A Survey of Performance Issues in Communications Protocol Design and Implementation

Subir Varma

ABSTRACT

In recent years there has been a growing realization that as the speed of links has increased, the communication bottleneck has shifted to the protocol processing in the nodes. In response to this there has been an extensive amount of research to find ways to alleviate this problem. The main thrust of this work has been in the following directions:

1. Improve the performance of existing protocols by tuning their implementations.
2. Implement existing protocols in special purpose hardware.
3. Suggest new protocols which are better suited for high speed networks.

The objective of this report is to give a non-technical survey of this area, and point out the main performance issues that arise thereby.

ITIRC KEYWORDS

- Efficient protocols
- Implementations

1.0 Introduction

One of the important trends in the past few years has been the explosion In link capacities due the fiber-optic revolution. The prevalence of high bandwidth is been) driven by three forces [Ja):

- New high speed link technologies such as HIPPI and FCS which are capable of providing link bandwidths in the gigabyte range.
- New transmission technology services such as SONET. BISON, SMDS etc. that are designed to take advantage of the high speed links.
- Future applications such as distributed computing, full motion imag.es, multi-media etc. that will require throughputs in the gigabyte range.

IBM mainframes are ideally positioned to serve as platforms for high bandwidth applications due to their rich storage and connectivity features. However. before this can become reality. it will be necessary to radically· redesign the access method technology so that ii will be capable of sup• porting high bandwidth applications. There are a number of different approaches that have been suggested in response to this situation. The purpose of this report is to provide a survey of some of these efforts and also point out their relevance to VTAM.

A survey of the literature reveals that researchers have adopted three different approaches in their attempt to improve protocol performance. These approaches are summarized below in order of increasing deviation from conventional techniques:

1. The first approach is to improve protocol implementations. An examination or the code of protocols that are implemented on conven1ional mainframes reveals that only a small part of the code is spent in protocol related functions, the rest is spent in overheads such as data movement. scheduling, buffer management etc.. Hence protocol performance can be improved by reducing these overheads. This approach is covered In Section 2.
2. The second approach is implement protocols on special hardware platforms or front ends. The justification behind this approach is that since most of the protocol processing is made up of overheads, moving to a special environment that is optimized for protocol processing would reduce those greatly. This approach is covered in Section 3.
3. The third approach is to design new protocols that are better suited to high speed implementations than conventional protocols. This approach is covered in Section 4.

2.0 Tuning Existing Protocol Implementations

This section is organized as follows: We first list the suggestions that have been made to improve the implementations of existing protocols. If possible, we also give examples from papers were the suggestions were implemented or tested. In the last part-of this section we explore the limits of throughput which can be achieved by tuning alone.

Context switching:

The issue of context switching is closely connected with the problem of how to map the multiple layers in a protocol with processes in the operating system (PoCh].

One option would be to take each layer and run it as a process. The advantages of this approach are:

- The different functions or each layer can be kept within the process's context and hence provide a clean interface.
- Automated tools may be used to produce the code that implements the layer.
- It *may* even be possible to replace a layer by a layer or another protocol that pro- vides the same services.

The principal drawback of following this approach is the heavy performance penalty due to the context switching overhead from one process to another. Hence by combining multiple layers into the same process would improve performance. However there is a limit to which multiple layers can be combined into processes. and it is generally prefer- able to put layers performing similar functions together. In the context of the OSI layers, the application oriented session, presentation and application layers would be in one process, the network oriented transport and networ1< layers would be in another process and the datalink layer would be in a third process since it is usually implemented in the kernel. The concurrency between the application and the network layers is essential in order that the network layer may be able to handle both user requests from upper layers, as well as network events from lower layers.

A number of implementations, have tried to reduce the context switching overhead by providing their own scheduler, which has a fewer overhead compared to the operating system scheduler. Other advantages of this approach are that porting the protocol to another operating system becomes easier and work elements within the protocol can be assigned different priorities (for example control messages are assigned higher priorities than data messages). Inspite of the use of its own scheduler, about 20% of the code during SEND and about 12% during RECEIVE is due to the scheduling ,md dispatching overhead (for a 4K message size). This overhead is approximately equally distributed between protocol and OS schedulers. Hence a way must be round to reduce the scheduling pathlengths. One promising approach would be to examine the idea of Light Weight Processes or Threads [MaLeScMa] which have a much lower context switching overhead as compared to traditional processes.

Data movement:

It has been long recognized that copying data constitutes one of the most significant overheads in protocol processing since i1 involves the touching of each byte in the message. Furthermore. in large systems it has been noted that excessive data movement indirectly decreases the performance or another address spaces, by increasing the cache miss rate (HaPo). A number or techniques have been adopted to overcome this problem:

- Pass buffers in-between layers by reference pointers, rather than by copying. Space for headers can be reserved in the buffer itself. Alternatively, or each

layer may construct its header separately, and when the packet is ready to be trans- mitted, these elements are gathered together into a single buffer.

- The solution suggested above does not solve the problem of having to move the data from pageable user storage to fixed system storage. One solution that has been suggested for the S/370 environment, is page table manipulation as in CLAW [AnDaCIEdRa]. The basic idea in this technique is to adjust the users page table entries to point to the data, rather than actually moving the data. However, this can only be accomplished if the 1/0 buffers have a fixed size of one page. Another solution is a common storage manager (CSM) [Io]. The intent of this technique is to transfer data from storage devices to a data space that is shared among several address spaces including that of the Protocol stack. Hence it .is possible to build a channel program and send the data directly from this address space, without having to do another move.

Exploiting protocol parallelism:

The following classification of the types of parallel processing pos- sible in a protocol implementation, Is a condensation of the ideas of Jain et.al. (JaScBa] and Zitterbart [Zi].

- **Coarse grain parallelism**: This involves assigning tasks belonging to different sessions to different processors. Contention between multiple processors is eliminated because activities on each connection are independent of the others. However note that the concurrency which can be achieved is limited by the number of sessions that are active simultaneously.
- **Medium grain parallelism**: This involves assigning successive packets to the next available processor, even if they belong to the same session. This scheme is capable of significantly improving the throughput of a single session. However the interference among multiple processors is increased since packets belonging to the same session are likely to reference the same data structures. Moreover the packets may have to be resequenced before being delivered to the upper layers.
- **Structural parallelism**: This is derived directly from the layered protocol stack model. Pipelining of packet processing so that several packets from the same session undergo processing in different stages of the pipeline at the same time is an example of structural parallelism. Each pipeline stage can be mapped lo one or more layers in the protocol stack. The fact that the SEND and RECEIVE for a single session can be executed in parallel is another example of structural parallelism.
- **Fine grained parallelism**: This involves using several processors simultaneously to process a single packet. For example in the context of pipelined processing discussed above. if a layer were to-be split up into a number of parallel processes, then it would be an instance of fine grain parallelism. This idea is closely related to the idea of integrated layer processing in Section 4. In a pipelined implementation, fine grained parallelism can be used to reduce the processing delay of the slowest stage in the pipeline.

Header Prediction:

The basic idea behind header prediction [CIJaRoSa], is the following:

On SEND side, is to keep a fixed template of headers (which is the same for every packet), and attach it to the outgoing packet. The packet dependent parts of the header such as the sequence number are obtained and attached separately for each packet On the RECEIVE side, pre-compute what values should be round in the next incoming packet header, and then a few simple comparisons suffice to complete the header processing. In an experiment carried out at Cray Research [NiGoBoYoRo], .the authors observed that the throughput went up from 394 Mbits/s to 423 Mbits/s on an 800 Mbits/s hostto host channel connection, because of the header prediction algorithm.

The concept of header prediction has been generalized to that of Protocol Bypass by Woodside et. al. [WoRaFr). They make the point that in most complex systems, 80% of the code is devoted to processing special cases and only 20%of the code is sufficient for executing the mainline functions. Applying this concept to communication protocol implementations, Woodside et. al. provided a bypass test at the top of the SEND stack and the bottom of the RECEIVE stack. If a packet satisfied the bypass test, then it was executed by a special fast path, while if it fails the test, then it is executed by the usual protocol stack. Interestingly enough, they observed that an OSI protocol stack implemented with bypass mechanism on a uni-processor, out performed a parallel implementation of the protocol on 4 processors without bypassing.

Caching information:

The SEND and RECEIVE operations in most protocols involve control block searches in several places. Clark et. al. [CIJaRoSa] in experiments with the TCP/IP protocol. have observed that the use or a cache with just one entry can reduce the search overhead substantially. In fact the hit ratio to this cache, even for traffic to a diverse set of connections, approached 90%.

Interrupts:

The overhead or interrupts from the channel can be substantial. One way to reduce it, as VTAM has done. would be to implement a timer mechanism, whereby messages arriving within the timer interval are blocked and transmitted together thus generating a single interrupt for the entire block. This solution is able to achieve high throughput, especially in heavy message traffic, but it increases the amount of transmission delay that individual messages experience. One solution to this, is to have threshold message size such that all messages smaller than the threshold are transmitted immediately, while messages larger than the threshold are blocked using the timer mechanism. The justification behind this approach is that smaller messages would be from interactive response lime sensitive sessions: while larger messages would be from non-interactive throughput oriented sessions.

We now describe the concept or a never ending channel program, that has been adopted by the designers of CLAW. This is designed to reduce the interrupt overhead significantly. The READ channel program operates as follows: The basic idea is to use a list of READ CCWs and an array of flags, with one flag associated with each CCW. When a READ CCW completes normally, then the corresponding flag is set by the control unit. Thus CLAW can keep track of the channel programs progress without receiving interrupts. When CLAW discovers that READ CCWs have completed normally, then it appends the READ buffers to the end of the CCW chain.

In this way, a finite number of READ buffers are kept circulating continuously. At times when
The host goes into a wait state, it explicitly asks the channel for a PCI interrupt at the end of

the CCW. The WRITE channel program operates basically operates along the same lines. There is a list or WRITE CCWs and array of flags, with one flag associated with each CCW. When a WRITE completes normally, then the corresponding flag is set by the control unit, thus avoiding an interrupt. As new data arrives, WRITE CCWs are built for them, and appended to the end of the CCW list. In this way the WRITE channel program can be kept operating continuously. Moreover there is also a facility for CLAW to receive a PCI interrupt every nth frame, when it can check whether previous WRITE CCWs have completed normally, and also to append new CCWs if necessary.

Locking Mechanisms

Since, communication software by its very nature has a high level of concurrency, locking mechanisms play a significant role in its performance. There are two broad varieties of locks, spin and suspend. In most concurrent programs, either one or the other is exclusively used. however recent research in operating systems has revealed that using a combination of the two would be more beneficial than using either alone [KaliMaO]. In the case of suspend locks, the significant overhead is the cost of a context switch, while in the case of a spin lock the significant overhead is the waste of processor resources. A middle path would be to spin for a while. and then suspend if the lock is still not available. Note that the number of cycles spent spinning should not exit C (the number of cycles required to do a context switch).

Karlin et. al. present a number of different alternatives for choosing !he threshold T, which should be spent spinning. The discovered that adaptive algorithms, in which the T is dynamically adjusted perform better, than algorithms in which T is fixed, and both perform much better than algorithms in which T is set to 0 or infinity. The best adaptive algorithm is known as random walk and it operates as follows: If number of spins spent waiting for the lock is more than C then decrease T by 1, otherwise increase T by 1 up to a maximum of C. They observed a decrease in elapsed time of 32% by using random walk, as compared to a pure block policy and 56% over a pure spin policy.

Buffer management:

We describe the way in which buffers are in managed environments. There is pool or fixed size buffers in fixed storage, into which the message is transferred before being transmitted. Not more than one message can be put in a single buffer and large messages *may* span multiple buffers. The buffer pool is capable or expansion and contraction, such that if the number of free buffers decreases below a threshold, then the pool is expanded by a fixed amount Conversely if the number of free buffers exceeds another threshold the pool is contracted by the same amount During expansion, additional buffers are obtained by means of OS calls, which can be very expensive.

The use of fixed size buffer leads to a more efficient implementation, however it may lead lo wasted buffer space if the messages are significantly smaller than the buffer size or it may lead to long channel programs if the messages are significantly bigger. Moreover, both extremes in message sizes are usually mixed in the same traffic stream, since control messages are smaller than data messages. Solutions to this problem may include monitoring the traffic now and automatically changing the buffer size and also multiplexing several small message-s in the same 1/0 buffer. The buffer pool expansion/ contraction algorithm may also benefit by changing from linear expansion/contraction to exponential expansion and linear contraction.

The I/0 buffers in CLAW are fixed at 4096 bytes in size, and this size cannot be varied by the user. The designers chose this size in order to implement their data movement scheme by means or page table manipulation. Another notable feature of their buffer management scheme is the inter-mix buffers belonging to different messages together so that a single buffer belonging to a small message may be put among the buffers of a large message. The advantage of this scheme is that the small message will

not have to wait for an excessive amount of time, for the large message in front of it to be transmitted.

Queue management

It is necessary to maintain queues of control blocks in most protocol implementations. If the queue size becomes too large, then retrieving control blocks may become an expensive operation [Co]. There are several ways in which this overhead may be reduced:

- If the control blocks are added and deleted only from the end of the queue then double headed queue is indicated. If several processes manipulate the double headed queue, then a lock is required to preserve its integrity. To avoid using the lock, one may use a single headed queue and reverse the queue pointers at the time when a control block has to pe dequeued.
- If the control blocks have to be searched from a list, and there is a key associated with each block, a more efficient data structure such as a hash table or an AVL tree should he used.

It would be interesting to estimate the minimal number of instructions required to implement a protocol, provided all overheads due to the operating system, data movement. buffer management etc. were removed. This would provide an intrinsic upper limit to the speed of the protocol, independent of the environment. ClarK et. al., (CIJaRoSa), have carried out precisely this exercise for the TCP/IP protocol. They were able to show that it is possible to implement the SEND or RECEIVE operations for a single packet, by using about 300 CISC instructions or about 400 RISC instructions. Assuming that the packet size is 4000 bytes, the code is run on a 10 MIPS processor and the TCP/IP processing overhead is the bottleneck, this translates to a throughput of 530 Mb/s after taking acknowledgment overheads into account. Wicki [Wi] estimates that the total number of instructions after taking all overheads and the device driver code into account would be about 2000. which translates into a throughput of 16.0 Mbits/s.

3.0 Implementation in Special Purpose Hardware

We concluded in the last section that if conventional protocols are to sustain gigabyte transmission rates, then the system ·overheads need to be cut down drastically, as well as processors with a largej ty,IPS rating would be required. An alternative approach is to implement the protocol using an board connected to the main processor. The processing environment in the outboard would be optimized. for protocol processing, thus cutting down on the system overheads. In the LAN arena, several products have been available for the past few years that Implement the MAC and lower layers in a front-end board [KrSa). Some of them also perform the front-end functions of their link level protocols. The intent of most of the work described in this section, is not only to implement the lower layer functions in hardware, but also upper layers such as the network and the transport, thus achieving a greater degree or independence from the main processor.

There are three important issues that have to be addressed if a front-end protocol implementation is to show significant performance gains over host implementations:

- The interface between the host and the front-end: If this interface is poorly designed, then the host will not be able to realize the performance gains in offloading the protocol.
- Taking advantage of the concurrency in the protocol: Since context switching in a front-end will be less

expensive than that in the host, there are greater variety or parallel implementations that become feasible. By taking advantage of parallelism between protocol layers within protocol layers and between successive messages, it would be possible to achieve gigabyte speeds by using a sufficiently large number of processors each with a modest MIPS rating.

- Designing the memory structure in the front end: All the issues raised in the previous section regarding data movement apply to the front end too, so that it has to be designed carefully to reduce this overhead.

In the rest of this section, we present the techniques that have been used by some front-end implementations to address the issues presented above. The specific front end implementations that we will discuss are:

- The VMTP Network Adaptor Board (NAB) [KaCh] from Stanford University. The NAB was designed to implement VMTP, a new transport protocol specifically created for efficient implementation in a high performance network adapter. The adapter uses a host interface that is designed for minimal latency, minimal interrupt processing overhead and minimal system bus and memory access overhead. Measurements show that the NAB with an onboard processor rated at 2 MIPS, is capable of a throughput of 44.3 Mbits/s per connection on a host-to-host connection, with a packet size of 1 Kbyte and user message size of 16 Kbytes. Delay measurements for small packets showed that only 5.9% of the response time was due to NAB processing and bus transfers, whereas 88% was due to the host. With the onboard processor ratedt 20 MIPS, the NAB achieved a throughput of 88.6 Mbits/s.

- Transputer based front-ends from IBM Zurich (TRAZ) [Wi] and the University of Kahlrusche (TRAK} [Zi]. The main objective of these projects was to demonstrate the feasibility of a high performance implementation of OSI using a general purpose processor such as the transputer. They sought to do so by exploiting the structural and fine grained parallelism in the protocol layers. Measurements were performed on TRAK, for an implementation or the OSI Network protocol and the OSI Transport protocol (TP4). Without including the checksum and segmentation free assembly, a throughput of about 60 Mbits/s was reported using S Kbyte packets. An implementation of the LLC layer in the TRAZ front end showed a throughput of 100 Mbits/s for a 2Kbyte message size.

- The MPP front end from Columbia University [JaScBa]. An architecture is proposed for accomplishing transport protocol processing at Gbps/s rates. The key concept is that of processing packets on distinct processors in parallel. The architecture assumes suitable hardware support to perform any necessary lower level protocol processing but makes no significant assumptions
  about the protocol to be implemented in the system. The authors carried out a throughput analysis based on the architecture (since an implementation was not yet available). and concluded that Gbits/s rated would be possible with about 5 processors.

- The AX9t-J host-network architecture [StPa]. This project is especially interesting since it seeks to optimize a mainframe environment for data transfer. The architected another network interface to provide a path directly between the network and host memory, without the intervention of a channel subsystem, thus delivering high bandwidth directly to applications. Another significant aspect of this architecture is the support provided for virtual shared memory on loosely coupled systems, i.e., paging is supported across multiple hosts.

- The Protocol Engine (PE} front end [Ch]. The PE is designed to implement in VLSI, the XTP protocol which combines the functions of transport and network layers into a single layer, called the ransfer layer. XTP was designed for LANs, MANS and WANS with a suitability for VLSI implementation. The initial implementation of PE was designed for 100 Mbi1s/s FDDI LANs and a later version is planned to reach 1 Gbits/s.

Host to front-end interface:

One of the important issues involved in designing an efficient host to front-end interface is movement of data from the host to the front-end. There are two main techniques used for moving data from the host:

- The host reads data from and writes data to the front end. The disadvantages of this approach are: The host wastes processor cycles in moving data, the- data passes **through -the cache and corrupts it and two bus cycles are required for each byte** transferred (from memory to processor and from processor to front end). The advantages are: If the host has to do additional processing such as encryption of check-summing. it can do it while moving the data and it can move data that is dispersed in memory to the front end and conversely disperse data from the front end to various locations in memory (adding this scatter-gather capability to the front end increases its complexity).
  - The host passes a descriptor control block to the front end that has pointers lo locations in memory where the data is to be found, and a special OMA mechanism in the front end actually transfers the data. The advantages of this approach are basically the disadvantages that are listed for the host based approach. The main disadvantage is the additional complexity in the front end.

The NAB project combines these two approaches by making the host directly write to the NAB board for short messages (that can fit in the fixed size packet header), and passing pointers to main storage to the NAB for long messages. For the second case the NAB has a high speed block copier that transfers data from the memory in a single cycle. The AXON front end has a similar mechanism for transferring data to and

from the host. In the TRAK and TRAZ projects, the host writes the data to the front end memory and passes pointers pointing to that data to the front end and this has been recognized by the authors to be a weakness in their implementation. The MPP front end uses an OMA to transfer to and from the host. A novel feature in this architecture is the presence of two internal buses in the front end. One of these buses is used for high speed data transfers into the front end (from the networks) and the host bus. while the other bus is used lo provide access to shared memory for all the processors in the system.

Another important issue involved in designing a host to front end interface is the frequency of interrupts to the host. Since this is one of the disruptive influences to the host that we would like to remove by offloading the protocol, its design is especially important. The NAB implementers presented only one interrupt to the host for every message received or transmitted. Since a large message may contain multiple packets, the frequency of interrupts for that case is reduced. The implementers of TRAZ, TRAK and MPP do not talk about this issue in their papers. A constructive suggestion in this regard would be to borrow the interrupt avoidance strategies of CLAW in this environment, i.e. provide an array of flags in the host that would be set by the front end as it writes or reads data. The host needs to be interrupted only if it has gone to sleep.

Exploiting protocol parallelism:

By implementing a protocol on front end, we increase the opportunities of exploiting concurrency since the cost of a context switch is smaller than in a host, and a large number of processors can be dedicated to the task of protocol processing. In the previous section we classified the types of parallelism found in protocol processing and we now discuss how some of them are exploited by front-ends.

In the NAB and AXON front ends, structural parallelism in the transport layer was exploited by implementing a pipeline to carry out the byte intensive functions such as check-summing and encryption/decryption. Furthermore, there were separate pipelines for the SEND and RECEIVE operations. The TRAZ and TRAK front ends implemented the protocol layers with transputers providing another example of structural parallelism. For example in TRAK, there was one transputer for the API, two transputers to implement the transport layers for the SEND and RECEIVE, and another transputer to implement the lower layers. Furthermore, fine grain parallelism was also exploited by splitting up a process in *a* single transputer into a number of processes that can be executed in parallel (since the cost of doing context switching in a transputer is minimal, this is feasible). In the MPP front end, medium level parallelism was exploited by putting several processors on the front end lo carry out the transport level functions, and routing an arriving packet in a cyclic round-robin fashion to each one of the processors, irrespective of the session !o which it belongs.

Memory design of the front end; The appropriate memory management technique for the front end depends upon the type of parallelism that is exploited in the implementation. Most conventional front-end implementations for the MAC layer in LANS employ a bus based architecture for access to the onboard memory by the resident processor, the OMA channel to the host and the network interface. At high speeds, this bus can become the system bottleneck and constrain the amount of parallelism that can be exploited. An alternative to this design is a multi-ported memory [Si], with one port devoted 10 each interface. We now briefly review the memory design of the front ends being examined.

The NAB front end has a single common buffer for sending and receiving packets. It is implemented in a Video RAM and can be accessed in parallel by the DMA controller and the on board processor. Hence processing of a packet can proceed in parallel while another packet is being received or sent. The AXON architecture provides a similar mechanism. However it also provides an option for transferring data directly to the network from the host memory without any buffering in the front end. This is possible only if the network transmission rate, the host to front end transmission rate and the front end transmission are rate matched to each other.

- Since multiple processors could process the same packet in a pipelined fashion in the transputer based implantations TRAZ and TRAK, it was necessary to provide global memories that could be accessed from all processors. This avoids expensive data movement from processor to processor. Accordingly two global memories were provided, one for SEND and the other for RECEIVE. The SEND and RECEIVE processor pipelines receive pointers to the location of the data units in the memories and they process the packet headers, but they do not move the data. Another global memory for storing connection and state information required for the SEND and RECEIVE pipelines is used. Furthermore. each processor has a private memory in which code and locally required data structures are stored.
- The MPP front end does not require a global memory for transmitting and receiving packets, since each packet is processes entirely by a single processor. Hence the packet resides in the processor's local memory. However a small common memory is required for storing the context records for each session. Another common memory is required to required to manage the resequencing of packets before they are passed to the host.

Note that the intent of all the front-end processor projects described in this section has been to offload the transport layer from the host. This is because in protocols such as OSI or TCP/IP, the transport layer is a major bottleneck. especially if it provides connection oriented services with flow control, error recovery, segmentation etc. In SNA however, the transport layer has much smaller functionality, in fact it is lumped with the network layer to form the path control layer. Hence if SNA is to gain any significant benefit from front-ends, it will be necessary to offload the transmission control layer and the datalink control layers. (which correspond-to the session layer in OSI) in addition to the path control layer, to the front-end.

4.0 New Protocols

Most of the innovations that have been suggested for new protocols, are in the area of transport layer design. Some of the· services provided by the transport layer are connection management, flow control, error handling and acknowledgement of data at ·the receiver. Most of these functions assume that the network and data-link layers are connectionless and unreliable. However, IBM protocols such as SNA and APPN operate under a virtual circuit oriented network service and a fully reliable data link transmission. Hence except for end-to-end connection management, they do not provide any other service to the upper layers. (The application layer is responsible for detecting end-to-end errors, and on detection, the connection is aborted. This happens relatively infrequently since the network layer never discards packets, and the data-links error recovery is effective most of the time [GrHaHolePo]). The new transport protocol called Rapid Transport Protocol (RTP). which being proposed for Autobahn Is closer to conventional protocols. in fact it has the same functionality as ISO/TP4 (GaMa). Many of the suggestions made below to improve transport protocol performance have been incorporated into RTP.

Connection management: The presence of a connection implies the existence of state information regarding the data transfer in the transmitter and the receiver. In a connection oriented transport layer, upto 3 handshaking signals may be required to start or end a connection. The opening packet exchange guards against opening due to duplicate packets and allows resource negotiation; the closing exchange assures that all data have been received, and both parties are prepared to close. This is inefficient for cases in which the message to be transmitted is small and/or the propagation delay between the two end- **points is large. An alternative that has been suggested is timer based connection management** (WaMa). which open connections when the first packet is received. and close **connections under timer control. The main** problem wi1h this approach is determining timer intervals and bounding packet lifetime. Another approach that has been adopted by APPC is to keep a pool of active connections always ready, and assign a conversation request to one of these connections when requested.

Packet organization: There is a close relationship between packet organization and the ability to exploit parallelism while processing the protocol. For example, in the NAB front end for the VMTP protocol, we noted that the checksum was computed in a pipelined fashion while the packet was being moved in or out of the main buffer. This was made possible by placing the checksum in the trailer of the packet, rather than in the header as most conventional protocols do. Other aspects of packet organization than can speed up processing are: All fields within a packet should be in fixed places and of fixed length, the boundaries of fields should fall on byte or word boundaries and the fields in the header should be so organized so as to facilitate parallel processing.

Flow control: In order to adapt traditional window based flow control techniques to high speed links, two changes had to be made: The size of the window had to be increased because a larger number of packets could now be transmitted during a time interval, and secondly some kind of rate control had to be introduced at

the interface between the source and the network. This is because the size of the window only indicates the amount of available buffer space at the receiver, not the rate at which the network can transmit !hose packets. This rate is negotiated at the lime of connection establishment, and is either specified in terms of a burst rate and a time interval, or an inter-packet gap. Mechanisms such as the leaky bucket are used to enforce the agreed upon rate.

Error recovery: The classical method of error recovery at the transport layer in conventional protocols is by means of timers at the transmitter. The receiver only returns ACKs for successful packets receives in sequence, and so when a timer for a packet expires, all packets succeeding it also have to be transmitted. This scheme leads to redundant re- transmissions and loss of network capacity. Two new schemes that have been suggested to solve this problem are packet blocking as in NETBLT and periodic state exchange [NeSa]. The packet blocking scheme implements a single timer in the receiver for a large buffer containing several packets. When the timer expires or any of the packets are received in error, then the receiver sends a list of these packets to be re- transmitted to the transmitter. Periodic state exchange-tries to eliminate the use or timers in either side, by exchanging complete state- information periodically and independently of the state of the protocol.

Integrated layer processing (ILP):
The idea of ILP is from the paper by Clark and Tennenhouse
[CITe]. They make the point that the naive implementation or a layered protocol suit involves the sequential processing of each unit of information, as it passes through the layer entities of the transmitters and receivers protocol stack. !LP captures the idea that
the protocol be so structured as to permit the implementer the option of performing all the manipulation steps in one or two Integrated processing loops, instead of performing them serially. For example session specific encryption operations can be entwined with data link level operations.

A similar idea has been proposed by Zitterbart and Tantawi [Zi], [ZiTa] and Haas [Ha]. They propose that the transport component no longer be designed in terms of protocol layers but as a set of protocol functions. Several of these functions can be performed in parallel and to guarantee that all the functions are performed before the packet is transmitted, a synchronization point should be provided. Also each application would be allowed to choose a set of functions appropriate for it, rather than go through the entire stack. This technique would be especially useful. in supporting multi-media applications. since each traffic stream such as voice, data. video etc. may have its own requirements.

Application fever framing: Clark and Tennenhouse point out that presentation processing by the application layer is a bottleneck in the overall network throughput. In order improve its performance, the application should be allowed to perform presentation conversion as soon as data arrives from the network. However lost or re-ordered data prevents this from happening. If the application is to have the option of dealing with the lost data unit, then losses must be expressed in terms meaningful lo the application. One way to achieve this would be for the application to break the data into suitable aggregates and the lower levels preserve the frame boundaries as they process the data. This concept is known as application level framing.

Transport protocol support for voice/video traffic: Many of the traditional transport protocol support provided for data applications such as error control and recovery, window based flow control etc. become useless in an environment where the traffic consists of voice and/or video packets. For this new environment Anastasi el. al.. have defined a new transport protocol called TPR (transport Protocol for Real-Time Services) for an FDDI environment, which provides the following new types of services:

- Delay jitter compensation at the receiver: This mechanism is used to eliminate the variability of end-to-end delay for successive packets. It consists or delaying each packet by an amount so that the end-to-end delay is the same for all packets.
- Drift correction mechanisms at the receiver: This mechanism is used to correct the drift between the transmitter and receiver clocks. This issue becomes important at high bit rates, when a mismatch can lead to overflow or underflow or the transmitting buffer.

Lost packet compensation at the receiver: Since lost packets are not re-transmitted for real time traffic, they are compensated by inserting a dummy packet in place of a missing packet

REFERENCES

(AnHrMePaRo) H.A. Anderson, O. Hrelic, C.B. Meltzer. E. Paradise and R.E. Rose 'Common link access to workstations (CLAW): Preliminary draft', IBM *Research Report,* (Oct, 1989).

[AnPo] H.A. Anderson and P. Poon, 'VTAM V3R2/3090 system performance evaluation in a data server environment'. IBM *Research Report,* (Apr., 1990).

[BrlyI'Ie] R.K. Brandriff, CA. Lyocn and M.H. Needleman, 'Development of TCP/IP for IBM/370', *ACM Computer Communications Review,* 2"8, (1985).

[CI) **[CI] D. D. Clark 'The structuring of systems using upcalls ACM *Computer Communication's Review,*** 171-180, (1985).

(CIJaRoSa) D.D. Clark. V. Jacobson, J. Romkey and H. Salwen, 'An Analysis of TCP Processing Overhead', *IEEE Communications Magazine,* 23-29, (June, i989).

(CoArMi) R. Collela, R. Aronoff and K. Mills, 'Performance improvements for ISO transport',

[Co] L Corbet, 'VTAM performance guidelines', *Manuscript,* 23-29. (Undated).

[FIChEf] A. Fleischmann, S.T. Chin and W. Effelsberg, 'Specifications and implementation of an ISO session layer', *IBM Systems Journal,* 255-275. (1987).

[Jac] V. Jacobson, 'TCP Header Prediction . *ACM Computer CommunicationsReview_,*13-15, (1990).

[Ja] J.M. Jaffe, 'High bandwidth attachment task force', *Manuscript,* (1991).

[KaUMaOw] A.R. Karlin, K. Li, M.S. Manasse and S. Owicki, 'Empirical studies of competitive spinning of a shared memory multiprocessor', *IBM Research Report,* {Apr., 1990).

(KaTo) H. Kanakia and F. Tobagi, 'Performance measurements of a data link protocol, *Proceedings of* ICC *1986,* 8$4-898, (1986).

[LaNoTh] KA Lantz. W.I. Nowicki and M.M. Theimer, **'An empirical study of distributed application** - performance·, *IEEE Trans. on* Soft. *Eng?* I 162-1174, (1985).

[Lo] L Long, 'VTAM HPT: Transport Mechanism Interface·, IBM *SLDIPLD Report,* (Feb,1991).

[MaLeScMa] B.D.Marsh, T.J.Le8lanc. M.L Scott and E.P. M rk3tos. "First class user level threads' *Proceddings of the* ACM *Symposium on Operating Systems Principles,* 110-121,{1991).

[NiGoBoYoRo) A. Nicholson, J. Colio, DA. Borman. J. Yovng and W. Roiger,'High speed networking at Cray **Rescarchi'" *ACM Computer Communications Review,* 99-110, (1990).**

[Pa] C.Partridge, 'How slow is one gigabit per second·, 44-53, (1988).

(PoCh] G.S. Poo and B.P. Chai, 'Modularity versus efficiency in OSI systems implementations', *Proceedings of ICC 1991,* 950-959. (1991).

[PoAnJ G. Poo and W. Ang, 'Cut-through buffer management technique for OSI protocol stack', *Computer Communications,* 166-177, (1991).

[SaCIJ J.H. Saltzer and D.D. Clark, 'End-to-end arguments in system design·, *ACM Transactions on Computer* Systems, 27-7-288, (Nov., 1984).

(SaCIRoGr] J.H. Saltzer, D.D. Clark, J L Romkey and W.C. Gramlich; 'The desktop computer as a network participant', ***IEEE*** *JSAC,* 468-478, (1985).

(Sv) L Svobodova, ·Implementing OSI systems', *IEEE JSAC,* 1115-1130, (Sept, 1989).

[TaScMelaAu)' . Tantawy, T. Schuen. H. Meleis. R. LaMaire and R. Auerbach, 'High performance protocol implementations: LLC case study', *IBM Research Report No. RC* 15850, (1990).

[WoRaFr] C. Woodside, K. Ravindran and R. Franks, 'The protocol bypass concept for high speed OSI data transfer', *Protocols for High Speed Networks,* 11, 107-122, (1990).

(Zi] M, Zitterbart, 'High speed transport components', *IEEE Network Magazine,* 54-63, (Jan, 1991).

5.1.1 Hardware Implementations

(Ch) G. Chesson, 'XTP/PE Design Considerations', *Protocols for High Speed Networks, H. Rudin and R. Williamson editors,* 27-33. (1989).

[KrSa] **A.S. Krishnakumar and K. Sabnani, 'VLSI implementations of communications protocols- A** survey', *IEEE* JSAC, 1082-1090, (1989),

[DaLaMaBeVo] G.T. Davis, R.E. Landa, B.O. Mandalia, J.W. van den Berg and D.C. Van Voorhis. .,Specialized communications processor for layered protocols.., ***US Patent no. 4991133,*** **(1991).**

(Cl) **G.G. Finn, "An integration of network communication with workstation architecture',** ***ACM Computer Communications REview,*** **18..-29, (Jan, 19.91).**

(JaScBa] **N. Jain. M. Schwartz and T.R. Bashkow. 'Transport protocol processing at Gbps Rates'** ***ACM*** SIGCOMM, 188-199, (1990).

[KaCh) H. Kanakia and D.R. Cheriton, 'The VMP Network Adaptor Board: High performance network communications for multiprocessors·, *ACM SIGCOMM,* 175-187, (1988),

[Si) M.A. Sidenius, ·Hardware support for implementation of transport layer protocols·, *Protocols for High* Speed *Networks,* 1251-267, (1990),

(Sk) M. Skov, 'Implementation of physical and media access protocols for high speed networks·, *IEEE Communications Magazine)* 45..53, {June, 1989).

[StPa] J. Sterbenz and G. Parulkar, ·Axonhost-network interface architecture for giga-bit communications·, *Protocols for high speed networks, 11* 211-236, (1990).

[TeYoHiMa] **M.** Terada, T. Yokoyama; T. Hirata and S. Matsui, 'A high speed protocol processor to execute OSI'. *Proceedings of ICC 1991,* 944-949, (1991).

[Wi] T.M. VVickJ, "A multi-processor based controller architecture for high speed communication protocol processing', *IBM Research Report RZ 2053,* (1990).·

5.1.2 New protocols

[AnCoGr] G. Anastasi, **M.** Conti and E. Gregori, "TPR: A Transport protocol for real time services in a FDDI Environment', *Protocols for High Speed Networks,11* 313-331, (1990).

[ChWi) D.R. Cheriton and C.L Williamson, 'VMTP as the transport layer for high performance distributed systems". *IEEE Communications Magazine,* 37-44, (1989).

(CILaLh] D.D. Clark, M.L L.:imbert and L Zhang, 'NETBLT: A high throughput transport protocol', *ACM* SIGC0MM, 353.;)59, (1988).

[CITe] D.D. Clark and D.L Tennenhouse, ''Architectural considerations for a new generation of protocols', *ACM* S/GC0MM *1990,* 200-208, (1990).

[DoDyKa] A.Doeringer, D. Dykeman. M. Kaisersworth. B.W. Meister, H. Rudin and R. Williamson. 'A survey of light-wieght transport protocols for high speed networks·, *IEEE Transactions on Communications,* 2025-2039, (November, 1990).

[GrHaHolePo] J.P. Gray, P.J. Hansen, P.Homan, M.A. Lerner and M. Pozefsky, ·Advanced program-to-program communication in SNA', IBM Systems *Journal,* Vol.22, No. 4, 298-318, (1983),

(Ha) Z. Haas: **'A** communications architecture for high speed networking', *IEEE Computer* ***Networking*** Symposium, 433-441, (1S90).

[LaSc] T.F.LaPorta and M. Schw3rtz., "Architectures, Features and Implementations of high speed transport protocols', *IEEE Network Magazine,* pp, 14-22. (May, 1991).

(SaNe] K. Sabnani and A Netravali, 'A high speed transport protocol for datagram/virtual circuit networks', *ACM SIGCOMM,* 146-157, (1989).

(TaMeZaRa] A Tantawi, H.Meleis. M.E. Zarki and G. Rajendran. 'Towards a high speed MAN architecture'. *Proceedings of ICC 1989,* 619-$24, (1989).

[WaMa] R.W. Watson and S.A. Mamrak. ""Gaining efficiency in Transport services by appropriate and Implementation choices', *ACM Transactions on Computer* Systems, 97-120, (1987).

(ZITa] M. Zitterbart and A.N. Tantawl. 'Transport service and protocols for high speed networks', *IBM Research Report No. RC 17074* (1991).

5.1.3 Measurements

[CoCoCaCa] M.H. Comer, M.1/1/. *Condry,* S. Cattanach and R.Campbell, 'Getting the most for your megabit', *ACM Computer Communic,,tionReview,* 5-12, (1991).

(CaHuKaMo] L Cabrera, E. Hunter, M.J. Karels and DA Mosher, 'User Process communication performance in networks of computers' *IEEE Trans. on Soft. Eng.,* 38-53, (1S88).

(GuBjNoPiSjStJ]. P. Gunningberg, M. Bjorkman, E. Nordmark, S. Pink, P. Sjodin and J.E. SIromqui,I, 'Applications protocols and performance benchmarks', *IEEE Communications Magazine,* 30-36, (June,1989).

[HeSt] S. Heatley and 0. Stokesberry, 'Analysis of transport measurements over a LAN', *IEEE Communications Magazine,* 16-22, (June,1989).

[StJ P.Strauss, 'OSI throughput performance: Breakthrough or bottleneck', *Data Communications,* 53-56, (1987).

[Sv] L Svobodova. 'Measured performance of transport! service in LANs' *Computer Networks and ISDN* Systems, 31-4S, (1988).