CITI Technical Report 96–3

A Study of TCP/IP Performance on MVS

David H. Detlefs
`dhd@citi.umich.edu`

Ronald F. Zeilinger
`rfz@umich.edu`

ABSTRACT

In this study we compare the performance of MVS TCP/IP software from Interlink and IBM. We test the performance of three separate TCP/IP applications—telnet (3270), FTP, and UDP. Then, we compare the test results and discuss our observations and network, hardware, or software issues that may have affected the test results.

November 1996

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

A Study of TCP/IP Performance on MVS

David H. Detlefs and Ronald F. Zeilinger

November 1996

This document reports the results of MVS TCP/IP network performance tests conducted at the University of Michigan's Center for Information Technology Integration (CITI). The tests were sponsored by IBM Corporation, Network Systems Division.

The tests were conducted to compare the performance of TCP/IP software from two major MVS TCP/IP providers, IBM and Interlink. Each vendor supplied MVS TCP/IP software and a communication controller. Network performance testing was conducted using three separate TCP/IP applications—telnet (3270), FTP, and UDP.

1. Configuration

The Network Topology diagram in Figure 1 shows the configuration of the network over which testing was conducted. FTP and UDP performance testing was isolated in the test FDDI Ring, shown in the lower right corner of the diagram. The telnet 3270 tests, however, utilized RS/6000s clients at UM CITI (upper right) and traversed the UM Campus FDDI Backbone and two Ethernet's in addition to the test FDDI Ring.

The first primary hardware component was an ES/9000 Model 560 IBM Mainframe that was one of three dedicated CPUs using 64MB dedicated real memory, 512MB dedicated extended storage, and 3390 disk storage. The software included MVS OpenEdition 5.1 (running under VM/ESA 2.1), IBM TCP/IP 3.1, and Interlink SNS/TCP 2.0. The second primary test hardware component was an IBM 3172 Model 3 offload controller (3172-3 offload) that had a Pentium 60 motherboard, an IBM FDDI Base Adapter, and an IBM Parallel Channel Adapter. The software included OS/2 2.1, IBM TCP/IP 2.0, and OS/2 offload feature (level HL94200). The third primary hardware component was an Interlink 3762 Extended LAN Controller Model 3 (ELC-3) and FDDI adapter.

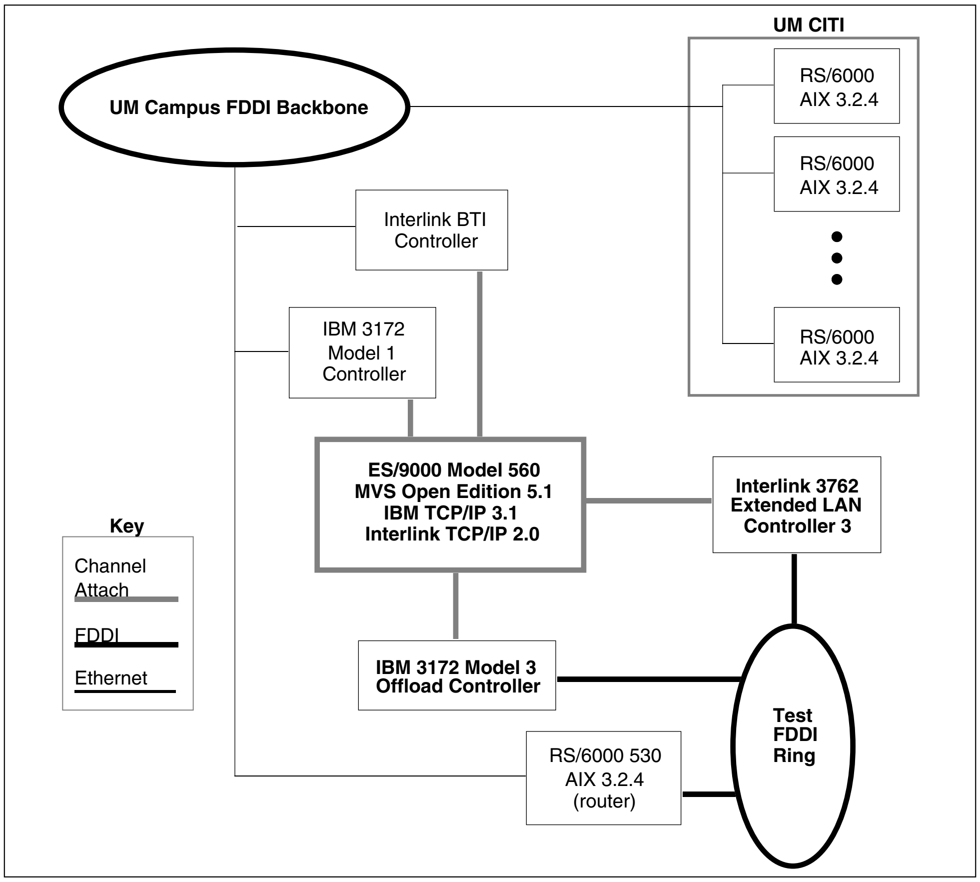

Figure 1. *Network Topology*

2. Telnet Server Performance

These performance tests were called “telnet server” tests because we wanted to determine how well a telnet-like server would perform on an MVS mainframe under light to heavy workloads.

To be more technically correct, we should have called them “telnet 3270 access to TSO” performance tests, because what we actually measured was the performance of the TCP/IP sub-system software used to provide network I/O service to the TSO/3270 facility on MVS. So that is what we mean by the term “telnet server” on MVS.

Our goal was to compare 1.) the mainframe resources (CPU, I/O, and memory) consumed by the TCP/IP software and 2.) the average response time provided to the telnet 3270 clients.

In order to generate a repeatable set of workloads (ranging from 100 to 2000 transactions per minute), we used multiple AIX workstations to execute a telnet 3270 script. Each script sent a standard request or command of 100 characters to MVS. This command was processed by a VTAM application (ITPECHO provided by IBM), which sent back an 800-character response. These lengths do not include telnet and 3270 control characters. This script represented a typical telnet session that could be executed concurrently to generate a telnet server workload representing up to 1000 client users.

The client side of the transaction was driven by a telnet 3270 script, customized to drive the transaction on both IBM and Interlink host software. This script was run under an AIX-based transaction generator software package supplied by IBM. The clients were all IBM RS/6000s with models ranging from POWERstation 320Hs to POWERstation 530s. The operating system running on the IBM POWERstations was AIX V3.2.4.

Performance statistics were gathered from SMF and RMF measurements on MVS and via facilities built into the transaction generator on the telnet clients. Statistics of primary interest on MVS were CPU utilization and service rate. Transaction response time was measured by the AIX clients.

Tests were conducted by varying the transaction rate over 1, 1.5, and 2 transactions per minute, and varying the number of telnet sessions over 100, 250, 500, and 1000 simultaneous sessions. The goal was to collect twelve samples for each vendor (three rates over four session values).

2.1 Telnet Test Procedures and Issues

Generating consistent test results proved to be the most difficult part of this study. The primary challenges to consistent testing were related to the telnet client software, the network environment, and the capacity and setup of the AIX telnet slaves. A more detailed description of each challenge follows.

2.1.1 AIX Telnet Client Software

The telnet client test software provided by IBM was originally designed to run on AIX for PS/2, and required some modifications to run in the CITI RS/6000 environment. The telnet client software was configured as a master scheduler and multiple slave machines. Commands were issued to the master scheduler, which passed each command across the TCP/IP network to designated slave machines. The primary commands used in running telnet sessions were “logon” and “start”.

When the “logon” command is passed to a slave, the slave forks a new process which serves as a telnet client. Each telnet client runs as a separate process and uses approximately 500K of spool space. The client invokes a customized script to establish a telnet session with the MVS host. The script consists of three functionally distinct sections: host logon, workload in a steady state, and host logoff.

When the “start” command is passed to a slave, the slave invokes a standard transaction by running the corresponding section of the script.

The scripting language works by sending a fixed string of telnet 3270 characters and waiting for either a fixed return string or a fixed number of characters. If the expected return string never arrives, the telnet session remains frozen. For the majority of sessions and transactions, the scripting language was adequate to complete the test. In some instances, however, environmental factors affected the timing of the telnet session and unexpected response strings were returned to the client. This seemed to happen most frequently during the high stress testing, when both the MVS host and RS/6000 client machines were busiest. As a result, the stated goal of testing with 100, 250, 500, and 1000 simultaneous sessions was degraded under the heavier workloads where as many as 8% of the sessions failed prior to test completion. A suggested improvement to the scripting language would be to have it accept multiple return strings, enabling recovery from error conditions.

In the tests involving 1000 sessions, the AIX clients were loaded with an average of 40 sessions each. Because each session is run in a separate process, some overhead is incurred when switching between the 40 processes. For some slaves, spool space was the limiting factor in determining how many telnet sessions could be run. An improved test design might be to use one process to handle all telnet sessions for a slave, better utilizing the hardware resources, increasing the session capacity of the slave, and reducing telnet response times.

2.1.2 Network Environment for Telnet tests

As Figure 1 shows, the RS/6000 telnet slave machines are located at UM CITI, which is several miles from the MVS host. To reach the host, two Ethernets, two FDDI Rings, and three routers must be traversed. The telnet tests were run in the evening and early morning hours to minimize interference from extraneous network traffic. Nonetheless, the network is public, so bursts of network activity may have affected the results of some telnet tests.

2.1.3 Capacity and Setup of AIX Telnet Slaves

The telnet slave machines running the telnet client sessions were all located at CITI, belonging to members of the CITI staff. Although CITI uses a standard “scrub” procedure to set up each RS/6000 with the AIX operating system, over time some machines had been modified in subtle and not so subtle ways. The result was that of over 40 machines available for telnet testing, only 26 were able to serve as telnet slave machines. As noted earlier, spool space became a problem on some of the machines, because each telnet session needed about 500K of spool space, or 20MB for 40 telnet sessions. Most of the machines were already running AFS and DFS client software, although activity was minimal at the time testing was conducted.

2.2 Telnet Test Results

The statistics used to analyze telnet server performance are MVS CPU utilization, MVS service rate, and telnet client transaction response time. The telnet tests were conducted by varying the transaction rate over 1, 1.5, and 2 transactions per minute and varying the number of telnet sessions over 100, 250, 500, and 1000 simultaneous sessions. The number of sessions shown on the x-axis in Figure 2 are “nominal values” because we did not quite achieve a 100% survival rate for the entire duration of the higher stress test. (The actual number of active sessions may be as much as 8% less.) At low CPU utilization levels several apparent anomalies appear in the data. Within a session group (100, 250, 500, or 1000 sessions), one would expect CPU utilization to increase with the transaction rate. Because MVS background processing often uses about 5% of the available CPU cycles, tests in the 100 and 250 session groups were more susceptible to variations in MVS background processing, in addition to the other environmental factors mentioned earlier.

2.2.1 Telnet CPU Utilization

Based on the MVS telnet CPU utilization data shown in Figure 2, for 100 simultaneous telnet sessions IBM telnet uses slightly less CPU than the Interlink telnet. For 250, 500, and 1000 simultaneous sessions, Interlink telnet uses slightly less CPU than IBM telnet. At the highest transaction rate (2 transactions per minute) and with a high number of users (500 and 1000), the difference in CPU utilization is most pronounced.

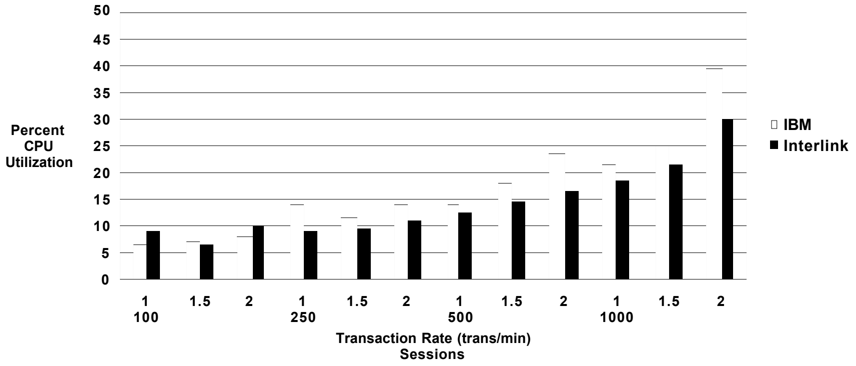

Figure 2. *MVS Telnet CPU Utilization*

2.2.1.1 Interpolated CPU Utilization

In order to plot how CPU utilization increases as the telnet workload or number of concurrent sessions increases, we needed to correct for any sessions that may have failed. So, the number of sessions plotted across the x-axis in Figure 3 are not the same as the 250, 500, and 1,000 session-scripts that were started. The number of actual sessions varies among the samples with over 100 sessions, so some data points represent interpolated values. Although sparse, the data suggests that CPU utilization may increase linearly with the number of sessions for a given transaction rate. Interpolated values are denoted with shaded cells in Table 1.

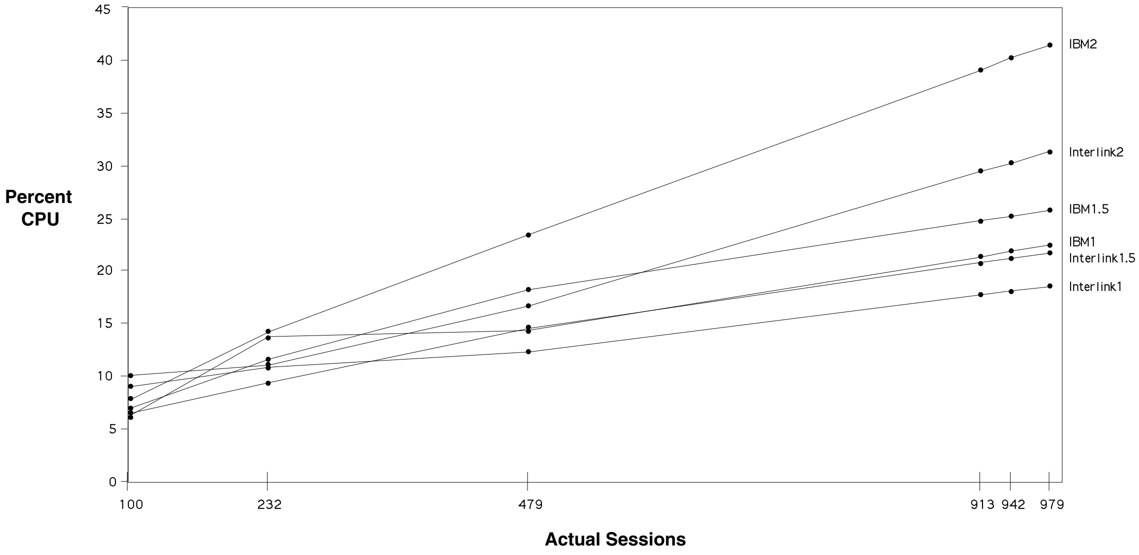

Figure 3. *MVS Telnet CPU Utilization (with interpolated values)*

TABLE 1. MVS Telnet CPU Utilization (with interpolated values)

	Sessions					
Series	**100**	**232**	**479**	**913**	**942**	**979**
IBM1	6.6	14.0	14.3	21.2	21.7	22.3
IBM1.5	7.2	11.6	18.1	24.7	25.1	25.7
IBM2	8.2	14.3	23.6	39.3	40.3	41.7
Interlink1	9.1	10.8	12.4	17.8	18.1	18.6
Interlink1.5	6.5	9.4	14.5	20.8	21.3	21.8
Interlink2	10.0	11.0	16.7	29.3	30.1	31.2

2.3 Telnet Service Rate

A more comprehensive measurement of resources consumed by the ES/9000 mainframe is provided by the "service rate," which includes I/O activity and memory usage as well as CPU utilization. The data in Figure 4 shows that for under 500 sessions the service rate for both providers is about equal. At 500 sessions, the service rate for Interlink is slightly higher than for IBM, and for 1000 sessions, the Interlink service rate is significantly higher.

This observation raises an interesting question: At the highest stress levels the telnet tests indicate that Interlink uses less CPU than IBM, but more resources, so what are the additional resources that Interlink uses? One possible explanation involves the manner in which I/O activity is measured. The IBM CLAW channel protocol, connecting the host with the IBM 3172-3 offload controller, is reputed to run as a continuous channel program on the host. This is measured as a single I/O operation, which is consistent with an extremely low I/O exhibited by the channel servicing the 3172-3 offload controller.

The *MVS/ESA Initialization and Tuning Guide* specifies that the following formula is used to calculate the service rate:

```
Service = (CPU * task execution service) + (SRB * service req blk service)
            + (IOC * I/O activity) + (MSO * memory occupancy)
```

Our coefficients used to measure service rate were CPU=10.0, IOC=10.0, MSO=3.0, and SRB=10.0. This means that I/O activity should be weighted to be about 30 percent of the service rate value. However, the SMF/RMF measurement values are reduced by the continuous loop CLAW channel program, and the service rates are lower when I/O counts are eliminated. The service rate metric should therefore be viewed as an interesting phenomenon, rather than concrete evidence of performance.

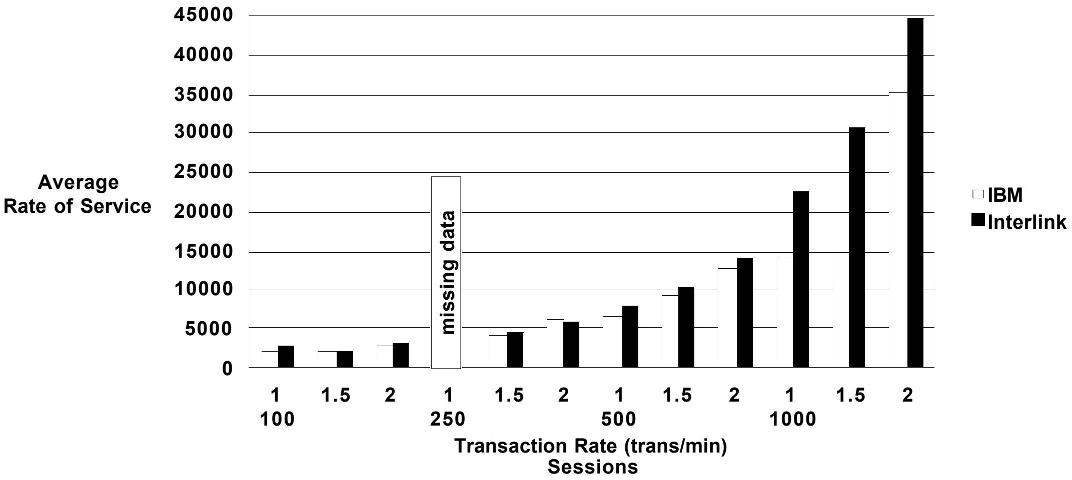

Figure 4. *MVS Telnet Service Rate*

2.4 Telnet Client Response

Telnet client response times were measured on the AIX clients in hundredths of a second. Response times (see Figure 5) seem to be approximately equal for both servers, up to the 500 session, 2 transactions/minute data. At this point, response times increase for both vendors, with IBM telnet appearing to be more sensitive to increases in the number of sessions and Interlink being more sensitive to the transaction rate. At the highest telnet stress levels, response times for IBM telnet are slightly better than Interlink telnet. Note that the 250 session, 1 transaction/minute data have been left out due to missing data.

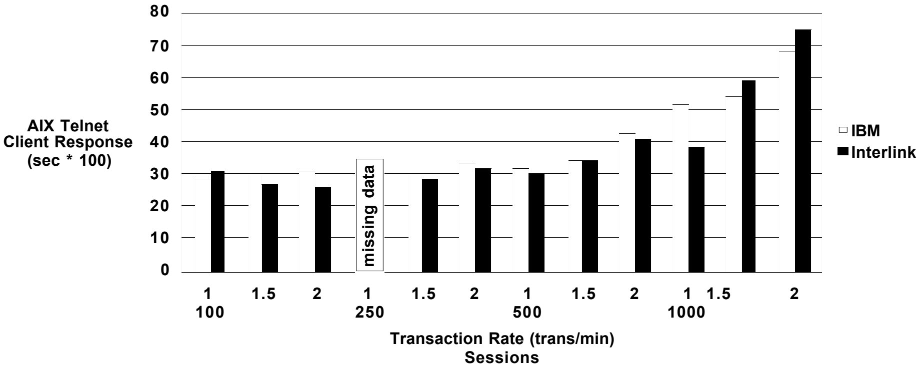

Figure 5. *MVS Telnet/AIX Client Response*

3. FTP Server Performance

Most of the FTP benchmarks were conducted on the dedicated test FDDI Ring, as shown in Figure 1. For purposes of comparison with the FDDI-based tests, a smaller number of FTP benchmarks were conducted using a local Ethernet.

The FTP tests measured three FTP performance metrics:

- FTP Server Average Throughput—The average data transfer rate, in Kbytes per second, for an individual session.
- FTP Server Total Throughput—The total throughput for all simultaneous FTP sessions running in a given test.
- MVS Average CPU Utilization—The average time the CPU is busy during the FTP test. For tests with simultaneous FTP sessions, the sample is taken while all sessions are busy.

3.1 FTP Test Procedure

Here is a brief outline of the procedure used to run an MVS FTP server performance test:

1. On MVS, shutdown all extraneous programs and start SMF/RMF recording.
2. On the AIX workstation, start one to eight simultaneous FTP sessions, automated by a ".netrc" file. Each session is logged to a separate file.
3. For a given FTP test, collect samples by varying the FTP file type between ASCII and BINARY, the FTP operation between "get" and "put", the number of simultaneous sessions between one and eight, and the FTP server on the mainframe between IBM and Interlink. This procedure results in 64 samples per FTP test (2 x 2 x 8 x 2).
4. After all samples have been run, the AIX-based log files are parsed and crunched with several Perl scripts. Each FTP session gets or puts the same file multiple times, and the scripts are designed to throw out the first and last FTP measurements to ensure that all simultaneous sessions are running during the sample period. Eight FTP throughput reports are created by this process, four for average throughput and four for total throughput.
5. On MVS, the RMF summary report is run, giving CPU utilization values at one minute intervals. The summary report is transferred to the AIX workstation and processed by another Perl script, creating an average CPU utilization report.

3.2 Average Throughput versus Server Buffer Size

To study the effects of the server buffer size on FTP performance for both vendors, the buffer size was varied between low and high values. The get BINARY test is representative of the buffer size tests. Figure 6 shows that the IBM MVS FTP server gains a performance advantage by using the maximum allowable buffer size (28K) for the 3272-3 offload controller.

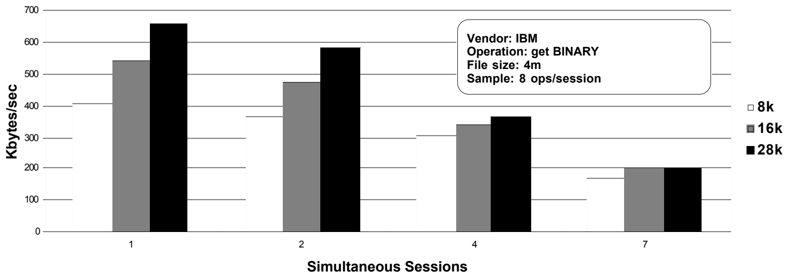

Figure 6. *IBM FTP Server Throughput (versus Buffer Size)*

Figure 7 shows that the Interlink software appears to perform best with a smaller buffer, but performance is relatively the same whether the buffer size is smaller or larger. Although the Interlink software manual indicated that a buffer size up to 64K can be used, our FTP tests failed at values above 16K.

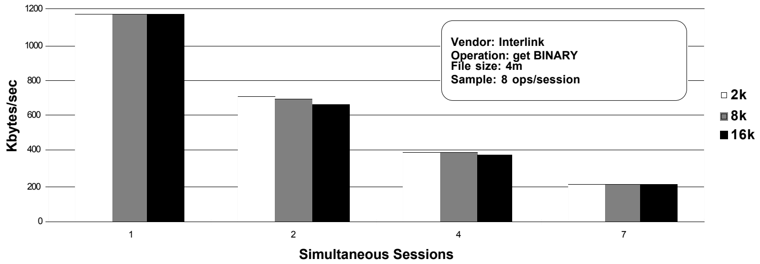

Figure 7. *Interlink FTP Server Throughput (versus Buffer Size)*

3.3 FDDI FTP Test Results

The first FTP test was conducted to determine the effect of server buffer size on FTP throughput. After that test, the parameters in Table 2 were used for the remainder of the FDDI-based FTP tests. The IBM server buffer size of 28672 represents a maximum that can be used with the 3172-3 offload controller.

TABLE 2. *FDDI FTP Test Parameters*

	Interlink	**IBM**
FTP Server	Interlink FTP Server	IBM Pascal FTP Server
Region Size	8MB	8MB
Server Buffer Size	16 * 2048 = 32768	28672 (3172-3 maximum)
Client Buffer Size	32K (send and receive)	32K (send and receive)
Network Packet Size	4352 (FDDI maximum)	4352 (FDDI maximum)
Block Size	27968 (1/2 track,3390)	27968 (1/2 track,3390)
Record Length	64	64

3.3.1 FTP Server Average Throughput (FDDI)

The graphs in Figure 8 compare average session FTP throughput for get/put and ASCII/BINARY variations. The general trend of these graphs is that the average throughput for the Interlink FTP server is 20 to 50 percent faster than the IBM FTP server for the single session case, but as the number of sessions increases, Interlink's advantage diminishes. For the put BINARY case, IBM performs better than Interlink, by a small margin, for the three and the four simultaneous sessions cases. In general, Interlink FTP performs ASCII file transfers better than IBM FTP, while IBM FTP performance is close to Interlink's for multiple session binary file transfers.

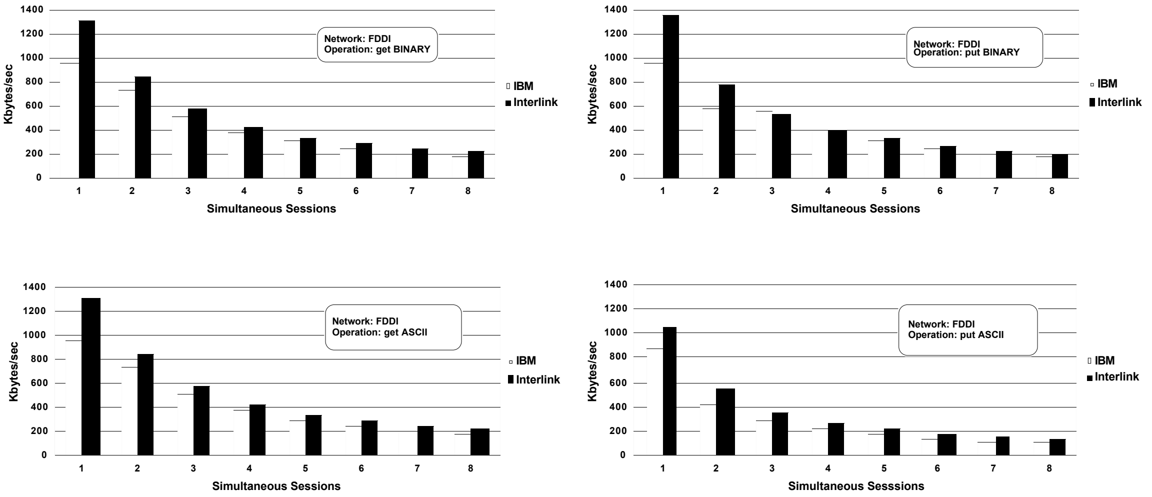

Figure 8. *FTP Server Average Throughput*

3.3.2 FTP Server Total Throughput (FDDI)

The total throughput metric is a gross measure of FTP capacity across all concurrent FTP sessions. The trend in total FTP throughput (see Figure 9), for both vendors, is to reach maximum capacity by three sessions and maintain that capacity for an increasing number of sessions. Performance for put ASCII operations remains about the same for all sessions. One might surmise that put ASCII file transfers are limited by the speed at which the transferred characters can be converted from ASCII to EBCDIC. Relative performance values of IBM and Interlink reflect those observed in the average throughput FTP measurements.

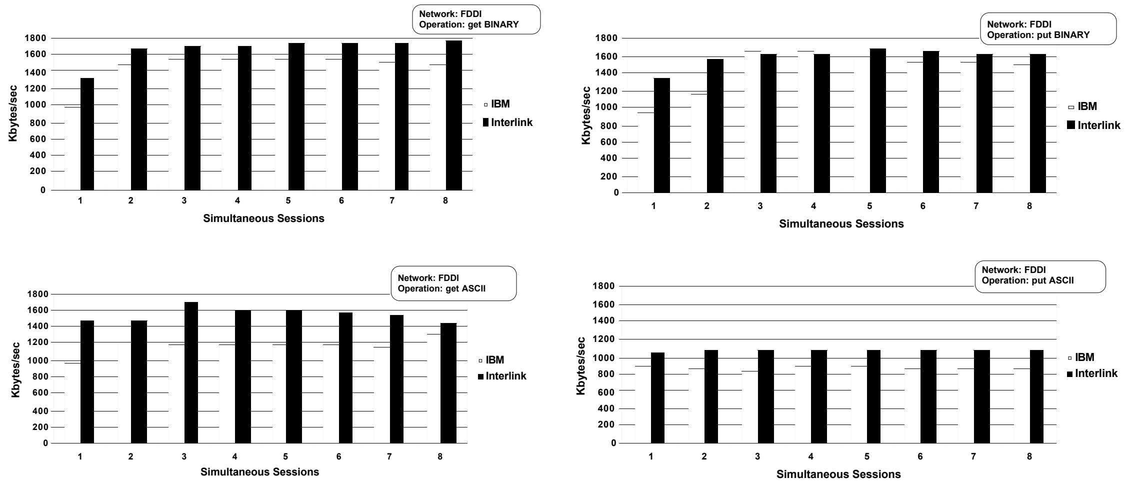

Figure 9. *FTP Server Total Throughput*

3.3.3 FTP Server Average CPU Utilization (FDDI)

Measurements of average CPU utilization (see Figure 10) show that IBM uses less CPU than Interlink during get ASCII operations. The put ASCII graph shows close CPU utilization values between the two vendors. The get/put BINARY graphs indicate that Interlink uses significantly less CPU than IBM.

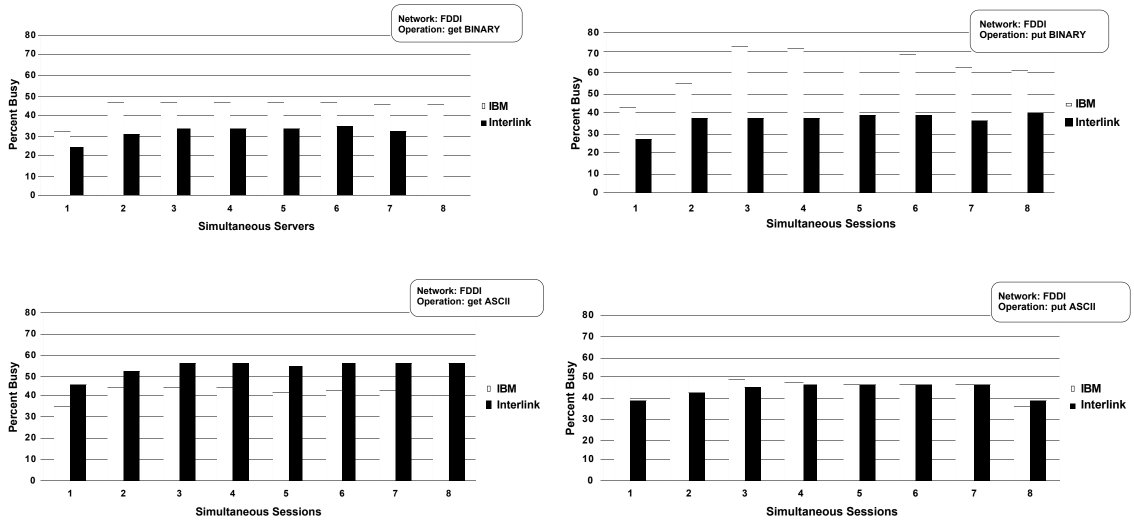

Figure 10. *FTP Server Average CPU Utilization*

3.4 Ethernet FTP Test Results

The setup parameters shown in Table 3 were used in the Ethernet FTP tests and are similar to the FDDI FTP tests, with the exception that network packet size was reduced to 1500 bytes due to router limitations.

TABLE 3. *Ethernet FTP Test Parameters*

	Interlink	**IBM**
FTP Server	Interlink FTP Server	IBM Pascal FTP Server
Region Size	8MB	64MB
Server Buffer Size	16 * 2048 = 32768	28672 (3172-3 maximum)
Client Buffer Size	32k (send and receive)	32k (send and receive)
Network Packet Size	1500 (Ethernet maximum)	1500 (Ethernet maximum)
Block Size	27968 (1/2 track,3390)	27968 (1/2 track,3390)
Record Length	64	64
Ethernet controller	BTI/ELC-2 (8232)	3172 Model 3 offload

Because the Interlink ELC-3 controller used in FDDI testing was not configured with an Ethernet board, we chose to use an older Bus-Tech, Inc. controller. We used the BTI/ELC-2 (8232) in place of the ELC-3 to conduct the Ethernet portion of the FTP benchmark. The Ethernet was locally attached to both controllers, as shown in Figure 1. The motivation for the Ethernet FTP tests was not to produce a definitive benchmark for Ethernet performance, but rather to gauge relative performance of the FTP servers and to provide a rough estimate of the benefits of using FDDI- versus Ethernet-based FTP connections. Due to time limitations, only binary files were used in the Ethernet FTP benchmarks.

3.4.1 FTP Server Average Throughput (Ethernet)

The test results for average Ethernet FTP throughput (see Figure 11) show that Interlink performs better than IBM for the get BINARY test. The put BINARY test results are very close, with Interlink providing slightly better throughput. In both cases, get and put, the difference in performance is greatest for the single session case, and diminishes as the number of sessions increases.

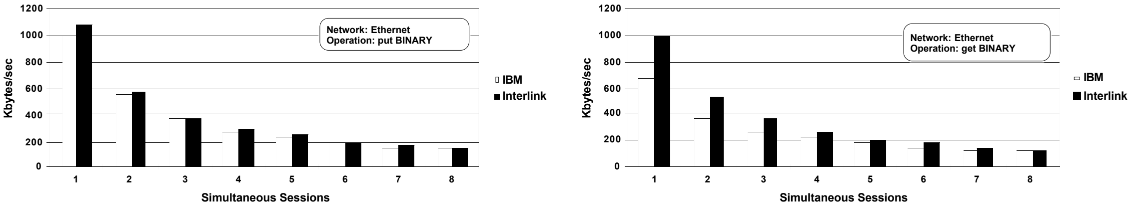

Figure 11. *FTP Server Average Throughput*

3.4.2 FTP Server Total Throughput (Ethernet)

The average Ethernet FTP throughput (Figure 11) multiplied by the number of simultaneous sessions is approximately equal to the total Ethernet FTP throughput (Figure 12). For the get BINARY test, Interlink achieves near maximum total throughput at three simultaneous sessions, whereas IBM total throughput reaches a maximum throughput at seven sessions. For the put BINARY test, both vendors achieve near maximum throughput at only two simultaneous sessions.

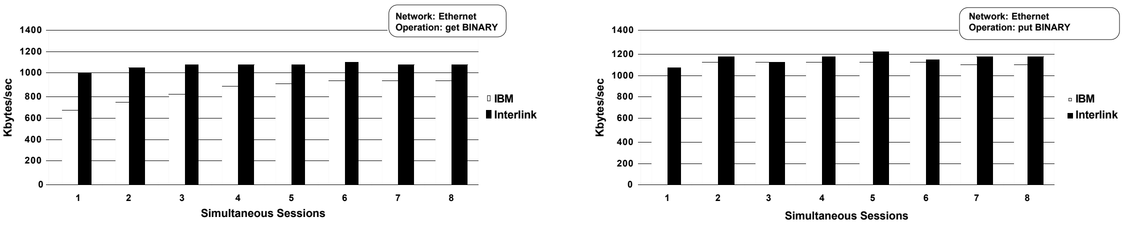

Figure 12. *FTP Server Total Throughput*

3.4.3 FTP Server Average CPU Utilization (Ethernet)

The CPU utilization measurements (see Figure 13) show that IBM uses less CPU for get BINARY, but more CPU for put BINARY. The CPU utilization test results suggest that using the 3172-3 offload controller may involve a trade-off between FTP throughput and CPU utilization.

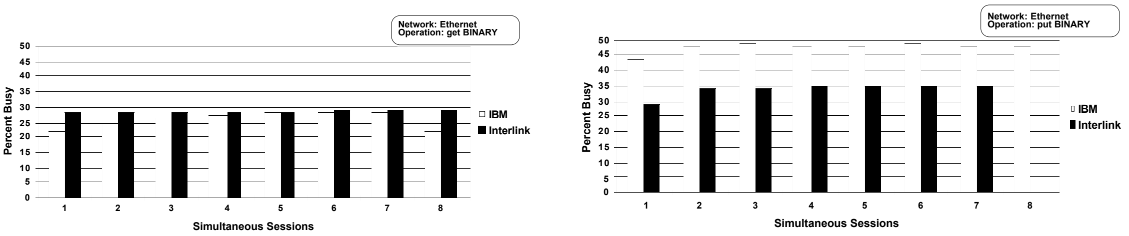

Figure 13. *FTP Server CPU Utilization*

4. UDP Socket Performance

The UDP benchmark consists of seven sets of tests, two tests running over local Ethernet and the remaining five running over an isolated FDDI ring. The UDP benchmark attempts to measure single session throughput over both Ethernet and FDDI networks, while providing a detailed look at CPU utilization on the MVS mainframe. Because the main focus of these performance tests concentrated on high-speed FDDI data transfer, and because we have conducted extensive Ethernet-based UDP tests in the past, we only ran two tests over Ethernet, just for comparative purposes.

A locally attached IBM RS/6000 workstation served as the remote node to the UDP socket application on the MVS mainframe. The udptest program, obtained from IBM, was modified to provide pacing for the UDP packets. This proved to be necessary for UDP testing because the unpaced RS/6000 was able to supply UDP packets faster than MVS TCP/IP (both IBM and Interlink) could accept them. The next section, UDP Testing Issues, provides a more detailed discussion of this topic.

TABLE 4. *UDP Test Environment*

Item	Description
Hardware	IBM ES/9000 IBM 3172-3 offload controller • IBM Parallel Channel Adapter • IBM FDDI Base Adapter • IBM EtherStreamer Adapter IBM 3172-1 Controller Interlink 3762 Extended LAN Controller 3 • Channel Adapter • FDDI Adapter IBM RS/6000, Model 530
Software	MVS OpenEdition 5.1 (ES/9000) IBM MVS TCP/IP 5735-HAL V3.1 (ES/9000) IBM MVS OpenEdition sockets (ES/9000) OS/2 2.1, OS/2 TCP/IP 2.1, offload feature (3172-3) Interlink SNS/TCPaccess V2.0 (ES/9000) AIX V3.2.5 (RS/6000)
Network	Local, subnet Ethernet, MTU=1500 Local, dedicated FDDI, MTU=4354
Test Program	udptest, by Mike Nieme, Watson Research Labs. Modified by Ron Zeilinger, University of Michigan

4.1 UDP Testing Issues

Although the socket application programming interface looks very similar for both UDP and TCP sockets, the two protocols are quite different. TCP is connection-oriented and will establish a connection to transmit a file or stream of records before it disconnects. UDP is connectionless and only transmits a single datagram, so there is no need to establish a connection. TCP is reliable because TCP/IP ensures that the data stream is in sequence and that the checksums are valid. UDP is more efficient because datagrams do not have to be in sequence and there are no checksums so TCP/IP has to do less work. TCP uses windowing to provide flow control so data does not get lost. UDP has no flow control and has no way to detect if any datagrams got lost. The lack of flow control becomes a serious problem for UDP socket applications under two conditions: the amount of data to be transferred becomes larger than one megabyte and the processor speed cannot keep up with network speed (FDDI).

We found that we could not transmit more than a megabyte to MVS nor could MVS receive at a sustained rate faster than 1 MB/sec without losing up to half the datagrams. To help run the Ethernet testing programs across the FDDI subnet we used a program called udptest, which had been used to test the HAL 2.2.1 bulkmode option suitability for the RX protocol supporting the Institutional File Server on MVS at the UM. The RX protocol had used a window of 15 datagrams with a request for acknowledgment every fourth datagram. If an ACK was not received before the 2-second timeout, the sending site would re-send all datagrams in the current window. This protocol did provide reliable delivery over FDDI so we used it for our UDP tests, transmitting 10MB in 2,431 UDP datagrams (4312 bytes) and responding with 608 ACKs (78 bytes).

All of the UDP tests were run using the udptest program that was written in C and designed to take the following statistics:

- The CPU utilization is extracted from the TCP/IP address space. The Task Control Block (TCB) time is extracted from the ASCBEJST field, and the SRB-time is extracted from the ASCBSRBT field of TCP/IP's Address Space Control Block.
- The CPU utilization of the udptest socket application address space is extracted from the application's ASCB using the technique described above.
- An accurate elapsed time is determined by using the STCK instruction to store the 64-bit time-of-day (TOD) clock and computing the difference. All CPU and TOD times provide nanosecond accuracy. The 64-bit values are converted to double-precision floating-point values scaled as seconds and printable as "%.6f" values in seconds.
- The data transfer rate is calculated by dividing the total number of user data bytes (ignores 40-byte headers on UDP datagrams) by the elapsed time.
- The datagram transfer rate is calculated by dividing the number of UDP datagrams transferred by the elapsed time.
- The overall machine CPU busy time is calculated by subtracting the CPU wait time from the elapsed time. CPU wait time is extracted from the LCCAWTIM field in the Logical Configuration Communication Area. This technique was copied from the ISFSRC(ISFDA) source code used by System Display and Search Facility (SDSF) to show CPU% busy on the top line of the Display Active Users (DA) panel.
- The percentage of CPU utilization for the entire machine and for the two address spaces (TCP/IP and udptest socket application) is extracted using the technique mentioned above.
- The MIPS instructions/sec rate for this model of CPU is extracted from the PCCARSPR field of the Physical Configuration Communication Area. However, we state our path-lengths in CPU_microseconds instead.

Our udptest program does not use TCP connection-oriented sockets like telnet and FTP. So we could not expect any correlation with the results of our telnet and FTP performance tests. udptest doesn't read/write any data to/from disk so its performance is not affected by disk I/O. udptest only operates over UDP connectionless sockets that send and receive individual datagrams without ever establishing a connection to a remote node. udptest contains both a sending client mode and a receiving server mode so the same code can interoperate as a client/server pair across the network between two separate machines. There are compile time (preprocessor) variables to specify:

- Operating system: (default = AIX), MVS, or VM
- Compiler: (default = C370), or SASC
- Socket library: (default = BSD), HAL, FAL, SNS, or OPEN_SOCKETS defines MVS OpenEdition 5.1 Integrated Sockets

We only needed one copy of udptest on our AIX workstation, but on the MVS mainframe we had three different versions—HAL, SNS, and OPEN_SOCKETS using three different socket libraries. Our test results compare the MVS performance of the HAL, SNS, and OPEN_SOCKETS libraries.

4.2 UDP Test Procedure

Here is a quick overview of how we ran a udptest test.

1. Start up the receiving server, which must be ready to receive before the client starts to send. Specify size of user data in datagram: 4312 max. for FDDI, 1460 max. for Ethernet. Select an ACK protocol for reliable delivery that uses a window of 15 datagrams and will acknowledge every fourth datagram.
2. Start up the sending client on the other node. Specify the same size of datagram and reliable ACK protocol as used by the server. Also specify server's network address and amount of data, 10MB, to be transferred.
3. The sending client will send datagram #0 to start measuring on the server and also makes its own initial resource usage measurement. Then the client will send out a burst of datagrams until the window is full.
4. In the meantime, the receiving server loops on non-blocking recvfrom() to read the datagrams as fast as they arrive. When every fourth datagram arrives, it responds by calling sendto() to send an ACK back to the client.
5. Only when the window is full, will the sending client stop to read the ACK datagrams and slide the window past all acknowledged datagrams. If no ACK datagram is received before a 2-second timeout occurs, the entire window (all 15 datagrams) are retransmitted.
6. Finally, when all of the data has been sent, the sending client will make its final resource usage measurement to determine the actual resource usage that is encapsulated in the terminating or signoff datagram which is sent to the receiving server.
7. When the client's signoff datagram is received by the server, it signals the end of the test. It makes the final resource usage measurement to determine the actual resource usage. This information, combined with the client's performance information, is used to prepare the server's performance report which is similar to the client's report but more complete because it contains performance information of both client and server.
8. After all data transfer is completed, the client calculates and prints the client's performance report containing:
    - Number of UDP datagrams sent and total number of bytes sent
    - Number of ACK datagrams received and avg.round-trip-time
    - Elapsed time in microseconds, and CPU idle and CPU busy time
    - TCB/SRB time in microseconds used by client
    - TCB/SRB time in microseconds used by TCP/IP
    - Path-length in CPU_usec/byte and CPU_usec/datagram
    - Data transfer rate in KBytes/sec and datagrams/sec

9. Then the client calls shutdown() and close() to delete the socket.

4.3 UDP Test Results

Table 5 summarizes the seven UDP test cases. Note that the IBM 3172-1, not the 3172-3 offload controller, was used for the first two Ethernet tests. The remaining five tests measure UDP performance over FDDI. Three different socket runtime environments were measured: HAL V3.1 Sockets in Tests #1–#4, OpenEdition 5.1 Open Sockets in Test#5, and Interlink's SNS V2.0 Sockets in Tests #6 and #7. The advantage of using the bulkmode or socket queuing option was measured in Tests #2, #4, and #7.

TABLE 5. *UDP Test Cases*

Test	Runtime Environment & Network Medium
Test#1	IBM 5735-HAL V3.1 Sockets (3172-1, Ethernet)
Test#2	IBM 5735-HAL V3.1 Sockets (3172-1, Ethernet, bulkmode)
Test#3	IBM 5735-HAL V3.1 Sockets (3172-3 offload, FDDI)
Test#4	IBM 5735-HAL V3.1 Sockets (3172-3 offload, FDDI, bulkmode)
Test#5	IBM MVS/OpenEdition 5.1 Open Sockets (3172-3 offload, FDDI)
Test#6	Interlink SNS/TCP V2.0 Sockets (3762/ELC-3, FDDI)
Test#7	Interlink SNS/TCP V2.0 Sockets (3762/ELC-3, FDDI, queuing)

Send and receive operations are from the perspective of MVS. Refer to the sections UDP Testing Issues and UDP Test Procedure, for further details.

4.3.1 CPU Usage/byte for UDP

Figure 14 displays CPU utilization as a measure of microseconds of CPU time used to send or receive one byte of data with the udptest program. In all cases except Test#4, sending data from MVS uses more CPU cycles than receiving to MVS. Whenever the sending client stops to read an ACK datagram, it invokes CPU-intensive select() function to wait for the next incoming ACK. So we can expect the sender to use more CPU. In Tests #2 and #4, using bulkmode reduces CPU usage for both send and receive, with the send operation receiving the greatest benefit. The Interlink CPU usage (Tests #6 and #7) is lower in all cases than any of the IBM-based UDP tests, for both send and receive. Using queuing with Interlink sockets did not reduce CPU usage. Of the IBM tests, Test#5 used the least CPU, probably because the new OpenEdition sockets use a direct PC call rather than emulating the IUCV transfer mechanism into the TCP/IP address space, which saves CPU cycles.

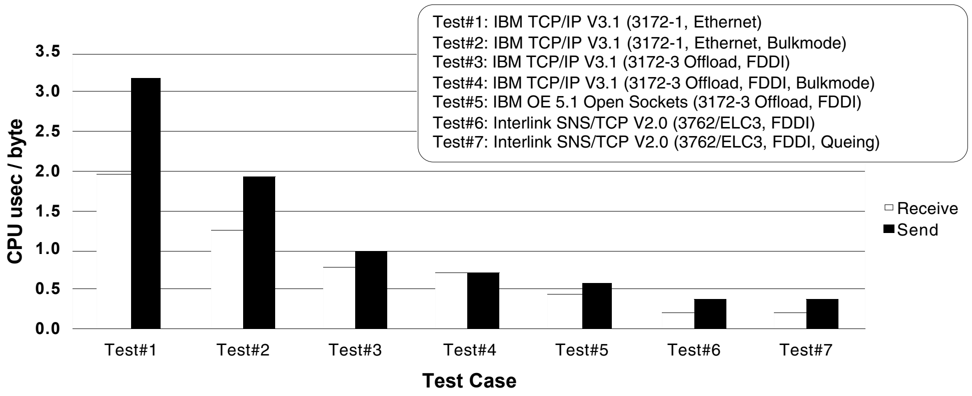

Figure 14. *UDP Pathlengths on MVS*

4.3.2 Detailed CPU Usage for UDP

Figures 15 and 16 show how CPU usage is split between the socket applications program, udptest, and TCP/IP. The udptest application was linked with three different runtime socket libraries, thus the CPU usage for udptest not only reflects the performance of the program application code but also includes CPU cycles spent in the runtime library routines. This may help to explain the correspondence between TCP/IP CPU usage and udptest CPU usage. Another observation is that for the two Interlink test cases, the udptest application CPU usage is either equal to (for receive) or greater than (for send) the CPU usage of Interlink SNS/TCP.

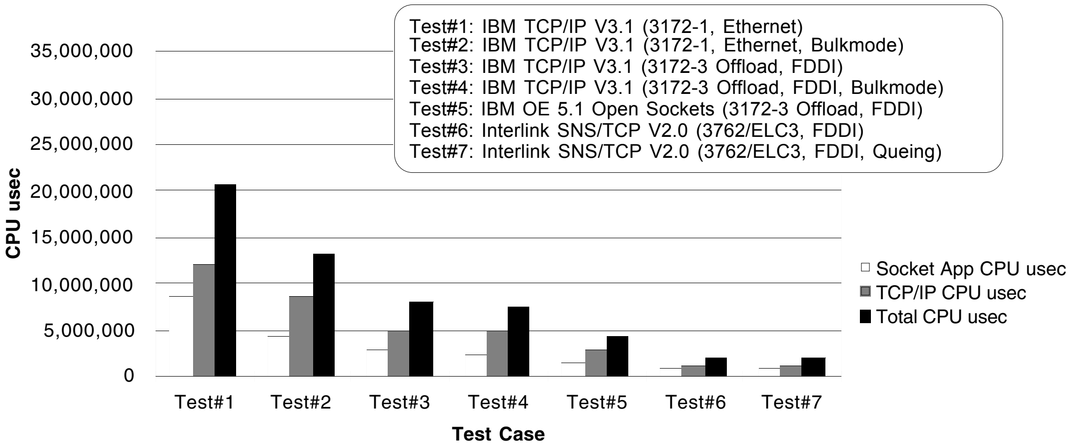

Figure 15. *UDP Receive CPU Usage*

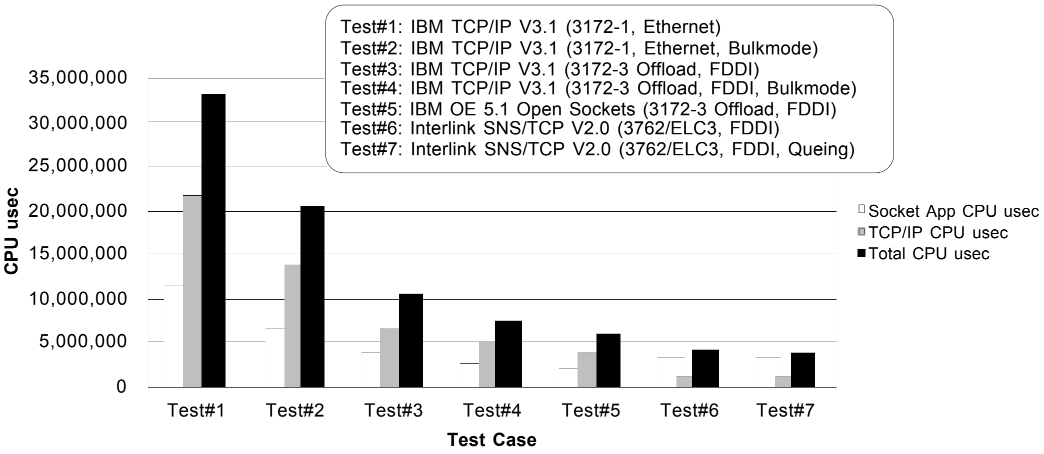

Figure 16. *UDP Send CPU Usage*

4.3.3 Network Throughput for UDP

The UDP throughput chart shows that FDDI does indeed provide faster data transfer rates than Ethernet. For FDDI, most send and receive times are nominally equivalent. Based on these tests, Interlink UDP is faster than IBM UDP. Also, IBM bulkmode and Interlink socket queuing have little discernible effect on performance; however, the pacing methodology used in the udptest program may have had an adverse effect on these options. Refer to the UDP Testing Issues section for an explanation of this possibility. Finally, OpenEdition Open Sockets is nominally equivalent to IBM TCP/IP V3.1 in UDP throughput.

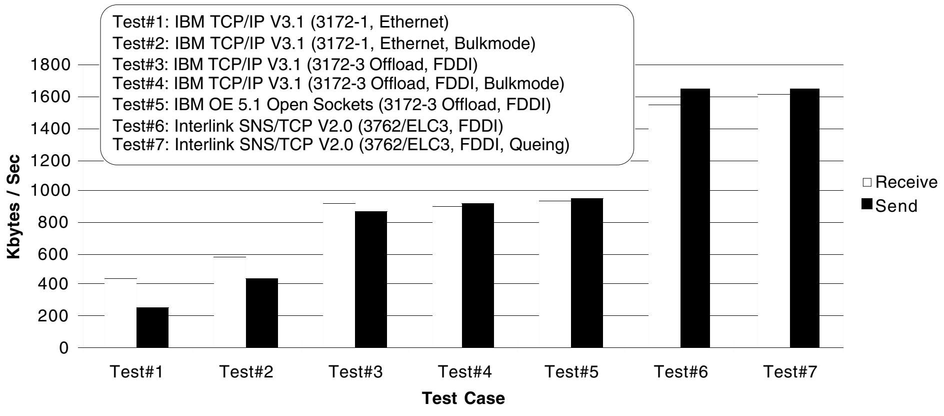

Figure 17. *UDP Throughput*

5. Non-performance Considerations

This section contains miscellaneous observations and impressions gathered while preparing for and conducting the TCP/IP performance tests. While fairness is a goal, no formal evaluation procedure was used in preparing the comments dealing with non-performance issues.

5.1 Installation

The MVS software for both products was installed using the SMPE software installation tool, which provides a proven installation and maintenance methodology. Because we may have been the first to install SNS/TCPaccess 2.0 into an MVS 5.1 environment, we encountered incompatibilities between the “Binder” and SNS load modules.

With the IBM TCP/IP product, we had to choose between two FTP servers: a Pascal-based server or the newer C-based server. We chose the Pascal FTP server. In the future, we hope that IBM makes the transition to a single FTP server, obviating the need to make a server choice.

Both products require parameters to be configured. In many situations, default values may suffice; however, we found that for some testing, especially telnet, some parameter values have to be increased to accommodate the heavy test load. Although a greater flexibility can be achieved by allowing the systems programmer to configure myriad buffer parameters, our opinion is that many network administrators do not have the time to effectively tune a network. What’s needed is an expert tool that will suggest and generate appropriate configurations based on commonly understood metrics, such as the number of users, the speed of the network, etc. In future versions, the software could dynamically adapt to changing load conditions.

We spent less time installing the Interlink 3762/ELC-3 communication controller. The ELC controller runs firmware, which can be adjusted via a control panel on the front of the box. We left the ELC control panel at its default values. The 3172-3 offload controller is a specialized personal computer running the OS/2 operating system (version 2.0 for our tests) and the IBM TCP/IP stack for OS/2. Getting the 3172-3 up and running involved software installation of OS/2, TCP/IP, and the additional offload software. All of this software had to be upgraded to the latest maintenance level several times during the testing process. In addition to offload mode, the 3172-3 can run in non-offload mode without OS/2. To do this, however, the hard disk must be replaced or reformatted and new software installed. The 3172-3 that we received did not have mounting hardware for a second disk and was installed “rack-mounted” in a cabinet, which precluded the ad hoc solution of resting a second disk on top of the controller. As a result, we did not test the 3172-3 in non-offload mode.

The combination of running MVS OpenEdition 5.1 under VM/ESA 1.1 and the newly-released 3172-3 offload controller held up testing for several months until VM device compatibility fixes were available.

5.2 Operation

On the whole, operating the IBM and Interlink MVS-based TCP/IP software was not difficult. Change control became an issue because we were trying many combinations of configurations and parameters in our tests. Some reduction in the number of files and parameters that must be maintained would have simplified change control. Change control was done by naming, i.e. using common naming elements to unify a set of files that are members of a single “network instance.” With PDS member names being limited to eight characters, there is not much room for variation, and suffixes are typically two or three characters strings, often without recognizable meaning.

One aspect of Interlink TCP/IP was that we could not get it to respond to a stop command and had to cancel the started task instead.

With respect to testing mainframe TCP/IP products, several problems arose that could be addressed by using the appropriate software. The first problem involves correlating the time difference between the MVS mainframe and the client workstation. When running a test, it would be beneficial to be able to call a routine that would return the MVS time. This would facilitate easier correlation of client/host activity. The second problem involved writing programs to parse RMF/SMF output, correlate CPU busy values, etc. with throughput values seen at the client. Although accurate time correlation between the host and client would help, an even better solution would be to request a specific RMF/SMF value at the client, which could then be incorporated into the client log.

5.2.1 Unique Characteristics of SNS/TCPaccess Sockets

The SNS/API Socket library routines provide three types of sockets:

- Stream sockets using TCP connection-mode service
- Hybrid sockets using UDP association-mode service
- Connectionless sockets using UDP connectionless service

Stream sockets look like typical BSD stream sockets and are created by

```
sock_desc = socket(AF_INET, SOCK_STREAM, 0);
```

The full-duplex, reliable byte-stream socket uses the TCP protocol to insure that data is not lost or duplicated. A stream socket must be in a connected state (via the connect() or accept() function) with another stream socket before any data can be sent or received through it. Data transfer can continue until this two-party connection is closed, shut down, or broken.

Hybrid sockets are a cross between connection-oriented, stream sockets and connectionless, UDP datagram sockets. Interlink calls the hybrid of these CO and CL service modes an association that establishes a two-party, long-term binding between a pair of hybrid sockets for the purpose of transferring UDP datagrams. Hybrid sockets are created by

```
sock_desc = socket(AF_INET, SOCK_DGRAM, 0);
```

which is somewhat confusing because DGRAM type BSD sockets do not have long-term binding between only two parties. An association must be established (via the connect() and accept() functions) between the two hybrid sockets before any datagrams can be transferred using the send() and recv() functions. However, trying to use sendto() or recvfrom() will abort with an error indication.

Connectionless sockets provide dynamic binding with each datagram sent, which permits multiplexed service with many remote correspondents through a single socket. CONNLESS sockets are created by

```
sock_desc = socket(AF_INET, SOCK_CONNLESS, 0);
```

These sockets do not require that a dedicated connection or association be pre-established before data transfer can occur. Rather, they allow datagrams to be sent to recipients specified in sendto() functions and to receive datagrams from multiple correspondents using recvfrom() which also returns the sender's socket address. However, trying to use bind(), connect(), send(), or recv() on a CONNLESS socket will abort with an error indication.

Unix programmers will notice that BSD socket type SOCK_DGRAM functionality has been split across hybrid and CONNLESS sockets with some additional features being added so that a server can issue a listen() and an accept() on a UDP socket to simulate a connection-based server. Using this pseudo-connection, hybrid sockets can transfer datagrams back and forth in TCP stream-like fashion, although without any guaranties of delivery. In place of sendto() over a hybrid socket, it may be possible to use a connect() then send() sequence because the socket association can be changed by issuing another connect() function. Likewise, in place of recvfrom() over a hybrid socket, you can try using an accept() then recv() sequence. When SNS receives a datagram, the hybrid socket code simulates a pending connection request that the receiver can acknowledge with an accept() function call. This facility is not provided in the Unix implementation of BSD sockets.

5.2.2 Port Number Assignments and Socket-address Binding

When a socket is initially created, it is called unbound because it has no associated addresses. A socket is bidirectional, so it has two addresses, each containing an Internet node address and a port number that are specified in a "sockaddr" structure. Typically, a server will use bind() to assign a local port number and host address to a socket so that inbound requests or messages can be received. A client will generally use connect() to assign a remote port number and destination address to its socket so it can send outbound messages. A "bound" socket will be associated with both inbound and outbound addresses.

Currently, in the SNS socket API implementation, the client and server cannot use the same port number. A server socket must use a port number less than 4096, and a client socket must use a port number above 4095. When using the SNS bind() function, only a server can explicitly specify a port number; the client must specify a port number of zero. When using SNS bind() to associate an inbound address with a socket, only a port number is allowed. The Internet node address must be set to zero or INADDR_ANY, else bind() will abort with an ESINVAL invalid indication. SNS/TCPaccess will assign the proper Internet home address when it can determine which of the possible network interfaces to use. The application software cannot decide which home addresses it will support.

5.2.3 Socket Error Codes

Whenever a C library routine detects an error, an error code is stored in the global variable, "errno". The compiler-supplied ERRNO.H header file maps the error codes and the perror() routine prints an appropriate diagnostic message. Both the SNS and IBM socket implementations provide their own additional header file of socket error codes. SNS defines 45 additional error codes, all defined with the prefix "ES" to indicate "Error in Socket call", and stores them in a header file called <SERRNO.H>. IBM defines 89 socket error codes with only a single-letter prefix "E" and stores them in a <TCPERROR.H> file. To check if a non-blocking socket read would wait for a datagram to arrive, the SNS socket application checks for ESWOULDBLOCK=11, while the IBM socket application checks for EWOULDBLOCK=35. Most of the codes have unique and non-equivalent names, so that any attempt to convert a socket application program from one vendor's socket implementation to another vendor's implementation will require more than just a simple recompiling of the C source code.

5.2.4 Header Files Library

This library contains the header files necessary for a C application program to properly interface to the socket library routines. These header files must be included by the C source code when the socket application program is compiled (by either the C/370 or SAS/C compiler) to insure proper compilation and execution with the socket runtime library. Unfortunately, the various header files are not quite BSD-compatible. The BSD header files commonly used with sockets on Unix have been modified to reflect extensions and restrictions of the MVS implementations. Some of the SNS headers were concatenated or merged to eliminate the need for multiple headers from which only a few lines were required for socket structures and symbolic definitions. For example, the SNS <socket.h> header file takes the place of seven headers: <sys/types.h>, <sys/time.h>, <sys/param.h>, <sys/stat.h>, <sys/ioctl.h>, <sys/fcntl.h>, and <sys/socket.h> with the Unix file I/O and pipes.

5.2.5 Handling Source Code Differences

In order to avoid having multiple copies of udptest source code that can handle the incompatible differences between HAL and SNS headers and library calls, we inserted a series of #ifdef and #endif statements using the compile time variables, HAL and SNS. This allowed us to include the header file <serrno.h> for SNS and include the header file <tcperrno.h> for HAL.

Table 6 lists a few examples from udptest that show differences in library calls from the source code where it makes library calls.

TABLE 6. *udptest Examples*

1	When we create a UDP socket for HAL we use
	`s = socket(AF_INET, SOCK_DGRAM, 0);`
	and for SNS we must specify:
	`s = socket(AF_INET, SOCK_CONNLESS, 0);`

2	To use bulkmode socket queuing with HAL, we specify:
	`setibmsockopt(s, SOL_SOCKET, SO_BULKMODE, &bulkstr, sizeof(bulkstr) );`
	and for SNS we set the send/receive queue sizes by specifying:
	`setsockopt(s, SOL_SOCKET, SO_SNDREQCNT, &sendQsiz, sizeof(sendQsiz) );`
	`setsockopt(s, SOL_SOCKET, SO_RCVREQCNT, &recvQsiz, sizeof(recvQsiz) );`

3	Datagrams are sent out through unconnected HAL sockets by calling:
	`sendto(s, (char *)p_dgram, msglen, 0, (struct sockaddr *) &outbound, sizeof(outbound) );`
	which won't work for SNS because its socket must first be connected:
	`connect(s, (struct sockaddr *)&outbound, sizeof(outbound) ):`
	`send(s, (char *)p_dgram, msglen, 0 );`

4	Before we can switch a HAL socket from bulkmode to non-blocking mode we must first call:
	`ibmsflush(s); /* to flush the Send queue */`
	`setibmsockopt(s, SOL_SOCKET, SO_NONBLOCKLOCAL, &turn_OFF, sizeof(turn_OFF));`

5	Before closing the HAL socket, we get the IUCV performance statistics for bulkmode operation by calling: `getibmsockopt(s, SOL_SOCKET, SO_BULKMODE, &bulkstr, sizeof(bulkstr) );`

6. Concluding Remarks

Benchmarking almost any software system is often more difficult than one would at first imagine. Networked systems increase the challenge, because each component adds to the overall complexity of the system. Thus, the tests we conducted to compare the performance of IBM and Interlink running TCP/IP applications—telnet(3270), FTP, and UDP—probably do contain some errors. We have done our best to treat both IBM and Interlink fairly and to provide consistent results.

Appendix A: MVS Telnet Data

TABLE 7. MVS Telnet CPU Utilization (Figure 2)

Sum of CPU		Vendor	
Sessions	**Rate**	**IBM**	**Interlink**
100	1	6.6	9.1
	1.5	7.2	6.5
	2	8.2	10
250	1	14	9.1
	1.5	11.6	9.4
	2	14.3	11
500	1	14.3	12.4
	1.5	18.1	14.5
	2	23.6	16.7
1000	1	21.7	18.6
	1.5	25.1	21.8
	2	39.3	30.1

TABLE 8. MVS Telnet Service Rate (Figure 4)

Vendor	Sessions	Rate	Service
IBM	100	1.0	2195
Interlink	100	1.0	2673
IBM	100	1.5	2391
Interlink	100	1.5	2356
IBM	100	2.0	2817
Interlink	100	2.0	3047
IBM	250	1.0	0
IBM	250	1.5	4059
Interlink	250	1.5	4485
IBM	250	2.0	6269
Interlink	250	2.0	6062
IBM	500	1.0	6541
Interlink	500	1.0	8174
IBM	500	1.5	9254
Interlink	500	1.5	10368
IBM	500	2.0	12730
Interlink	500	2.0	14081
IBM	1000	1.0	13874
Interlink	1000	1.0	22397
IBM	1000	1.5	14792
Interlink	1000	1.5	30717
IBM	1000	2.0	35072
Interlink	1000	2.0	44770

TABLE 9. MVS Telnet/AIX Client Response (Figure 5)

Sum of Response		Vendor	
Sessions	Rate	IBM	Interlink
100	1	28	31
	1.5	30	27
	2	31	26
250	1	0	0
	1.5	30	28
	2	33	32
500	1	32	30
	1.5	34	34
	2	43	41
1000	1	52	38
	1.5	54	59
	2	68	75

Appendix B: MVS FTP Data

TABLE 10. IBM FTP Server Throughput (Figure 6)

	1	2	4	7
8k	408	365	304	170
16k	545	475	340	202
28k	666	581	363	204

TABLE 11. Interlink FTP Server Throughput (Figure 7)

	1	2	4	7
2k	1177	709	396	212
8k	1179	688	390	210
16k	1172	664	381	210

TABLE 12. FTP Server Average Throughput, FDDI, “get binary” (Figure 8)

	1	2	3	4	5	6	7	8
IBM	961	740	511	382	305	254	217	185
Interlink	1315	838	571	427	345	288	248	219

TABLE 13. FTP Server Average Throughput, FDDI, “put binary” (Figure 8)

	1	2	3	4	5	6	7	8
IBM	944	580	551	414	318	257	217	185
Interlink	1344	780	540	405	338	277	230	201

TABLE 14. FTP Server Average Throughput, FDDI, “get ASCII” (Figure 8)

	1	2	3	4	5	6	7	8
IBM	965	601	394	298	237	196	164	165
Interlink	1458	738	563	396	321	263	217	178

TABLE 15. FTP Server Average Throughput, FDDI, “put ASCII” (Figure 8)

	1	2	3	4	5	6	7	8
IBM	880	422	279	221	175	143	122	106
Interlink	1055	539	359	271	216	179	153	133

TABLE 16. FTP Server Total Throughput, FDDI, “get binary” (Figure 9)

	1	2	3	4	5	6	7	8
IBM	961	1481	1535	1530	1529	1529	1524	1480
Interlink	1315	1676	1715	1708	1725	1732	1742	1757

TABLE 17. FTP Server Total Throughput, FDDI, "put binary" (Figure 9)

	1	2	3	4	5	6	7	8
IBM	944	1161	1655	1656	1591	1546	1525	1485
Interlink	1344	1560	1620	1621	1690	1665	1611	1615

TABLE 18. FTP Server Total Throughput, FDDI, "get ASCII"(Figure 9)

	1	2	3	4	5	6	7	8
IBM	965	1202	1184	1192	1187	1176	1152	1320
Interlink	1458	1477	1689	1584	1605	1578	1524	1429

TABLE 19. FTP Server Total Throughput, FDDI, "put ASCII" (Figure 9)

	1	2	3	4	5	6	7	8
IBM	880	845	837	884	877	863	859	849
Interlink	1055	1078	1078	1086	1082	1077	1075	1069

TABLE 20. FTP Server Average CPU Utilization, FDDI, "get binary" (Figure 10)

	1	2	3	4	5	6	7	8
IBM	32	46	47	46	46	46	45	45
Interlink	24	31	33	33	33	34	32	0

TABLE 21. FTP Server Average CPU Utilization, FDDI, "put binary" (Figure 10)

	1	2	3	4	5	6	7	8
IBM	43	54	73	72	71	69	63	61
Interlink	27	37	37	38	39	39	36	40

TABLE 22. FTP Server Average CPU Utilization, FDDI, "get ASCII" (Figure 10)

	1	2	3	4	5	6	7	8
IBM	35	44	44	44	42	43	43	41
Interlink	46	52	56	56	54	55	55	55

TABLE 23. FTP Server Average CPU Utilization, FDDI, "put ASCII"(Figure 10)

	1	2	3	4	5	6	7	8
IBM	50	50	49	48	47	47	47	36
Interlink	39	43	45	46	47	47	47	39

TABLE 24. FTP Server Average Throughput, Ethernet, "get binary" (Figure 11)

	1	2	3	4	5	6	7	8
IBM	664	372	271	218	182	155	135	117
Interlink	992	532	358	268	215	184	152	133

TABLE 25. FTP Server Average Throughput, Ethernet, “put binary” (Figure 11)

	1	2	3	4	5	6	7	8
IBM	998	551	369	277	221	186	155	137
Interlink	1074	582	374	288	242	190	165	146

TABLE 26. FTP Server Total Throughput, Ethernet, “get binary” (Figure 12)

	1	2	3	4	5	6	7	8
IBM	664	745	814	873	914	931	946	943
Interlink	992	1065	1075	1073	1076	1105	1067	1070

TABLE 27. FTP Server Total Throughput, Ethernet, “put binary” (Figure 12)

	1	2	3	4	5	6	7	8
IBM	998	1103	1108	1108	1108	1120	1091	1100
Interlink	1074	1164	1122	1154	1213	1140	1159	1171

TABLE 28. FTP Server CPU Utilization, Ethernet, “get binary” (Figure 13)

	1	2	3	4	5	6	7	8
IBM	22	25	26	27	28	28	28	22
Interlink	28	28	28	28	28	29	29	29

TABLE 29. FTP Server CPU Utilization, Ethernet, “put binary” (Figure 13)

	1	2	3	4	5	6	7	8
IBM	44	48	49	48	48	49	48	48
Interlink	29	34	34	35	35	35	35	

Appendix C: UDP Data

TABLE 30. UDP CPU usec/byte on MVS (Figure 14)

	Receive	Send
Test#1	1.964	3.158
Test#2	1.254	1.949
Test#3	0.771	1.001
Test#4	0.723	0.716
Test#5	0.434	0.573
Test#6	0.213	0.388
Test#7	0.21	0.385

TABLE 31. UDP Receive CPU Usage (Figure 15)

	Socket App CPU usec	TCP/IP CPU usec	Total CPU usec
Test#1	8,593,923	12,001,939	20,595,862
Test#2	4,488,029	8,663,309	13,151,338
Test#3	2,949,775	5,135,227	8,085,002
Test#4	2,429,871	5,150,307	7,580,178
Test#5	1,543,052	3,007,845	4,550,897
Test#6	1,065,861	1,171,567	2,237,428
Test#7	1,047,553	1,151,444	2,198,997

TABLE 32. UDP Send CPU Usage (Figure 16)

	Socket App CPU usec	TCP/IP CPU usec	Total CPU usec
Test#1	11,427,223	21,687,954	33,115,177
Test#2	6,487,349	13,950,463	20,437,812
Test#3	3,803,001	6,689,347	10,492,348
Test#4	2,454,400	5,052,116	7,506,516
Test#5	2,133,567	3,877,141	6,010,708
Test#6	3,094,671	973,501	4,068,172
Test#7	3,066,157	964,531	4,030,688

TABLE 33. UDP Throughput (Figure 17)

	Receive	Send
Test#1	438	265
Test#2	580	452
Test#3	912	863
Test#4	905	928
Test#5	937	954
Test#6	1550	1649
Test#7	1630	1645