpdump/Libpcap. http://www.tcpdump.org/ Accessed Jan. 20, 2004.
5. Ostermann, Shawn: tcptrace. http://www.tcptrace.org/ Accessed Jan. 20, 2004.
6. xplot. http://www.xplot.org/ Accessed Jan. 20, 2004.
7. Mogul, Jeffrey C.: Observing TCP Dynamics in Real Networks. Conference Proceedings on Communications Architectures and Protocols, ACM Press, New York, NY, 1992, pp. 305–317.
8. Tirumala, Ajay, et al.: Iperf Version 1.7.0. http://dast.nlanr.net/Projects/iperf/ Accessed Jan. 20, 2004.
9. Rizzo, Luigi: dummynet. http://info.iet.unipi.it/~luigi/ip_dummynet Accessed Jan. 20, 2004.
10. Kelly, Tom: Scalable TCP: Improving Performance in Highspeed Wide Area Networks. Submitted for publication, 2002. http://www-lce.eng.cam.ac.uk/~ctk21/scalable Accessed Jan. 20, 2004.
11. Floyd, Sally: HighSpeed TCP for Large Congestion Windows. RFC 3649, 2003. http://www.icir.org/floyd/hstcp.html Accessed Jan. 20, 2004.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 2004	Technical Memorandum

4. TITLE AND SUBTITLE

Evaluation of Swift Start TCP in Long-Delay Environment

5. FUNDING NUMBERS

WBS–22–184–10–01

6. AUTHOR(S)

Frances J. Lawas-Grodek and Diepchi T. Tran

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, Ohio 44135–3191

8. PERFORMING ORGANIZATION REPORT NUMBER

E–14378

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546–0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA TM—2004-212938

11. SUPPLEMENTARY NOTES

Responsible person, Diepchi T. Tran, organization code 5610, 216–433–8861.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Categories: 17 and 99 Distribution: Nonstandard

Available electronically at http://gltrs.grc.nasa.gov

This publication is available from the NASA Center for AeroSpace Information, 301–621–0390.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

This report presents the test results of the Swift Start algorithm in single-flow and multiple-flow testbeds under the effects of high propagation delays, various slow bottlenecks, and small queue sizes. Although this algorithm estimates capacity and implements packet pacing, the findings were that in a heavily congested link, the Swift Start algorithm will not be applicable. The reason is that the bottleneck estimation is falsely influenced by timeouts induced by retransmissions and the expiration of delayed acknowledgment (ACK) timers, thus causing the modified Swift Start code to fall back to regular transmission control protocol (TCP).

14. SUBJECT TERMS

Transport protocol; Space communications

15. NUMBER OF PAGES

28

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	