

Evaluating LAN Communications Performance for a Real-Time Environment

Philip M. Irely IV, Robert D. Harrison, and David T. Marlow
System Research and Technology Department
Combat Systems Branch
Naval Surface Warfare Center, Dahlgren Division
Dahlgren, Virginia 22448-5000

Abstract

The Navy is a large user of real-time systems. A modern surface ship is deployed with hundreds of computers which are required for the ship to perform its mission. The Navy is moving away from using "niche" market components to meet its real-time computing needs and towards distributed processing using Commercial-Off-The-Shelf (COTS) computing components. The use of commercial real-time computing components requires careful evaluation to determine if they can meet the real-time requirements. This paper focuses on evaluating the communication resources for use aboard a Navy ship. Presented are: (1) Key performance metrics for assessing the communication capabilities of computers for use aboard Navy ships; (2) An assessment of existing methods for determining these metrics; (3) A methodology for collecting data to evaluate a particular component's performance related to these and (4) Examples of applying the methodology.

Keywords: Network, Testing, Performance Evaluation, ATM, TCP, UDP, IP, LAN

1: Introduction

The incorporation of COTS components in real-time distributed combat systems poses a multitude of challenges to the Navy system engineer. This paper focuses on one important challenge: how to evaluate COTS communications components to determine if they meet the performance requirements of a real-time distributed combat system. Many of these components are incorporated in the communications subsystem within a computing element. These components are comprised of hardware network adapters, their software device drivers, and a series of software communications protocol implementations as shown in Figure 4. Since the communications subsystem is a critical component of a

real-time distributed combat system, this is a key area for study.

The Navy uses (and plans to use) a variety of LAN communication technologies. Current Navy systems using COTS interconnection techniques are based on Ethernet and the Fiber Distributed Data Interface (FDDI). The communications subsystems to be considered includes these as well as emerging standards such as Asynchronous Transfer Mode (ATM). The Navy has great interest in ATM due to its unique properties such as the potential for high performance and Quality-of-Service (QoS) control. The media shown in Figure 4 could be of any of these media types.

The Navy primarily uses protocols from the Internet protocol suite to run on top of the LAN communications technologies: the Internet Protocol (IP), the Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP). These protocols comprise the "communications subsystem" in Figure 4. These Internet protocols are currently used in the shipboard environment and are key components in the Navy's SAFENET Standard [SAFENET].

To evaluate the adequacy of the COTS products selected, a set of performance metrics and an evaluation methodology applicable to real-time systems must be defined. Performance will be evaluated using these metrics and using the evaluation methodology.

2: Performance metrics

Metrics are defined in this paper with equations which reference the pseudo-code fragments of Transaction and Data Stream scenarios shown in Figures 1 and 2. (The pseudo-code is similar to the "C" programming language.) The pseudo-code fragments precisely show the scope of the data exchanges that must be performed.

Three standard operating system calls are referenced in the pseudo-code: send(), recv(), and gettimeofday(). The send() and recv() system calls are used to send and receive data respectively. The gettimeofday() system call returns the current time of day.

In the pseudo-code segments and the equations, it is assumed that both the client and server (or transmitter and receiver) have synchronized clocks or can generate synchronized time values. If synchronized clocks are not

$$\text{transaction time} = \text{stop_time_d} - \text{start_time_a} \quad (\text{EQ } 1)$$

The one-way latency time is computed as shown in Equations 2 and 3 which reference the pseudo-code in Figure 1.

$$\text{client to server one way latency} = \text{stop_time_b} - \text{start_time_a} \quad (\text{EQ } 2)$$

$$\text{server to client one way latency} = \text{stop_time_d} - \text{start_time_c} \quad (\text{EQ } 3)$$

Latency is often computed by dividing the time elapsed during a round-trip message exchange by 2 (i.e.

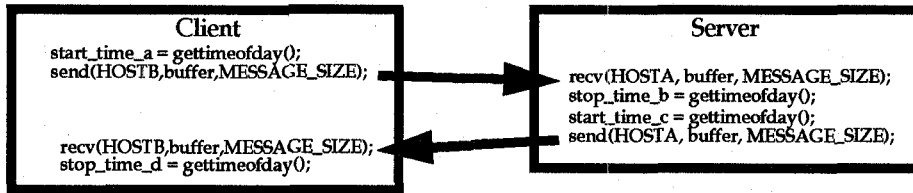


FIGURE 1. Transaction pseudo-code segment 1

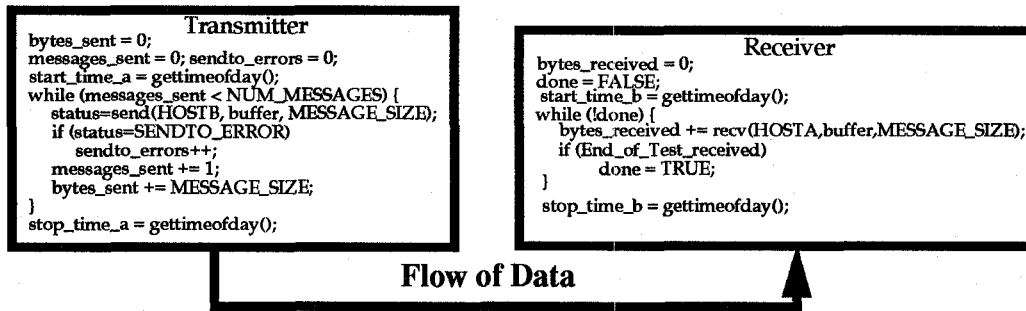


FIGURE 2. Data stream pseudo-code segment 2

available, a host can calculate how far its clock is offset from the other host and apply this offset to values read from the local clock to generate synchronized time. A variety of algorithms exist to do this [MILLS].

2.1: Transaction and one-way latency performance metrics

The computation of both round-trip transaction and one-way latency times is important in the real-time environment. The performance of many applications may be dictated by either or both of these metrics.

A single transaction time measurement is computed as shown in Equation 1 which references the pseudo-code in Figure 1. The transaction time metric is important to real-time processes such as machinery control applications which use transactions to initiate a series of lock-step operations to initiate a specific action. For example, a host may issue a request "open the actuator" to which the receiving host would respond "the actuator is open" when the action is completed.

average one-way latency). This can be can give misleading results in a heterogeneous environment where the one-way latency in each direction of a 2-way message exchange may be quite different. Also, the timing of a variety of real-time applications, such as control loops, is often dictated by a one-way latency chain of events, as shown in Figure 3, not a transaction time. Here, Process 1 sends a message to Process 2 to initiate some action. Process 2 sends a message to process 3 and so forth. Eventually, Process N sends a message to Process 1 indicating that computation chain is complete. Equation 4 shows the total communication time for the computation. Each latency in this summation is a one-way latency.

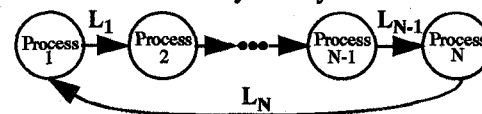


FIGURE 3. Latency chain of events

$$\text{total communication latency} = \sum_{i=1}^N L_i \quad (\text{EQ } 4)$$

2.2: Throughput metrics

Throughput is another key metric in the real-time environment. One example of where throughput is critical is the communication between a sensor and a signal processor. Here the sensor may rapidly generate large volumes of raw data for processing. Transferring this data to a signal processor for timely analysis requires high throughput.

Two types of throughput metrics are defined: interface throughput and end-to-end throughput. As explained in Section 3.2, interface throughput can provide misleading measurements in certain circumstances (e.g. UDP measurements). The metric can be useful, however, in measuring TCP throughput in scenarios where the "pipe" between the transmitter and receiver is full. The interface throughput metric is commonly used in the commercial networking industry where it is called "throughput".

Interface throughput is calculated as shown Equation 5 and 6 which reference the pseudo-code in Figure 2.

$$\text{transmitter interface throughput} = \frac{\text{bytes_sent}}{(\text{stop_time_a} - \text{start_time_a})} \quad (\text{EQ } 5)$$

$$\text{receiver interface throughput} = \frac{\text{bytes_received}}{(\text{stop_time_b} - \text{start_time_b})} \quad (\text{EQ } 6)$$

End-to-end throughput is calculated as shown in Equation 7. Unlike the interface throughput calculations, the "start of test" time is collected on the transmitting host and the "end of test" time is collected on the receiving host, thus accounting for the queueing delays in the communications subsystem.

$$\text{end-to-end throughput} = \frac{\text{bytes_received}}{(\text{stop_time_b} - \text{start_time_a})} \quad (\text{EQ } 7)$$

2.3: Messages per second performance metric

Similar to throughput, the messages per second metric can be computed on both an interface and end-to-end basis.

Interface messages per second measurements are computed as shown in Equations 8 and 9 which both reference Figure 2.

$$\text{transmitter messages/sec} = \frac{\text{NUM_MESSAGES}}{(\text{stop_time_a} - \text{start_time_a})} \quad (\text{EQ } 8)$$

$$\text{receiver messages/sec} = \frac{\text{bytes_received}}{(\text{stop_time_b} - \text{start_time_b})} \times \frac{1}{\text{MESSAGE_SIZE}} \quad (\text{EQ } 9)$$

A single end-to-end messages per second measurement can be computed as shown in Equation 10 and references Figure 2.

$$\text{end-to-end messages/sec} = \frac{\text{bytes_received}}{(\text{stop_time_b} - \text{start_time_a})} \times \frac{1}{\text{MESSAGE_SIZE}} \quad (\text{EQ } 10)$$

2.4: Sendto error metrics

The Sendto Error metric is only valid for a connectionless protocol such as UDP in the Internet protocol suite. It is the count of transmitter buffer overruns which happens when no socket buffer space is available. When this occurs, the packet being transmitted or received is dropped. Consequently, a message transmission which results in a Sendto Error is not sent on the network. A single sendto error measurement is computed as shown in Equation 11 which references Figure 2. This metric can be used synergistically with the data received performance metric defined below.

$$\text{number of sendto errors} = \text{sendto_errors} \quad (\text{EQ } 11)$$

2.5: Data received performance metrics

Data received performance metrics measures the percent of data that was sent by the transmitter which was received by the receiver. End-to-end data received measurements are computed as shown in Equation 12. This metric is probably only useful for a connectionless protocol like UDP which can lose data. It should be noted that bytes_sent may include data which resulted in a Sendto error.

$$\text{percent data received} = \frac{\text{bytes_received}}{\text{bytes_sent}} \quad (\text{EQ } 12)$$

This metric can be used in concert with the sendto error metric to draw conclusions about system performance. For example, if low performance is observed and there are few sendto errors and the percent data received is low, data is probably being lost at the receiver. If on the other hand, there are a large number of sendto errors and the data not sent due to them accounts for a large percentage of the lost data in the percent data received metric, data is probably being lost at the transmitter.

3: Existing performance analysis tools

Initially, dedicated hardware based network analyzers and public domain software tools were evaluated to see if they could measure the metrics defined.

3.1: Dedicated network analyzers

Dedicated Network Analyzers are able to measure both latency and throughput. Testing determined that the measurements reported by these analyzers did not correspond to the actual performance observed by the applications transmitting and receiving data. This is because these systems are only capable of measuring performance while the data is present on the transmission media being used. Since the dominant source of application to application queueing delays occurs before the data is present on the transmission media and after it is removed from the transmission media, these measurements are not appropriate in a real-time environment where application to application performance is key.

3.2: Software performance analysis tools

Several public domain software tools, including NetPerf and TTCP, which are widely used to measure network performance, were evaluated to determine their suitability to measure the metrics defined. Since both tools measure the communications performance observable within an application, they were thought to be well suited to measuring the real time metrics. It was found that neither of the tools measure all the quantities of interest.

For latency, NetPerf actually calculates an average round-trip time. It is assumed that the average round-trip time is approximately equivalent to twice the one-way latency in either direction. In a heterogeneous real-time environment, this assumption may not be true. In a shipboard environment, system performance can vary widely from high-performance multiprocessor (e.g. a signal processor) to a low-performance single board computer (e.g. a fire-alarm), thus the one-way latency in each direction can also vary widely. This variance would be masked by an average round-trip time calculation. TTCP does not measure latency or any quantity from which latency can be derived.

For throughput, analysis revealed that these tools sometimes only measure interface throughput rather than end-to-end-throughput. (TTCP always measures interface throughput. NetPerf measures interface throughput for UDP). The difference between interface throughput and end-to-end throughput is shown in Figure 5.

Interface throughput is defined as the amount of data per unit time that can be written into (transmit interface throughput) or read out of (receive interface throughput) the communication subsystem by an application. Transmit and receive interface throughput are measured independently

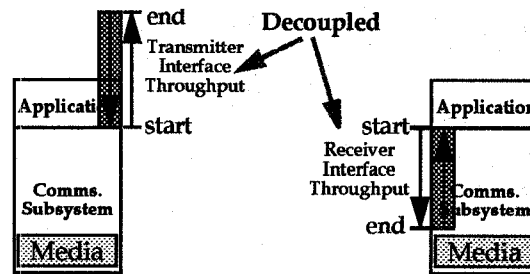


FIGURE 4. Interface throughput measurements

so these measurements do not account for the time data is queued in the communications subsystem. Under a variety of conditions, interface throughput measurements can yield misleading results.

For example, in Figure 5 below, the UDP performance of a workstation is plotted. The top curve shows receive interface throughput while the bottom curve shows end-to-end throughput. A common misconception is that these two metrics are measuring the same quantity, but as can be seen in the plot, this may not be true. The difference, between what the tester was expecting to be measured and what was actually being measured, provided the motivation to precisely define the metrics of interest in a real-time environment and to develop a tool to measure them.

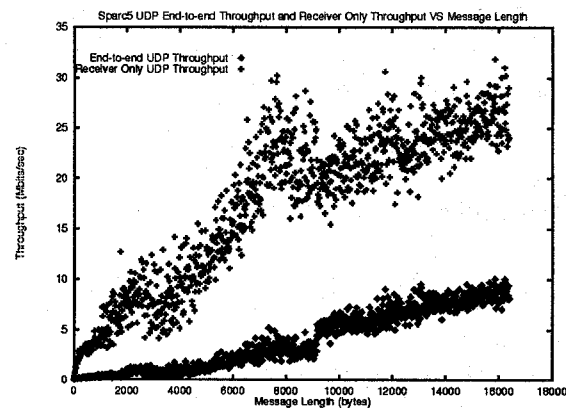


FIGURE 5. End-to-end throughput versus interface throughput

Neither TTCP nor Netperf directly computes the percent of data received metric. Both do, however compute the total bytes received from which this metric could be derived.

4: Measurement methodology

Now that metrics have been defined for measuring real-time communication performance, a methodology must be developed for collecting data to

determine a particular communication component's capabilities. In addition, a means is needed for system engineers to view the data collected so that they can quickly assess the component's performance for a particular metric. The term visualization is used in this paper to express this function.

The ability to bound a performance measurement is critical in the design of a real-time system. Even in an ideal real-time environment, measured performance is not consistently repeatable, but generally falls within a certain range. The measurement methodology used must be able to quantify these bounds if it is to be useful in the real-time environment.

The visualization of the data collected is also an important part of the measurement methodology. Test fidelity, the ability of the measurement methodology to accurately capture the true performance of the system over the defined test range, is critical in the visualization process. It is dependent on two test aspects: 1) statistical characterizations such as a mean, variance and standard deviation which are computed for a particular message size; and 2) the number of samples (i.e. different message sizes) collected. The statistical computations show how stable the measured performance level is for a particular message size. The number of samples is set so that disturbances in an otherwise well-behaved dataset are not missed.

4.1: Key measurement parameters

Three key parameters are defined to drive the data collection process: MESSAGE_SIZE, NUM_MESSAGES, and NUM_ITERATIONS.

MESSAGE_SIZE is the size of each message sent during a single performance measurement. From an application's point of view it is the size of each message sent by the transmitting application.

NUM_MESSAGES is the number of messages of the same MESSAGE_SIZE sent during a single performance measurement. For a throughput measurement, it is the number of user messages (of length MESSAGE_SIZE) which are sent during a test. Varying this parameter can affect how full the "pipe" is between the transmitter and receiver. For a latency or transaction measurement, it is the number of individual latency packets or round-trip message exchanges sent in calculating the latency or transaction time experienced between two hosts.

NUM_ITERATIONS is the number of times to repeat a single performance measurement. Multiple iterations are done so that statistics such as means, standard deviations, and variances can be computed and used to bound performance thus increasing fidelity

A schematic description of a test run is shown in Figure 6.

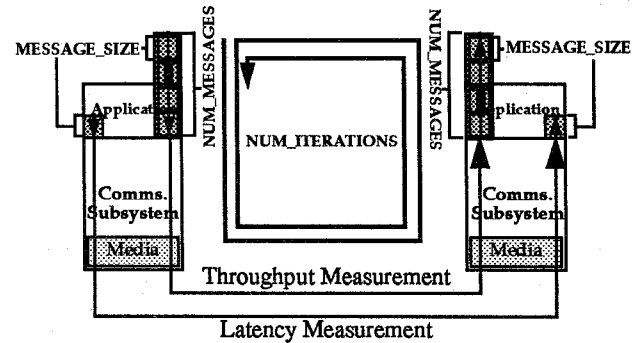


FIGURE 6. Schematic operation of a test run

4.2: Visual representation of measurements

The three key parameters of a test run play an important role in the visualization of the data collected. Each test run has a scope which is the granularity of the measurement. Latency is computed for each user message sent while throughput is computed for the entire stream (of length NUM_MESSAGES) of user messages.

TABLE 1. Scope of measurement and measurements per plot point

Metric	Scope of each measurement	Number of measurements per plot point
latency	MESSAGE_SIZE (i.e., 1 message)	NUM_ITERATIONS x NUM_MESSAGES
throughput	NUM_MESSAGES (e.g. 1 group)	NUM_ITERATIONS

Collected data is visualized using a plot. Each point on a plot is actually a statistical representation (usually a mean) of multiple measurements as shown in Table 1. The number of iterations (NUM_ITERATIONS) specified for the test controls how many samples are collected per point. For example, each point on the plot in Figure 5 of end-to-end throughput versus the size of the user message (MESSAGE_SIZE) is computed as shown in Equation 13.

$$\text{end-to-end throughput} = \frac{1}{\text{NUM_ITERATIONS}} \sum_{i=1}^{\text{MAX_ITERATIONS}} \text{end-to-end_throughput}_i \quad (\text{EQ } 13)$$

To assess stability and repeatability of a performance measurement, variance and standard deviation are computed as well as shown in Equations 14 and 15.

$$\text{latency variance} = \frac{1}{\text{NUM_ITERATIONS} - 1} \sum_{i=1}^{\text{MAX_ITERATIONS}} (\text{latency}_i - \text{mean_latency})^2 \quad (\text{EQ 14})$$

$$\text{latency standard deviation} = \sqrt{\text{latency variance}} \quad (\text{EQ 15})$$

Standard deviation or variance is shown in a plot with errorbars. For each standard deviation mean point plotted, an error bar is drawn from the mean plus one standard deviation to the mean minus one standard deviation. Variance is plotted in a similar manner.

To characterize the bounds of performance variance, during a test run the maximum value and minimum values of a metric are also recorded for each MESSAGE_SIZE. MAX/MIN plots are graphed with an error bar drawn from the mean value to the maximum value and to the minimum value recorded for a specific MESSAGE_SIZE during the test as shown in Figure 7.

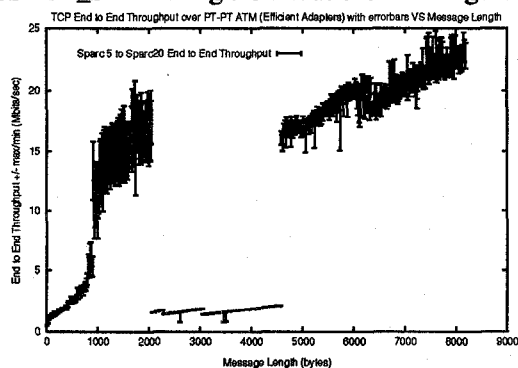


FIGURE 7. Plot with MAX/MIN error bars

4.3: Parameters used for test runs

In the tests presented here, the scope is applied to the user message length with a range 16 bytes to 16 Kilobytes with samples taken every 16 bytes, NUM_ITERATIONS was set to 10, and NUM_MESSAGES was set to 2000. These settings represent a trade-off between test run time and fidelity. Using these parameters, tests can take over 24 hours to complete.

5: Applying the methodologies and metrics to COTS components

All measurements presented in this paper were produced with a program called NTTCP which was

developed at the Naval Surface Warfare Center, Dahlgren Division. NTTCP is based on the public domain TTCP implementation. A rewrite of that code and a significant amount of new code was written to measure the real-time metrics and to use the data collection methodology defined in this paper.

All tests are performed on the COTS components as they are delivered from the manufacturer with no special tuning (e.g. changing default socket buffer sizes from the application, the default Maximum Transmission Unit (MTU), or other tuning parameters). The rationale for this is simple: we are trying to assess how the COTS product will perform as delivered. Performance tuning will be done after areas in which a particular technology set is deficient are identified.

One characteristic to note for both the TCP/IP and UDP/IP measurement is that throughput generally drops at user message sizes which are "near" multiples of the MTU which is the size of the largest IP datagram which can be sent over a data link without being fragmented. MTUs vary with data link types: 1500 bytes for ethernet; 4352 bytes for FDDI; and 9180 bytes for ATM. We say user messages sizes "near" multiples of the MTU because TCP/IP packets require a 40 byte header and UDP/IP packets require a 28 byte header which are inserted before the user data in a packet. At the point where the user data plus the headers exceeds the capacity of a single IP datagram (i.e. the MTU), an additional packet is required. At this point, the throughput drops. The throughput increases as more user data is placed in the additional packet because the header overhead is dominated by the data size in that packet.

5.1: ATM test procedures

ATM is being evaluated by the Navy for use in the combat system environment. Our ATM testing was performed in a Sun Workstation environment running the Solaris 2.4 operating system. The focus of the tests was running IP over ATM. As a result, the unique features of ATM (e.g. QoS) were not exercised or tested.

The test configuration used for the ATM testing is shown in Figure 8. Two workstations are connected to an ATM switch. All ATM interconnects are over SONET OC-3 links which operate at 155 Mbits/second. A Permanent Virtual Circuit (PVC) is established between the two workstations. Several types of ATM adapters

(including Fore Systems and Efficient Networks) and a Newbridge switch were used during the tests.

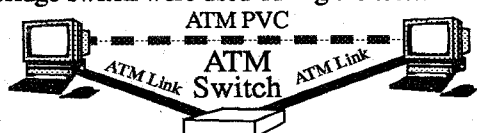


FIGURE 8. ATM test configuration

5.2: Early ATM testing

The first round of testing was done on boards from Efficient Networks and Fore Systems. These boards performed quite well, however, there were some dramatic performance anomalies as shown in Figure 9. The

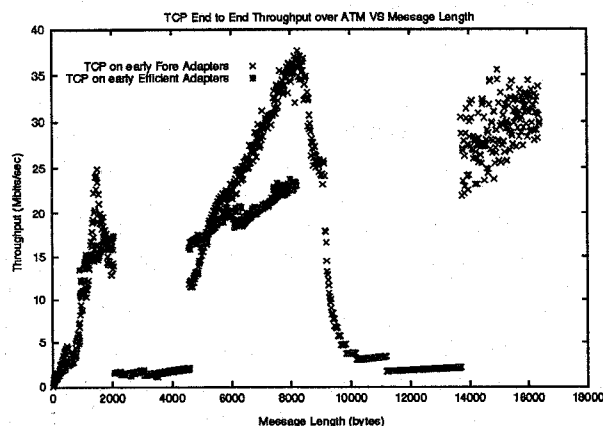


FIGURE 9. Early ATM performance

throughput measured for messages of between 2000 and 4500 bytes or 9180 and 14000 bytes dropped dramatically. (Here the scope and fidelity selected allow us to visualize these drops). When the phenomenon occurred with the first adapter tested, it was suspected that it was an adapter problem. However, when it occurred on the second card tested, it looked more like a host problem.

The problem uncovered is related to the complex interaction between the TCP implementation on the host, buffering parameters used by the application, and some parameters used by ATM. A detailed description of this behavior is described in [COMER]. This problem was reported to the vendors and was fixed in a subsequent release of the vendor's software drivers and the host operating system.

Given that the test run shown in Figure 9 is performed over ATM links running at 155 Megabits/second, the throughput results are very disappointing.

5.3: "Classical IP" versus a proprietary solution

In the following tests, Fore ATM adapters hosted in Sun Microsystems SparcStation 20 Model 51

workstations are being tested. The goal of these tests is compare the performance of a proprietary solution for running IP over ATM with a standard's based solution, Classical IP and ARP over ATM [RFC1577]. Plots comparing end-to-end throughput are shown in Figures 10 and 11.

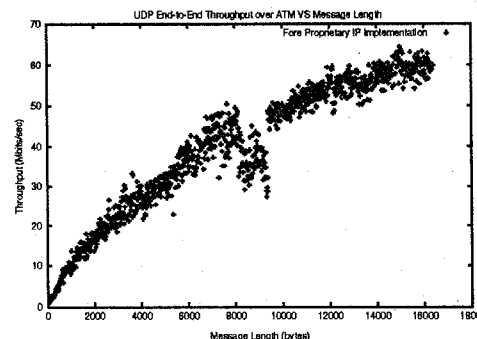


FIGURE 10. Fore proprietary UDP end-to-end throughput over ATM

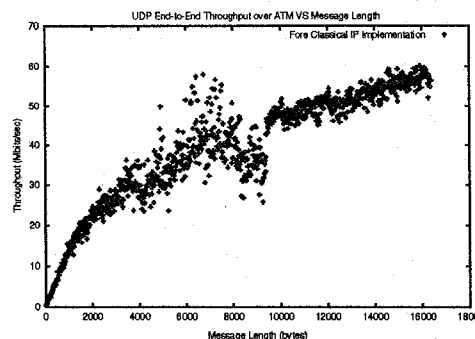


FIGURE 11. "Classical IP" UDP end-to-end throughput over ATM

The throughput and latency (although not shown here) observed with the two solutions is nearly identical. Thus, there appears to be no advantage in this case of using a proprietary solution instead of a standards based solution. In fact, the "Classical IP" (CIP) implementation may even have a slight performance edge over the proprietary solution. This may be a result of a newer implementation than the proprietary solution which will likely be phased out. Any performance differences would likely be in the connection establishment procedure which should have a minimal impact (if any) on the tests conducted.

5.4: Implementation impact on performance

The implementation of an ATM adapter and it's drivers has an impact on performance. Poor performance of a particular test may be due to design trade-offs for the adapter rather than a limitation on the underlying networking technology. The next tests compare the

performance of two typical ATM adapters, again from Efficient Networks and Fore Systems. In these tests, the adapters are placed in SparcStation 20 Model 51 workstations running the Solaris 2.4 operating system. The workstation type and operating system are not changed during the tests. Since only the ATM adapters themselves are varied between tests, any performance differences noted should be due to the adapters and their drivers.

In the transaction and latency tests (not shown), the Efficient adapter has a very slight performance advantage over the Fore for messages of less than 4 kilobytes. After this point, the Fore adapters increasingly surpass the performance of the Efficient Networks adapters as the message size gets larger.

On the other hand, the Efficient adapters increasingly outperform the Fore adapters on throughput performance. At around 9000 byte messages, the Efficient adapters have about twice the throughput of the Fore adapters as shown in Figure 12. The reason for this can be seen in Figure 13.

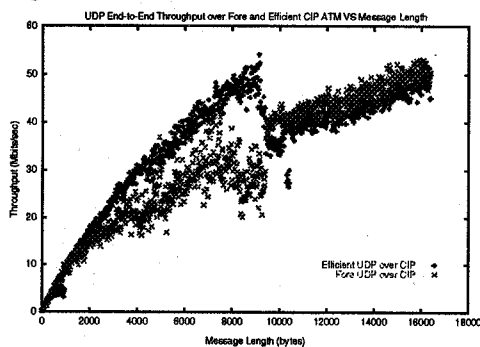


FIGURE 12. Fore and Efficient ATM adapters percent of UDP data received

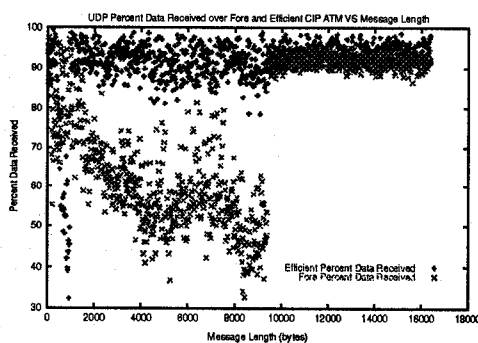


FIGURE 13. Fore and Efficient ATM adapter end-to-end throughput

When Efficient adapters are used, they generally receive between 90 and 100 percent of the UDP data transmitted. This leads to the steady throughput increase with message size. At the Maximum Transmission Unit

(MTU) for ATM (e.g. 9180 bytes), the throughput drops, as expected, and then steadily increases.

The performance of the Fore adapters grows rapidly up to about 2000 bytes but then grows more slowly as the MTU is approached. The performance actually jumps up after the MTU and then grows steadily. After the MTU the Fore adapters consistently outperform the Efficient adapters. The reduced performance noted between 2000 bytes and the MTU is a result of the Fore adapters receiving just under 50 percent of the data transmitted. This corresponds exactly with the 50 percent performance advantage of the Efficient adapters at this point.

5.5: ATM testing conclusions

ATM is in it's infancy but as can be seen in the testing, it is maturing. The initial boards tested showed very poor performance, boards tested later showed promising performance with some performance anomalies. Boards being tested today show the high performance potential of ATM and that implementation flaws are being corrected. Also, different ATM adapters may show showed markedly different performance characteristics even in the same environment.

6: Conclusions

This paper defines metrics for evaluating the LAN communications performance in a real-time environment and methodology for gathering data based on these metrics. As demonstrated, the metrics can be used to evaluate the performance of COTS components. In addition to observing general performance trends, these tools can be used to reveal subtle performance characteristics.

7: References

- [COMER] *TCP Buffering and Performance Over An ATM Network*, Douglas E. Comer and John C. Lin, Purdue University.
- [MILLS] RFC-1305, Mills, D. *Network Time Protocol (Version 3) Specification, Implementation and Analysis*, March 1992.
- [RFC-1577] RFC-1577, M. Laubach, *Classical IP and ARP over ATM*, January 1994.
- [SAFENET] MIL-STD-2204A - *Survivable Adaptable Fiber Optic Network (SAFENET)*.