Performance Measurements of Motorola's Implementation of MAP

W. Timothy Strayer
Alfred C. Weaver

Computer Networks Laboratory
Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903

(804) 979-7529

Introduction

The Computer Networks Laboratory at the University of Virginia has recently completed a performance study of the Motorola implementation of the Manufacturing Automation Protocol (MAP). Three layers in the seven layer ISO OSI Reference Model were identified as points of major data transfer service enhancement: the datalink layer, the transport layer, and the Common Application Service Elements (CASE) service of the application layer. Data transfer performance measurements were made and observations are offered.

The Motorola MAP implementation includes the full seven layer protocol stack as defined by MAP 2.1. It includes Directory Services for the resolution of application titles into some other form of information, specifically addresses and service access points. The File Transfer, Access and Management (FTAM) and the Manufacturing Message Format Standard (MMFS) protocols are also implemented, but were not studied for this report.

We include a brief discussion of each of the layers of interest and the services and enhancements provided. We then discuss the network used and how the application (performance) programs employed the communications software. Finally, the methods used to collect the performance data along with our data and observations.

Layers of Interest

The datalink layer is the lowest layer where data transfer primitives are provided. At this layer, bits may be grouped into frames and transferred as a whole. Thus a node-to-node, unguaranteed *datagram* service is provided.

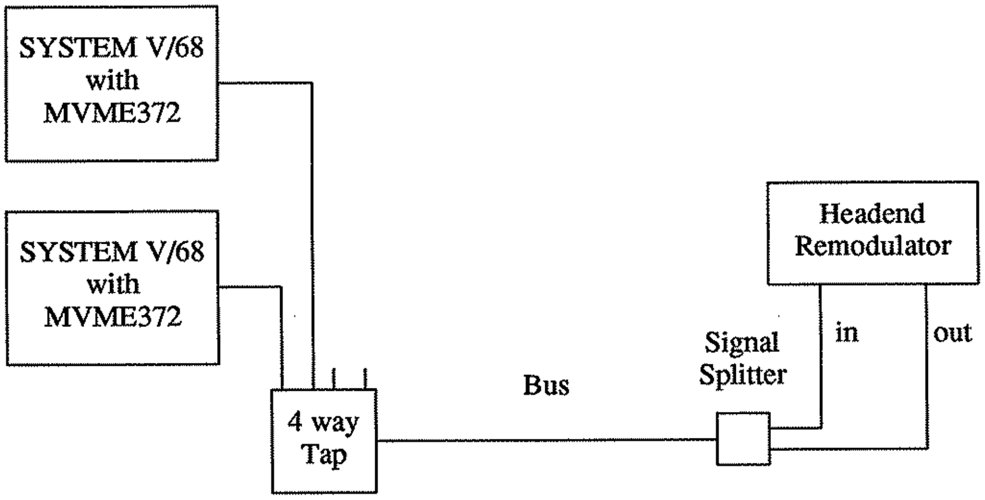
The transport layer builds upon the datagram services of the datalink layer and the routing services of the network layer to provide end-to-end data transfer. MAP 2.1 prescribes Class 4 Transport, which provides a highly reliably connection-oriented service called a *virtual circuit.*

Through error detection and recovery, the virtual circuit provides the transport user with the illusion of having a dedicated communication line established just for his purposes. Messages, generally of arbitrary size, are segmented into finite size protocol data units (PDUs) and transferred, and each PDU is acknowledged. The responsibility of achieving reliability and segmentation is completely assumed by the transport entity, and thus is transparent to the transport user.

The application layer provides services to the network user while allowing that user to distance himself from concerns of the network itself. CASE provides the services most commonly required of a network: association of a local application process to a remote application process, data transfer, graceful close, and abort. Application titles are used instead of network addresses and service access points. These application titles are resolved into the appropriate addresses by Directory Services, a co-occupant of the application layer.

The MAP Network

The MAP network consisted of two Motorola SYSTEM V/68s, each running the UNIX operating system, and each employing the MVME372 MAP Interface Module front-end processor for offloading the task of communications. This is shown in Figure 1. The seven layers of the MAP software, along with VMEbus communications protocol software, are downloaded onto the communications board. Each layer is implemented as a task. Task and memory management is done by VRTX, a real-time, multi-tasking kernel. The tasks communicate by a queue *pend* and *post* messaging mechanism over shared memory. In a similar fashion, programs on the host wishing to use the services of the communications board use the Common Environment to provide inter-task communication across the VMEbus.



Host: Motorola SYSTEM V/68
- MC68020 16MHz microprocessor
- UNIX
- 4 Megabyte RAM

Front-End Processor: MVME372 MAP Interface Module
- MC68824 Token Bus Controller
  (IEEE 802.4 MAC layer controller)
- MC68020 12.5 MHz microprocessor
- VRTX
- 640 Kilobytes onboard RAM
- MC68901 Multi-Function Peripheral
  (interrupt handler and timers)

INI Headend Remodulator with Cable Kit

Figure 1. Motorola MAP Token Bus Network

The Common Environment provides the means by which processes running on the host can communicate with the MAP communications board. This communication is affected by formatted data structures called Event Parameter Blocks (EPBs). An EPB holds such information as which process queue was sending the event and to which queue process it is headed. EPBs, therefore, provide the method for an application process to communicate directly to the processes implementing the layers of MAP, as well as the means for intra-layer communications on the MAP communications board itself.

Another field in the EPB is a buffer pointer. This is a pointer to a structure which holds the actual data to transfer as well as its length (and in a chain fashion, that buffer may point to others). The application process is responsible for filling this buffer.

Within the current Common Environment, there is a constraint on the amount of data that can be transferred per EPB. This maximum is 1050 bytes, which implies that the application process must do segmentation of large amounts of data itself. However, the buffer size is a configurable parameter to the Common Environment, and with it set to a larger value, segmentation of larger messages may be restored as an important property of the transport layer.

Performance Programs

There was no global external clock with which we could synchronize both of the MAP stations. Therefore we used the system call *times()* which provided a resolution of 1/60th of a second, or 16.67 milliseconds. We ran the experiments sufficiently long (i.e. minutes) to obviate the effect of the poor resolution.

To measure throughput, one station was designated the transmitter and the other the receiver. For CASE and transport layer measurements, an association (or connection) was made by the transmitting station. The transmitter then looped, sending messages as quickly as it could, while the receiver likewise looped, accepting incoming messages. The start and stop times were recorded for the transmitter and the throughput was calculated using this time.

In the CASE end-to-end delay experiments, we did not classify one station as the transmitter and one as the receiver; rather, one station was designated initiator. The stations then alternated the role of transmitter, trading messages one for one. By timing the initiator from start to finish, and knowing that the total number of messages relayed was twice that sent by one station, the end-to-end delay per message was calculated as the total time divided by the total number of messages. Again, by measuring the performance of many messages we overcame the effect of poor clock resolution.

Throughout these experiments we note two critical resources: (1) the use of the VMEbus and the copies associated with moving data from user data space to host shared memory and then across the VMEbus from the host to the communications board, and (2) the communications hardware and the software of the board itself. Thus to use the communications services to transmit required three copies of the data: one from user data space to shared memory, then from shared memory on the host to the shared memory on the communications board (requiring contention for the VMEbus), and finally from the on-board memory onto the physical media. Likewise three copies were required to receive data.

CASE Performance

CASE performance measurements were made for throughput and end-to-end message delay. We varied the message size from 16 bytes to 1024 bytes, considering that the Common Environment placed a constraint on the largest message size.

In the throughput measurements we sent messages of varying size from the transmitter to the receiver. Within the CASE EPB we could set a boolean value End Of Transmission (EOT), which could be used to mark the end of a set of messages. For any messages that the EOT was set to true, the CASE layer would not allow subsequent messages to be transmitted until the receipt of that message had been acknowledged. By setting EOT to true in every message we observed a "stop and wait" protocol. By setting EOT to true for only the last message, we observed pipelining to the extent allowed by transport's window size.

Figure 2 shows both curves with respect to throughput. Note that the pipelining curve displays a nearly linear ascent to a maximum throughput of about 125 Kbytes/sec for 1000 byte messages, then a drop in throughput for messages of length 1024 bytes. The drop is due to segmentation at the transport layer once CASE's and Session's header had been affixed. The linear nature of the curve shows the strong relationship between message size and transfer rate; for each doubling of the message size the throughput also nearly doubled.

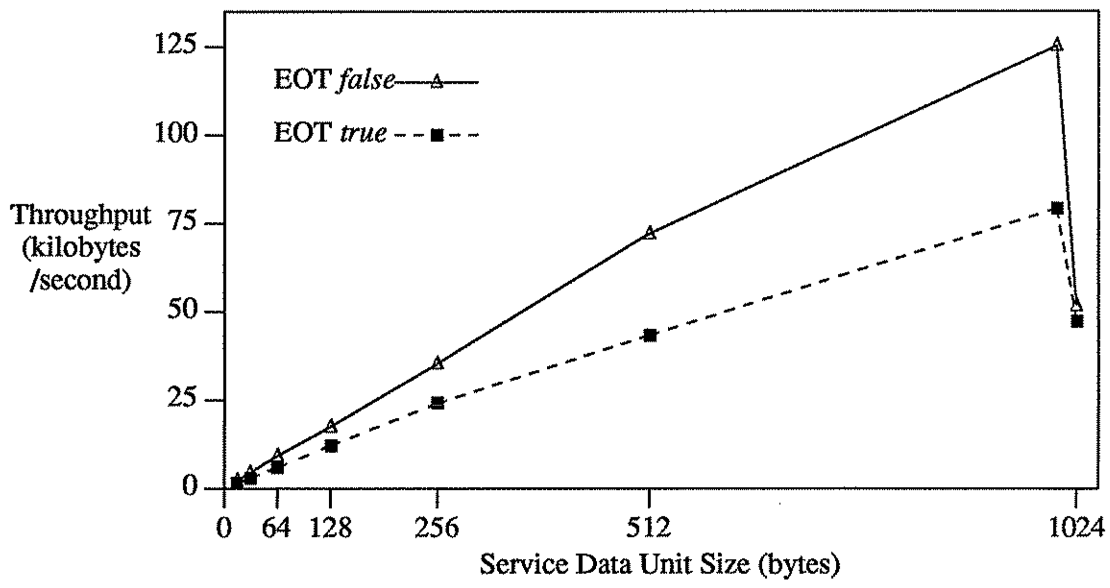


Figure 2. Throughput vs. CASE Service Data Unit Size

Figure 3 shows the number of messages that can be transmitted per second at the CASE layer for both stop-and-wait and pipelined protocols. The number of messages per second is relatively unaffected by the length of the message for lengths less than 1000 bytes. This was a constant of about 140 messages per second. The constant nature of this graph suggests that there was a constant rate of transfer for messages that use the entire protocol stack. First the message was copied and transferred across the VMEbus to the communications board. Then the pointers to the messages were passed from layer to layer (i.e. from task to task) and headers were constructed and prefixed until it was placed on the signalling media at a rate of 10 megabits (1.25 megabytes) per second. VMEbus transfers are measured in number of transfers per second, since the overwhelming majority of the cost of transfer is in start up. Once the message is on the board (presumably at a constant cost per message, regardless of message size), a pointer to the message was passed from task to task. Headers were constructed and prefixed to each message for each layer in the protocol stack. Again, these were constant costs per message. Since each message at CASE was acknowledged, there was a constant delay per message while awaiting that acknowledgement. This delay was sufficient to minimize the effect of the per-byte cost due to the copies from user data space to host shared memory to communications board shared memory.

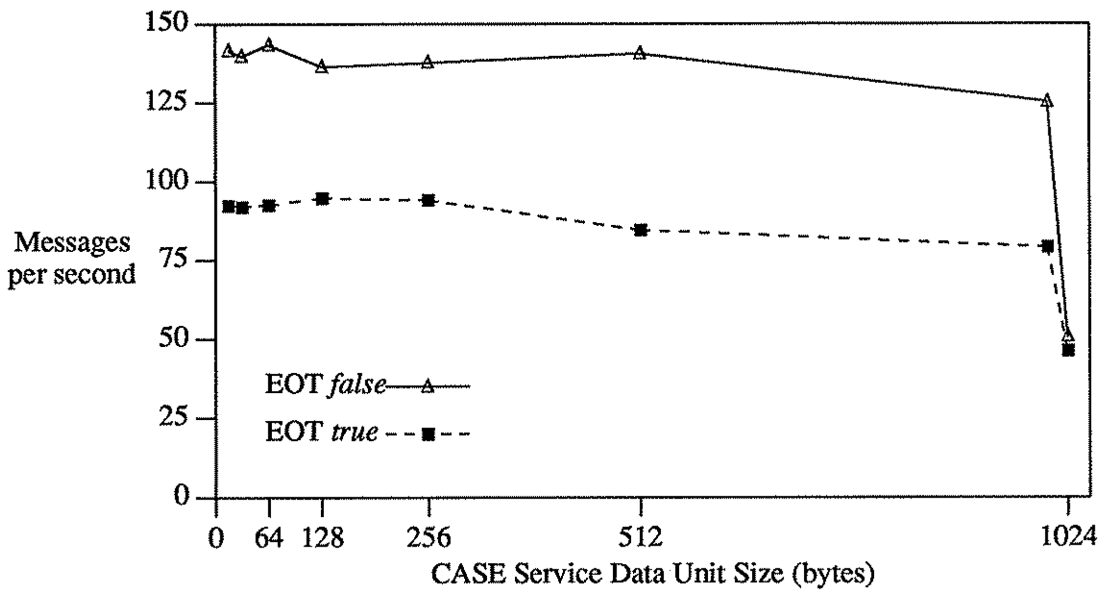


Figure 3. Number of Messages per Second vs. CASE Service Data Unit Size

Figure 4 shows that there was a constant and a per-byte cost incurred by using the CASE services. The y-intercept of about 16 milliseconds represents the start-up and per-transfer delays for each message. These delays were not affected by the size of the message. Added to these constant delays were the delays for copies, which did have a per-byte cost.

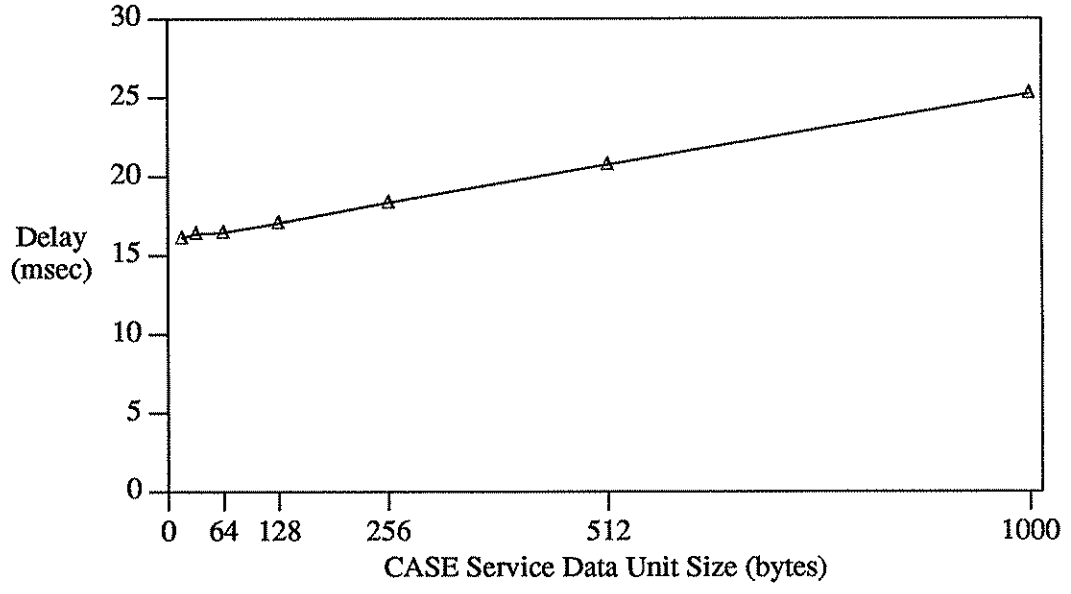


Figure 4. End-To-End Delay vs. CASE Service Data Unit Size

Transport Performance

The CASE data transfer services did little to enhance the data transfer services of the transport layer. CASE enhancements pertain more to associating processes on different nodes for the purposes of data transfer. Hence, the transport throughput (Figure 5) and number of messages per second (Figure 6) graphs are very similar. The throughput again shows linear dependence on message size, with a severe drop in the graph for messages of size 1024 due to segmentation. The number of messages per second is about 163, which stays fairly constant for the same reasons as stated above.

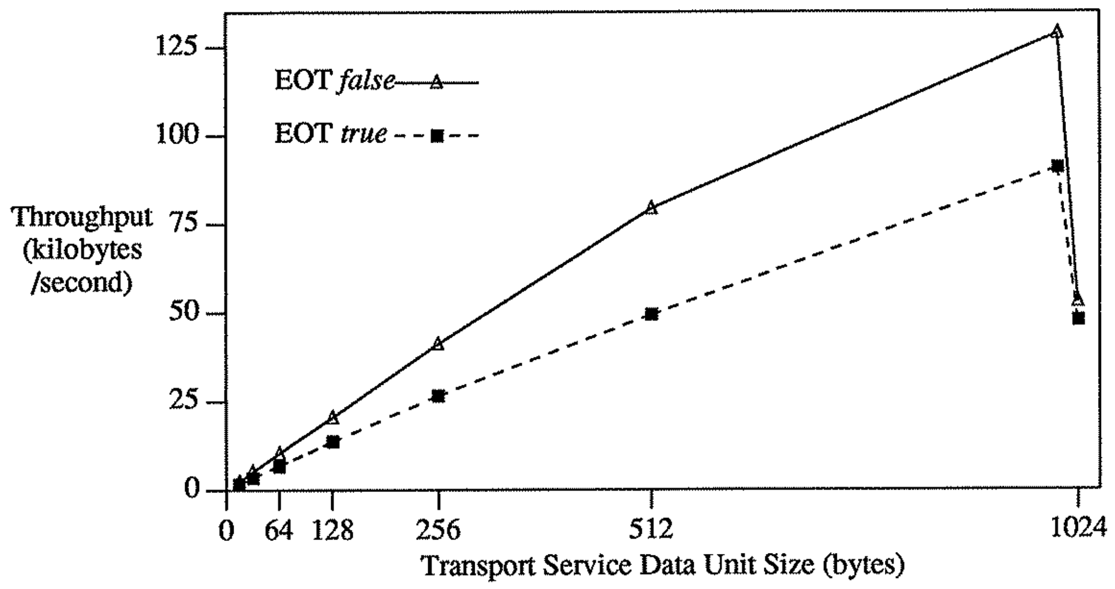


Figure 5. Throughput vs. Transport Service Data Unit Size

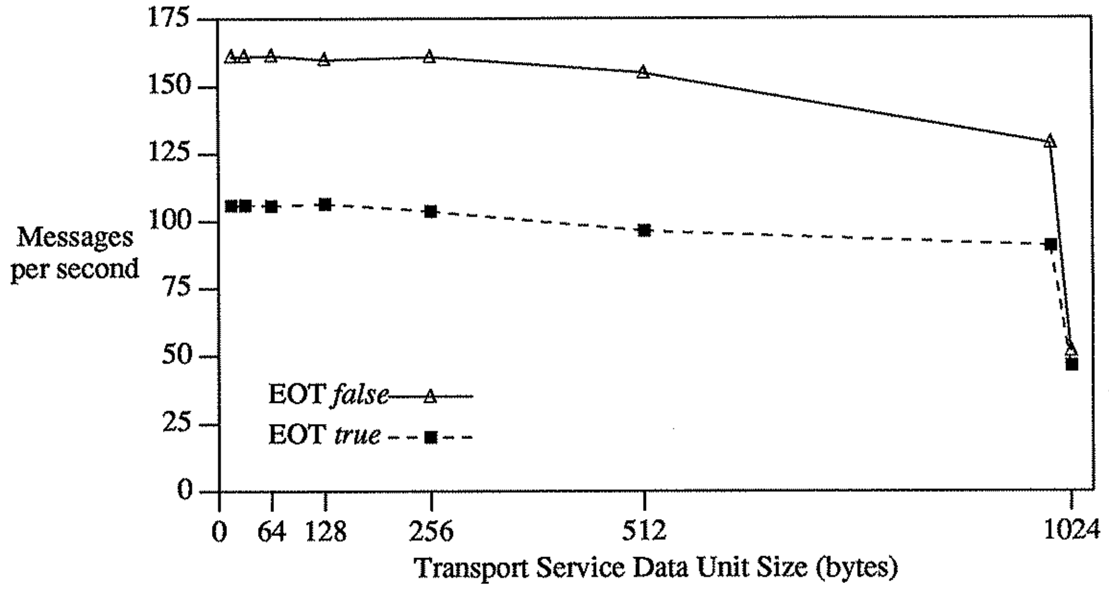


Figure 6. Number of Messages per Second vs. Transport Service Data Unit Size

Datalink Performance

The datalink layer provided the most primitive interface to the network that would allow the framing of bits into messages of variable, but limited, length. The service provided by the datalink layer is called a datagram, where the message is not guaranteed delivery, but rather it is guaranteed only the best efforts of the network. No acknowledgements are provided for datagrams.

Again there was a designation of one station as the transmitter and one station as the receiver. The transmitter looped, sending messages as quickly as possible, while the receiver looped accepting them. Since there were no acknowledgements, the transmitter's speed was not influenced by the speed of the receiver. Thus the rate of transmission was dependent only upon how fast the transmitter could format EPBs, how fast the copy from host to communications board could be done, and how fast the network could be accessed.

Figure 7 shows datalink throughput vs. message size. Note that this curve has logarithmic shape, as opposed to those of CASE, which were linear in nature, and asymptotically approaches 130 kilobytes/second. This indicates that there was a per-byte influence during the transfer of each message. Since the datalink layer is so close to the actual hardware, the speed of the data transfer prevents the per-transfer cost from minimizing the effect of the cost of the copies and (to a much lesser extent) the cost of using the communications hardware. As the message size increased, more bits had to be copied to the communications board. As long as the communications hardware could transfer the bits it had faster than new bits could arrive, the only influence on the transfer rate was how quickly whole messages could arrive. But since datalink primitive are so close to the communications hardware, there was much less overhead due to headers and intra-task communications.

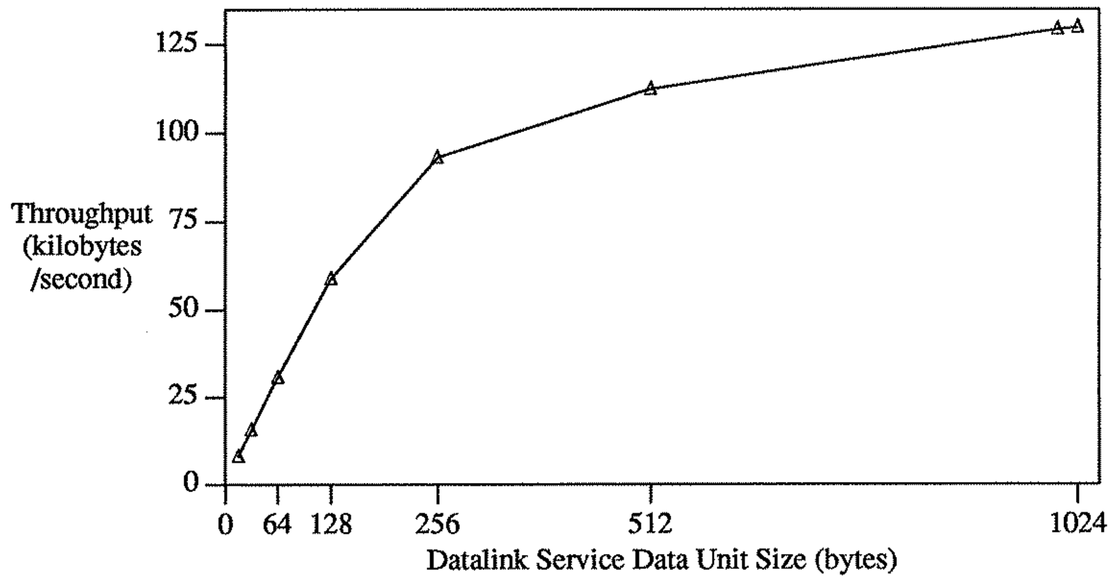


Figure 7. Throughput vs. Datalink Service Data Unit Size

Figure 8 shows the number of messages per second at the datalink layer. This time the curve is decreasing (in a somewhat linear fashion) rather than remaining constant. This also indicates that there is no longer a constant per-transfer cost associated with each message. Clearly the transfer rate of each message was affected by its length.

Figure 9 shows datalink, transport and CASE throughput curves. It is clear that as the CASE and transport message sizes increased, the efficiency of the protocol stack increased. This is due to the fact that pointers are passed between tasks rather than whole messages. Also, this suggested that the intra-task communications was very efficient, since messages at CASE required 5 tasks (layers) to communicate, whereas datalink only required one. This convergence to 130 kilobytes/second for each curve further suggests that even if CASE and transport did not have to segment messages into less than 1024 bytes, they each had almost reach the upper bound on the transfer rate. However, it should be noted that since segmentation is effectively removed from the front end processor because of the constraint due to the

Common Environment, the services provided by the upper layers that were actually exercised by these experiments were not very robust.

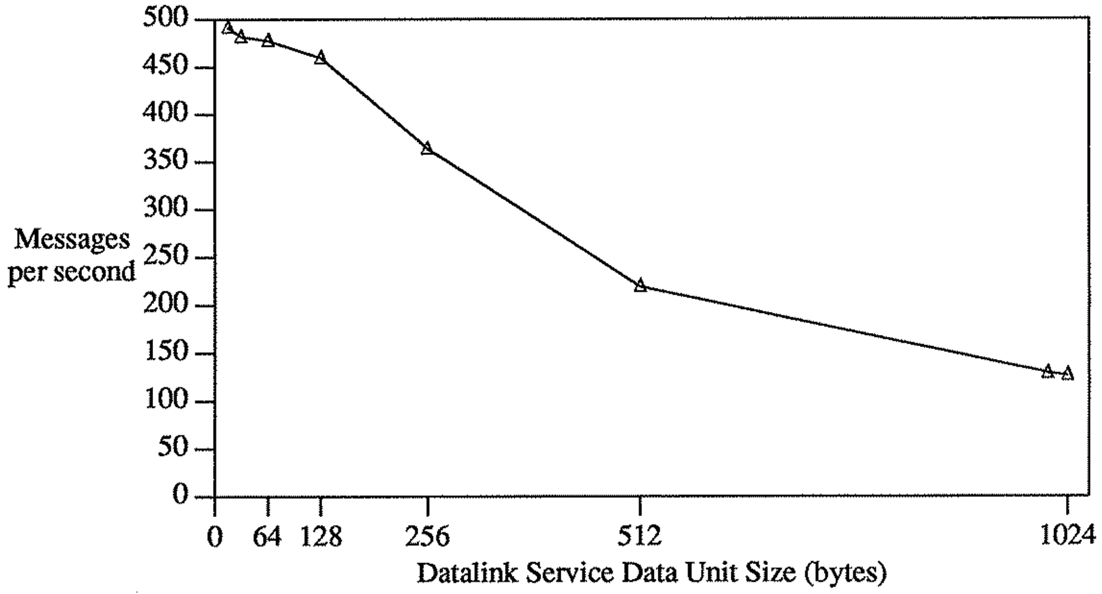


Figure 8. Number of Messages per Second vs. Datalink Service Data Unit Size

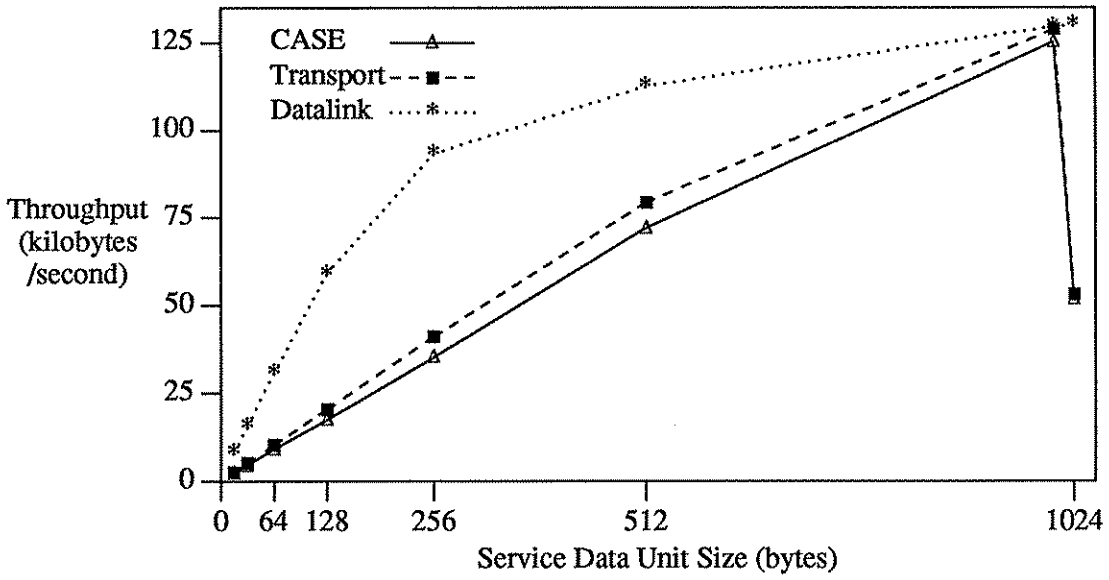


Figure 9. Throughput vs. Service Data Unit Size

Conclusions

Performance measurements were reported for data transfer at three layers in Motorola's implementation of MAP. The range of the message size was 16 bytes to 1024 bytes. We observed that by setting the EOT flag to true during data transfer at CASE and the transport layer, a stop-and-wait protocol resulted, whereas setting this flag to false for all except the last message sent resulted in a pipelined protocol. For the pipelined protocol, maximum throughput for CASE and the transport layer was about 125 kilobytes per second. We explained that the linear nature of the throughput graph, along with the constant nature of the number of messages per second, indicated that the rate of message delivery was really a per-transfer rate, and not a per-byte rate. We noted that the maximum datalink throughput was about 130 kilobytes, but that because the throughput curve was logarithmic in shape, there were both per-byte and per-transfer costs.