XTP and VMTP on Multiprocessor Architectures

Torsten Braun, Burkhard Stiller
Institut für Telematik
Universität Karlsruhe
Am Zirkel 2
D-W7500 Karlsruhe 1
Phone: +49 +721 608-[3982/3983]
Fax: +49 +721 60 60 97
E-Mail: [tbraun|stiller] @ ira.uka.de

Martina Zitterbart
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598, U.S.A
Phone: +1 914 784-6952
Fax: +1 914 784-6219
E-Mail: zit @ watson.ibm.com

Abstract

The processing of data units inside a communication node became to be the bottleneck in communications, because protocol processing did not increase in the same order of magnitude as the networks' bandwidth. For that reason, new and efficient approaches to surmount this bottleneck have been proposed. First, the development of new protocols and, second, the use of dedicated hardware platforms have a number of advantages. Throughout this paper the integration of these advantages will be presented using the light-weight protocols XTP [1] and VMTP [2] on multiprocessor architectures. Measurement and simulation results proof the performance gain of the selected approach. The results show that XTP and VMTP are more appropriate for implementations on parallel architectures than comparable OSI protocols.

1. Introduction

In communication systems during the last few years, transmission rates of communication networks have evolved faster than the performance of protocol processing. Often nowadays, the bottleneck in communication systems is the protocol processing of layers above the media access layer.

Recently, applications such as multi-media-applications are coming up, which require very high performance communication systems. High data rates of more than 100 Mbit/s as well as low delays and delay restrictions according to isochronous data traffic are required. To provide the performance requirements of those applications there are two approaches:

- Existing protocols, e.g., OSI protocols or TCP/IP, are implemented efficiently by the use of special implementation techniques, hardware, or multiprocessor architectures.
- So called light weight protocols such as the Xpress Transfer Protocol (XTP) and the Versatile Message Transaction Protocol (VMTP) or special purpose protocols have been developed. Several mechanisms of conventional protocols are often considered as unsuitable for high speed networks. Moreover, conventional protocols often restrain efficient protocol implementations in hardware or on multiprocessor architectures.

This paper describes an implementation of XTP on a multiprocessor architecture and a simulation of VMTP on such an architecture. The protocols are evaluated under the aspects on usability for efficient implementation on multiprocessor architectures based on the performance results obtained from measurements and simulations.

2. An Overview of the Analyzed Protocols

In this section we will focus on the distinctive features of the analyzed protocols. The special subjects will be the main characteristics of the high-speed protocols VMTP and XTP. The short overview of ISO Transport Protocol Class 4 (TP4) will be recommmended as a basis for comparison of the analyzed functionalities of VMTP and XTP.

2.1 VMTP

The realization of the protocol VMTP has been driven by the *client-server-model*. In this model the client addresses a *user-request message* to a server. This server will return a *user-response message*. The exchange of these two specific messages is called *transaction*.

Each communication system running the VMTP includes a *manager*, which is dealing with the transaction- and state-information management. The manager is responsible for the reliability of the data transfer. For that reason, *manager-request messages* and *manager-response messages* may be exchanged between VMTP managers using the VMTP mechanisms recursively.

The internal structure of a single VMTP node shows figure 1. Transmitting data via VMTP the sender acts as a Client and the receiver as a Server. The Manager may be regarded as a pseudo-user above VMTP but below a real layer 5 user.

The two used *data formats* for the request- and response-message differ in small parts only. The packet header, consisting of a 24 Byte VMTP-Control part as well as of a 40 Byte Message Control Block, and a trailer including the checksum only, are of fixed length. The maximum amount of transmitted data at a time will be restricted up to 4 Mbyte and will be divided into 256 *packet groups* of 16 kByte data each, where each group will contain 32 *packets* of 512 Byte.

The combination of continuous running packet groups of the same type (requests or responses) into a single transmission unit is called *run*, which simplifies the acknowledgements.

The mechanism called *streaming* allows starting of more than one transaction at a time, which is the initiating of a new transaction before the old one has been acknowledged. It is important that all transactions in this case have to be included in a single run. Otherwise all old transcations will be aborted.

The VMTP does not include an explicit connection establishment and release phase. The transmission of a response-message establishes a connection implicitly. Returning the last response-message releases this connection but does not lead to the throw away of the connection state information.

For that reason, two packets including user data are needed for the connection management only.
The possibility to maintain that kind of connection forces the existence of a data structure called *client state record* (CSR). Sender and receiver store the connection state information for a dedicated period in the CSR. If a CSR does not exist in a communication node, the VMTP manager will solve the lack of connection state information. The so called *probing* mechanism will be initiated by the requested manager using the VMTP transmission protocol recursively. The CSR will be kept in memory of a communication node for a certain time to minimize the number of probings. Following transactions between the same communication users will use the old CSRs and do not have to set-up a new connection.
The addressing mechanism uses *entity identifiers.* The network layer independent addressing allows the migration of applications between different locations. *Group identifiers* provide the possibility of multicasting. Additionally, each packet group owns a *transaction identifier.* Therefore, a transaction will be identified uniquely by the pair <entity id, transaction id>. The 4 Byte length of the transaction identifier is important to grant duplicate recognition of late packets, because of the missing connection set-up and release phase.

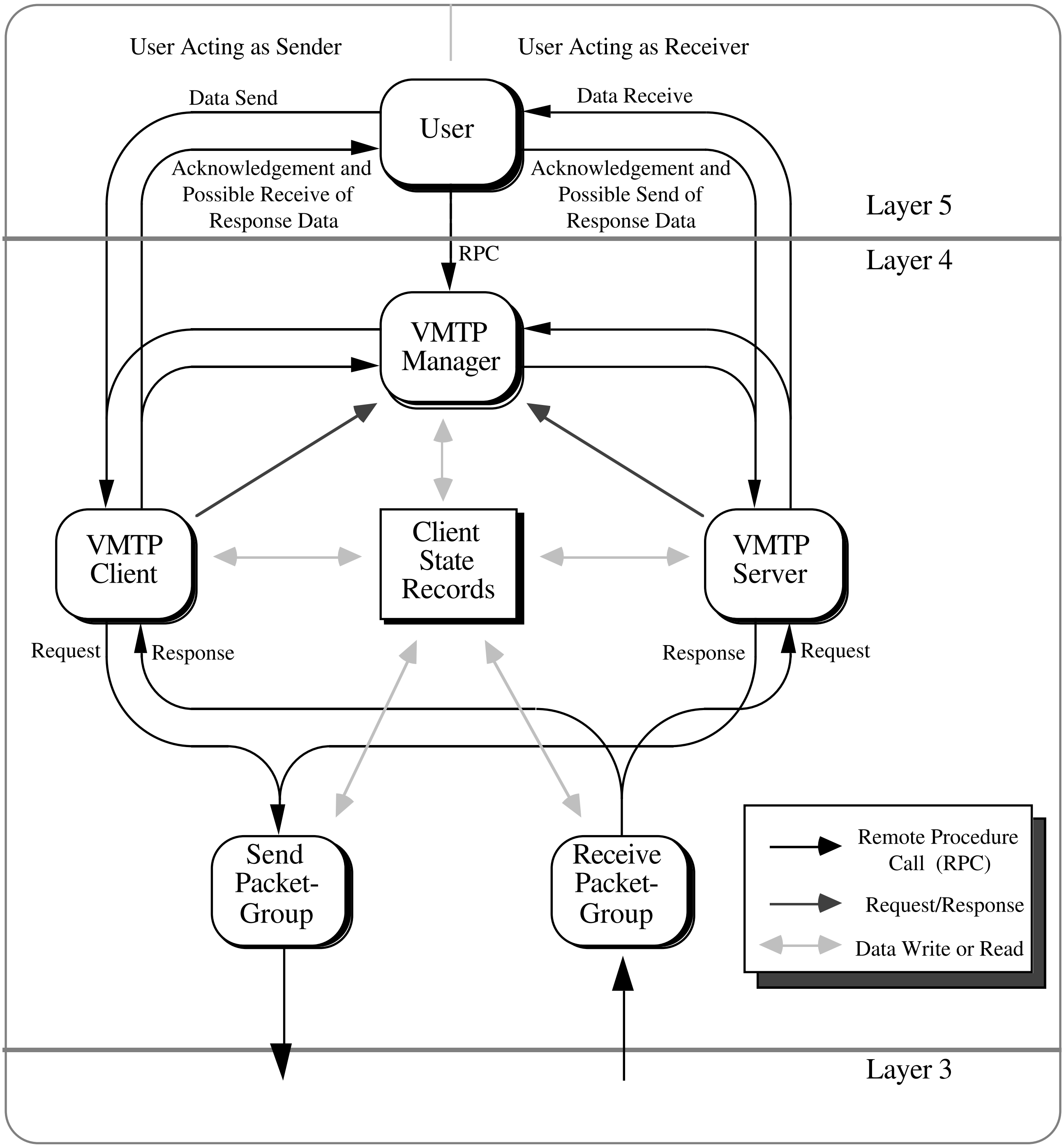

Figure 1 Internal Structure of a VMTP Communication Node

The *acknowledgement* of request-messages will be done by the returned response-messages implicitly. Any following request-message will acknowledge the preceding response-message. The arrival of messages on time will be controlled using different server and client timers. The *checksum* calculation is similar to the one used in the Internet Protocol. The *rate-based flow control* will employ the interpacket gap to control the flow of data.

The VMTP specification will be found in a partial incomplete and inconsistent version 0.7 in [2].

The design of a Network Adapter Board integrating the VMTP protocol has been done by [3]. This approach using hardware support shall lead to faster processing of incoming and outgoing packets than the software solution will be able to.

2.2 XTP

XTP [1] is a transfer protocol designed for the use in high speed networks from 100 Mbit/s to 1 Gbit/s, which has functions of the transport and the network layer according to the OSI reference model. The design of XTP was influenced by the following ideas:

- The receiver part is simplified, because receiving is mostly the bottleneck in protocol processing. Several functions moved from the receiver part to the sender part.
- XTP offers a connection oriented service, but it is also appropriate for connectionless data transfer. Datagrams are supported by so called short-lived connections. Single bits in the packet header can cause connection establishment and termination.
- XTP utilizes two packet formats: one for control packets and one for information packets. Fixed headers and trailers support efficient processing of both packet formats. The protocol is intended for implementation in VLSI chips. Efficient hardware implementation shall be achieved by parallelism inside the protocol. A protocol processing hardware to support XTP is currently developed in the Protocol Engine Project [4].
- XTP was not designed for special kinds of networks or for special application types. To minimize unnecessary functions XTP offers different modes, options, and formats for several requirements.

Several mechanisms are realized to achieve the generality mentioned above. Connections can be established and closed implicitely and explicitely. When a connection between end systems has to be established through one or more routers, these routers have to do the routing decision. Each connection through the router is identified by a value called *route*. A route is generated with a FIRST packet. Subsequent packets of this connection contain the same route values.

XTP uses a byte oriented flow control mechanism such as in TCP. The receiver's control parameters are requested by the sender. They contain information about its buffer state and are delivered within control packets to the sender. Besides the window-based flow control mechanism there is also a rate control mechanism that restricts the size and the time spacing of bursts of data from the sender.

Similar to TCP and OSI TP4 checksum calculation is used to detect bit errors and to initiate retransmission. Checksum is placed in the trailer, so it can be calculated while a packet is being sent. Corrupted packets are discarded by the receiver without any other actions. Only when a packet with a sequence number out of order arrives, reactions are triggered at the receiver side. For error recovery the protocol allows go-back-n or selective retransmission.

XTP uses a few timers only. The most important ones are the connection, the path, the wait, and the rate control timer. The two former ones expire when connections or routes have been inactive for a certain time. The wait timer is used to monitor the receiver's reaction after a request for a control packet by the sender. When the wait timer expires, the sender attempts re-synchronization. The rate control timer is explained below.

As already mentioned above, XTP is designed to be implemented in VLSI. The protocol consists of a couple of finite state machines (FSMs) to support efficient execution [5].

Some FSMs are multiplied for every context and some for every route. The FSMs cooperate on common data, e.g. context data, and by signalling events to other FSMs. FSM state transitions are triggered by those events and are dependent on the contents of current context data. There are different types of events: network, timer, internal, and external events. Network events occur when packets arrive from the network. External events are triggered by the layer above XTP (XTP user). Internal events are created by the XTP FSMs and are signalled to other XTP FSMs. Timer events are triggered by expired timers.

The central FSM is the ***context manager state machine*** (CMSM, cf. figure 2) that wakes up other FSMs of a certain context. CMSM becomes active when a new connection has to be created. Then, CMSM creates a new context, signals events to the other FSMs to activate them, and monitors the corresponding connection with the connection timer. After a connection has been closed, the corresponding context is released. The CMSM and the other FSMs return to inactive state.

The ***output state machine*** (OSM) has to decide whether, which, and how much data has to be sent to the network. The OSM contains the sender's flow control activities, checks the flow control parameters and decides whether control packets must be requested from the receiver.

The ***sync state machine*** (SSM) supports the synchronizing handshake that is used to synchronize control communications between two hosts. A sender increments a packet counter whenever a data packet has been transmitted. The current counter value is sent in the *sync* field of control packets. The receiver returns this value in the *echo* field of its control fields. The sender matches sync and echo fields and can decide to retransmit packets sent prior to the control packet with this sync value. Sync/echo packets are also used to estimate current round trip time.

Rate Control is supported by the ***rate control state machine*** (RACSM). The RACSM is periodically interrupted by the rate control timer to reset the *credit* value. Credit indicates how many bytes the sender can transmit to the according receiver in a certain period of time. OSM decrements the credit value whenever data has been sent. By this mechanism the amount of data sent during a certain time period is restricted.

The ***control send state machine*** (CSSM) creates and formats control packets. It is triggered by various other FSMs, when those demand to send control packets. CSSM competes with OSM for network access.

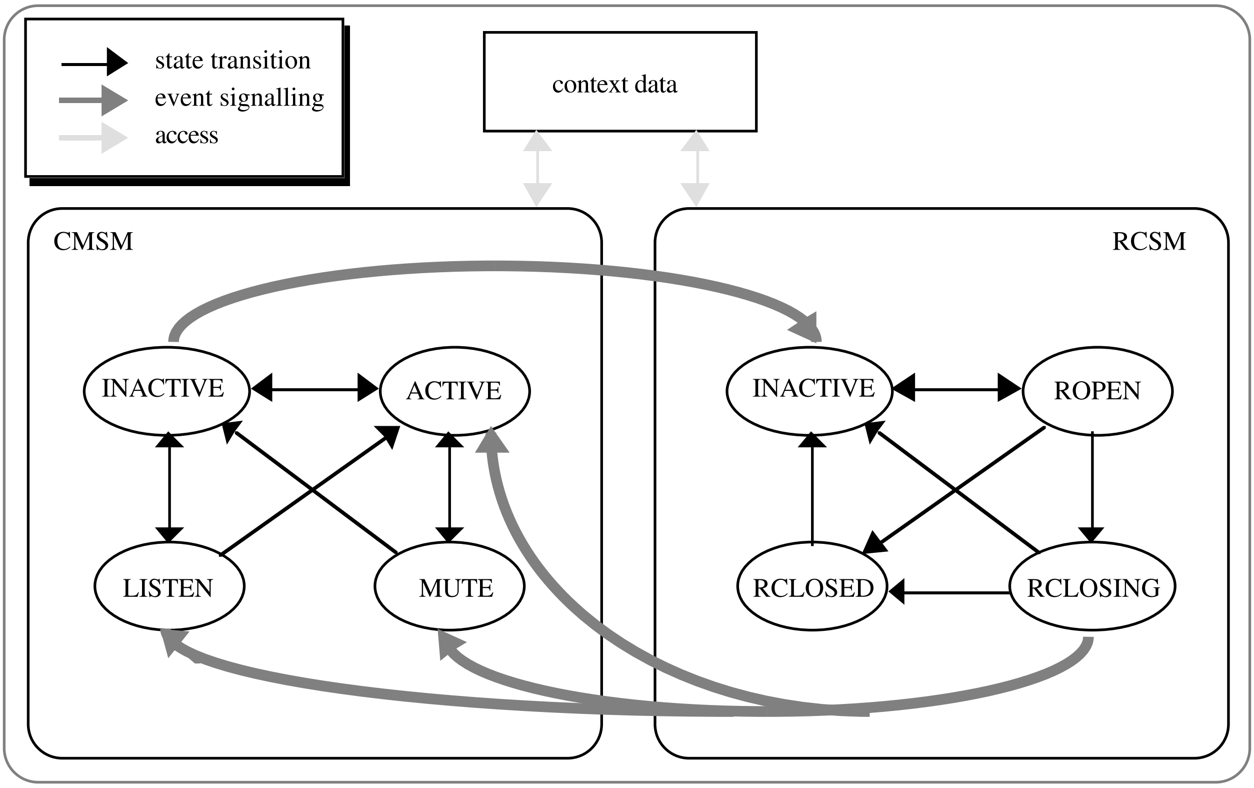

Figure 2   State Transition Diagram of Context Manager State Machine and Read Close State Machine

The data and control packets on the receiving data path are processed by the ***input state machine*** (ISM) and the ***reader state machine*** (RSM). ISM checks addresses, the sequence of data packets, updates context data, causes sending of control packets to the associated sender, if necessary, and delivers data packets to the user via RSM.
The ***write*** (WCSM) and ***read close state machine*** (RCSM) perform closing of connections. It is checked whether data sent earlier arrived correctly at the associated receiver before closing the writer (sender). A receiver has to acknowledge received data before closing.
Two FSMs exist for routing activities. The ***host route manager*** (HRM) monitors the routes being active supported by a path timer. The router state machine (ROSM) performs routing and route exchange.

2.3 OSI Protocols

In local area networks OSI based protocol stacks are often a combination of the OSI transport protocol class 4 (TP4) and the connectionless network protocol (CLNP). TP4 [6] provides a reliable connection oriented transport service by the use of checksum calculation, timer-based retransmission, window-based flow control, duplicate recognition, sequencing, expedited data transfer, segmentation/reassembly, concatenation/separation, and (de-) multiplexing. Thirteen quality-of-service parameters concerning the speed and the reliablility of a dedicated connection can be negotiated. CLNP [7] supports the establishment of a global network out of a number of heterogenous networks. The main tasks are the routing and relaying functions. Furthermore, checksum calculation, segmentation/reassembly, and error reporting are defined.

3. Protocol Simulations and Implementations

This section deals with performance aspects of the protocols. The VMTP behaviour on a multiprocessor architecture has been simulated. XTP has been implemented on a multiprocessor architecture based on Transputer networks. OSI protocols have been implemented on the same environment. Performance evaluations and improvements of the implementations are still going on. Actual performance results are presented in the following.

3.1 VMTP Simulation Model

The VMTP simulation model [8] provides a slightly reduced functionality of the VMTP specification, because of the reduction of simulation time. But still the important mechanisms are supported; these are runs, probing, rate-based flow control using the interpacket gap, the optional checksum calculation, datagram service provided via the DGM-flag, manager based selective retransmissions, priorities, and different request codes. The simulation model allows the treatment of lost, altered, and garbled packets. The implementation of the address generating mechanism is mandatory, because the VMTP-manager and a VMTP-user are two different users of the VMTP-protocol in every communication node and they have to be addressed uniquely.
Figure 3 shows the process configuration implemented in the simulation model. Each box represents a single process and each arrow the interaction inbetween them. The inter-process communication has been realized using special messages supplied with the attributes message-type, priority, and the message itself. The communication time between processes will be variable like the processing times itself of every single process for simulation reasons. The memory manage-

ment does not distinguish between local and global memory yet. The integration of this distinction has been prepared.
The logical distinction of the simulation model should be seen as follows:

- The processes UserSend, SendRequest, HandleResponse, Fragmentation, ChecksumSend, Transmit, ReceiceTimeout and ClientTimeout belong to the client.
- The processes UserRec, HandleRequest, SendResponse, Collect, ChecksumReceive, and ServerTimeout belong to the server.
- The process Manager will act as a client or a server depending on the protocol state.

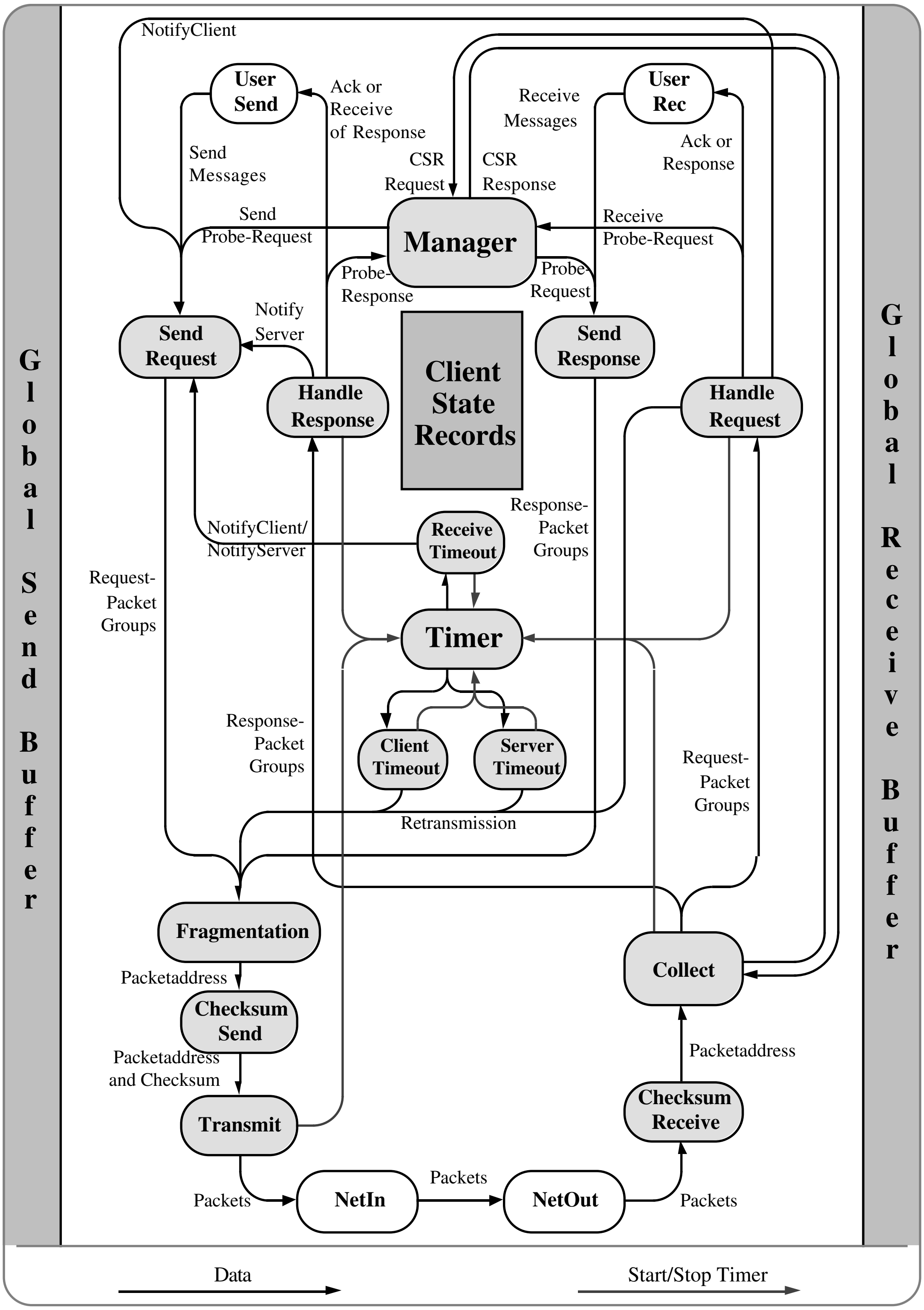

Figure 3 VMTP Processes of the Simulation Model

This model is the result of an analytical research considering functions as basic units, which may be processed in parallel on a real multiprocessor architecture. This advantage will be curtailed by the additional inter-process communication between these parallel units. Therefore, a simulation variable representing the inter-process speed has been introduced. Reducing a priori copying overhead of data the concept of a global send- and receive buffer will be used. Only pointer to the unprocessed user data may be exchanged additional to the protocol header data, which are manipulated in different VMTP functions.

A short description of each separated function will be given. The UserSend and UserReceive concludes every detail of the upper layers. SendRequest administrates runs and takes messages apart into packet groups. It initializes the CSR and composes the packet header. Fragmentation divides packet groups into single packets. Before issueing the packets via Transmit to lower layers the checksum will be calculated in ChecksumSend. Different timer control the send and receive activities. NetIn and NetOut simulate all characteristics of an underlying network like maximum transmission capacity and error rates. On the receiver side ChecksumReceive proves the checksum and hands a correct packet to Collect, where a redivision into packet groups will take place.

Additionally, a manager request for an appropriate CSR will be initiated. The whole packet group will be send to HandleRequest if and only if the Manager responded with CSR information and all packets of a single packet group arrived. HandleRequest itself recombines packet groups to the originally send message. SendResponse receives acknowledgements, which may be a single ack or a huge amount of data, or responses from the user. For that reason, large data in SendResponse may be processed like in SendRequest. Finally, the Manager serves as an CSR administration authority in the communication node.

3.2 VMTP Simulation Results

Simulation results will be presented to prove the performance of the high-speed protocol VMTP. These results will be regarded for three different situations, whereby two communication nodes are connected. In the first case a single message of a fixed length has been send. In the second and third case a continuous output of fixed length messages has been transmitted. Time Receive depicts the time between sending and receiving a data unit. Time Acknowledge contains the additional time for returning the acknowledgement back to the sender. By analogy, Rate Receive defines the throughput of transmitted data only, and Rate Acknowledge includes the acknowledgement data into the throughput calculation.

First, figure 4a depicts the influence of the connection management via the probing mechanism depending on the transmitted message size. The checksum calculation on sender and receiver side is integrated, the network has a capacity of 100 Mbit/s, and the packet size is 4 kByte. If messages less than 26 kByte size are regarded, which will be segmented into 4 kByte packets, the time to create a new CSR at the server's side and the time transmitting a single message will be larger than the time transmitting the data only, because of the additional round trip time for the probe-request and probe-response.

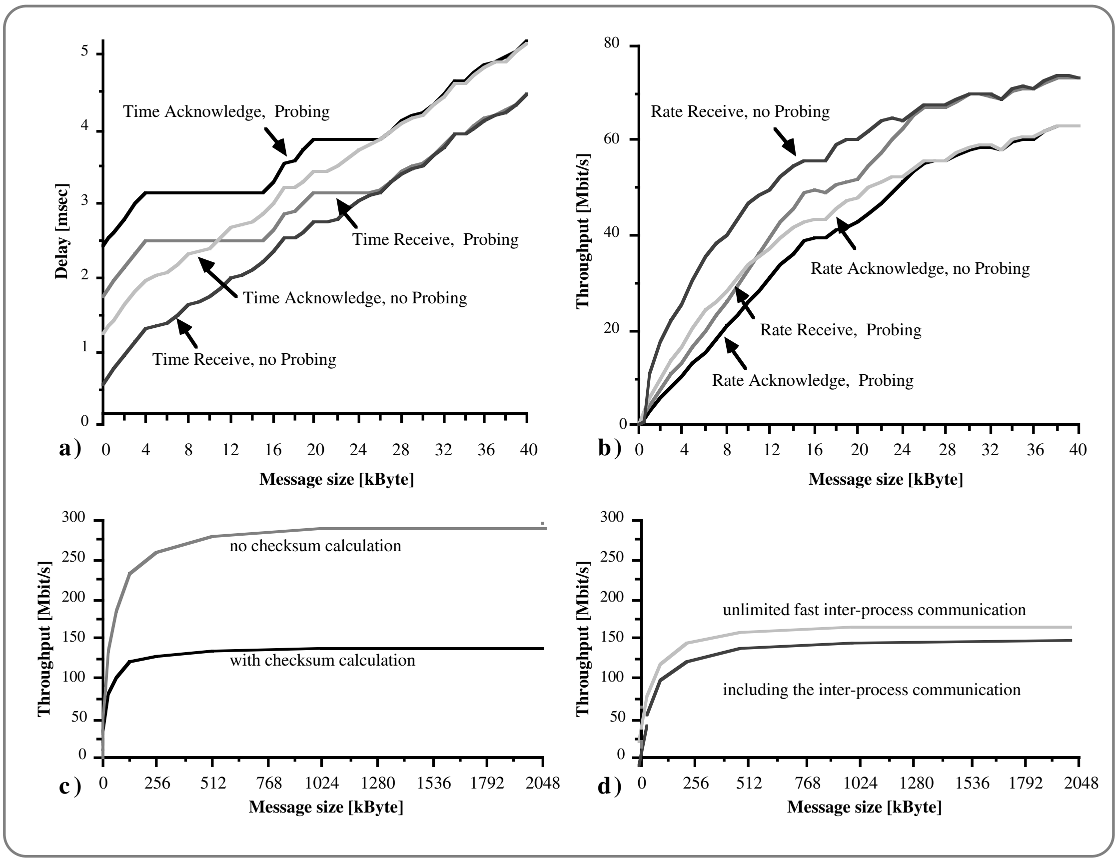

Figure 4 VMTP Simulation Results

For messages larger than this break-even point of 26 kByte the delay waiting for the probe response will be less than the transmission time of the number of 4 kByte packets. Conclusion: for large messages the non-existence of CSRs does not increase delays. The corresponding throughput up to a message size of 40 kByte will be found in figure 4b, where equal values for the probing and non-probing situation exist.
Second, the maximum throughput for two communication systems connected by a 10 Gbit/s link, representing a network that accepts data at almost every time, will be found in figure 4c. The checksum calculation - 20 MByte/s fast - still influences the throughput heavily. For that reason, the throughput drops from about 285 MBit/s to 135 MBit/s for large messages.
Third, the affects of the speed of the inter-process communication will be found in figure 4d. The more theoretical maximum result of unlimited fast inter-process communication leads towards a throughput just 30 MBit/s higher than the throughput with the stated inter-process communication speed of 15 Mbit/s. These results include the checksum calculation also.

3.3 XTP-Implementation on Transputers

Design Aspects

For the design of the XTP implementation different aspects have been considered. First, it should be possible to insert the XTP implementation into an implementation of a protocol stack on a multiprocessor architecture according to the concepts also presented in [9]. The protocol layers operate in a pipelined principle: each packet moves through the protocol layers step by step. Such a layer pipeline performs overlapped computation of sublayers to exploit parallelism. Additionally to hierarchical structuring of sublayers the protocol entities of those layers can be divided internally into a send and a receive part resulting in a send and a receive pipeline for each network node. Usually, the send and the receive pipeline are not completely independent of each other. For example, the sender and the receiver part have to exchange control information such as connection state data in a connection oriented protocol or data units in a gateway system.
To avoid movements of data packets between different layers in a protocol stack, a global data memory must be provided, to which all layers must have access. Inside this data memory all data packets remain at a fixed place while being processed by the layers of the protocol stack, i.e. the data packets are not moved when passing from one layer to another. The different layers only add (send side) or remove (receive side) the packet header or trailer to or from the data unit. The global memory is divided into two parts, one part for data of the sender functions and one part for data of the receiving ones of the protocol stack. For each global memory part manager processes are implemented to manage the access to the global memory. Copying of data is reduced significantly. Only references to data packets stored in the global memory are delivered between adjacent layers. One disadvantage may be the overhead caused by the communication between the protocol processes and the memory management processes. In gateway systems data may have to be copied, if there is always a memory part for each network.
Another design issue is how to implement the XTP FSMs. It is impossible to create a process for every FSM because for each connection or route and for each FSM type there exists always one FSM. The number of FSMs would become very large. In consequence, performance would decrease by process switching. Also creation and release of processes are very time consuming.
The presented XTP implementation creates a separate process for every FSM type. This process is responsible for every created context or route. FSM state transitions are triggered by events and predicates. These predicates are mainly context data, which are common for several FSMs. Event signalling between processes is done by passing messages across channels. For two processes, i.e. for two FSM types, there exists exactly one channel, if the two processes are signalling events between each other. Various event types can be signalled across this channel. A message identifier indicates the event type and the corresponding context.
The different protocol processes operate on common context data. To allow all processes to run on different Transputers, the context data is distributed as well as possible to those processes that mainly operate on it. Context data needed by several processes have to be exchanged whenever any context values change.
The primary goals of this XTP implementation are efficiency and flexibility. For that reason, mainly small FSMs that exchange a lot of events with other FSMs and that share a considerable amount of common data with other FSMs are combined to a product FSM. Too fine process granularity causes high communication overhead because of the communication concepts realized on Transputers. The resulting FSMs are implemented by a single process that can run on a single processor or that can run concurrently with other processes on a processor.

XTP Processes

The process net of the implementation [10] is depicted in figure 5. The implemented XTP FSMs are mapped to seven processes: user interface (UI), input (IP), output (OP), control packet (CT), context manager process (CM) and network interface processes for send (NIS) and receive (NIR) part.
The ***user interface process*** (UI) is the interface between send and receive part of XTP and the users. Only UI represents the XTP entity to the users. Different users can use UI at the same time by sending a source identifier with each message. RSM, RCSM and WCSM are implemented inside UI. UI has also the task to deliver data to the user and to control closing of connections.
The sender part of XTP consists of the processes OP, RT, CT and NIS. ***Output process*** (OP) comprises functions of OSM and RACSM. OP has to send data packets to the network and to perform functions such as flow control, rate control, and retransmission of lost or corrupted packets. OP uses the ***routing table process*** (RT) to determine the route for a connection to another instance.
The ***control packet process*** (CT) has to react on requests of other processes that trigger sending of control packets (cf. CSSM). The control packets are necessary for acknowledgement or synchronization procedures. Another task is to perform synchronizing handshakes, i.e. CT performs SSM.
The ***network interface send process*** (NIS) is the interface between the XTP send part and the lower layers. NIS supports OP to perform OSM. NIS controls access to the underlying network and delivers control or data packets to the network. Another task is to complete the packet trailer and to send it to the memory manager. Headers are generated by

CT or OP dependent on sending a control packet or a data packet. Keeping context data necessary for header or trailer generation is distributed among the three processes CT, OP, and NIS.

Similar to NIS, the ***network interface receive process*** (NIR) is the interface between the XTP receive part and the network. NIR uses the ***hash table process*** (HT) to get the local context of an incoming XTP header. Then, NIR delivers the packet to the ***input process*** (IP). IP and NIR perform ISM. IP distributes new context values to those processes, where context data is stored. Morever, IP triggers reactions as control packet requests or connection releases dependent on the header and the trailer information.

The central process of the XTP protocol implementation is the ***context manager process*** (CM) that is executing CMSM. CM triggers initializing context data and computes keys for the created connections. After connection setup CM monitors the connections by means of the connection timer. Timers are implemented by special ***timer processes.*** After closing a connection CM triggers the release of the context data.

The ***forward process*** (FWD) performs routing and relaying functions. It has to forward packets addressed to other destination addresses and to manage the active routes.

There are other processes in the test environment besides the XTP protocol processes. These processes can be classified as test processes and memory manager processes. The network simulation process substitutes the underlying network. This process transfers XTP packets between XTP instances. Test user processes are implemented for test purposes.

The tasks of ***memory manager processes*** for ***send part*** (MMS) and ***receive part*** (MMR) have been already described above. Cooperation between memory manager processes and protocol processes is organized in a client/server principle. Protocol processes (clients) send requests to the memory manager processes (servers) to read or write data packet information and to allocate or release buffer space. Memory manager processes perform the desired actions and return the results to the clients. For example, a protocol process must send a command to MMR, if the process wants to read the received XTP header. MMR returns the header data.

The presented XTP implementation can easily be implemented on various Transputers by mapping processes to processors and interprocess communication connections to software or hardware channels (links) dependent on two processes being mapped to the same Transputer.

If there are more than four interprocessor channels required for one processor, so called multiplex-processes are inserted. The multiplex-processes multiplex several logical interprocess channels to a reduced number of interprocessor channels. Additional overhead caused by the multiplex-processes is mostly very low because interprocessor communications can run in parallel to other computations.

The processes need not to be interconnected directly, if they have to communicate with each other. There is a mechanism implemented in every process to route messages not belonging to the process to the correct destination process.

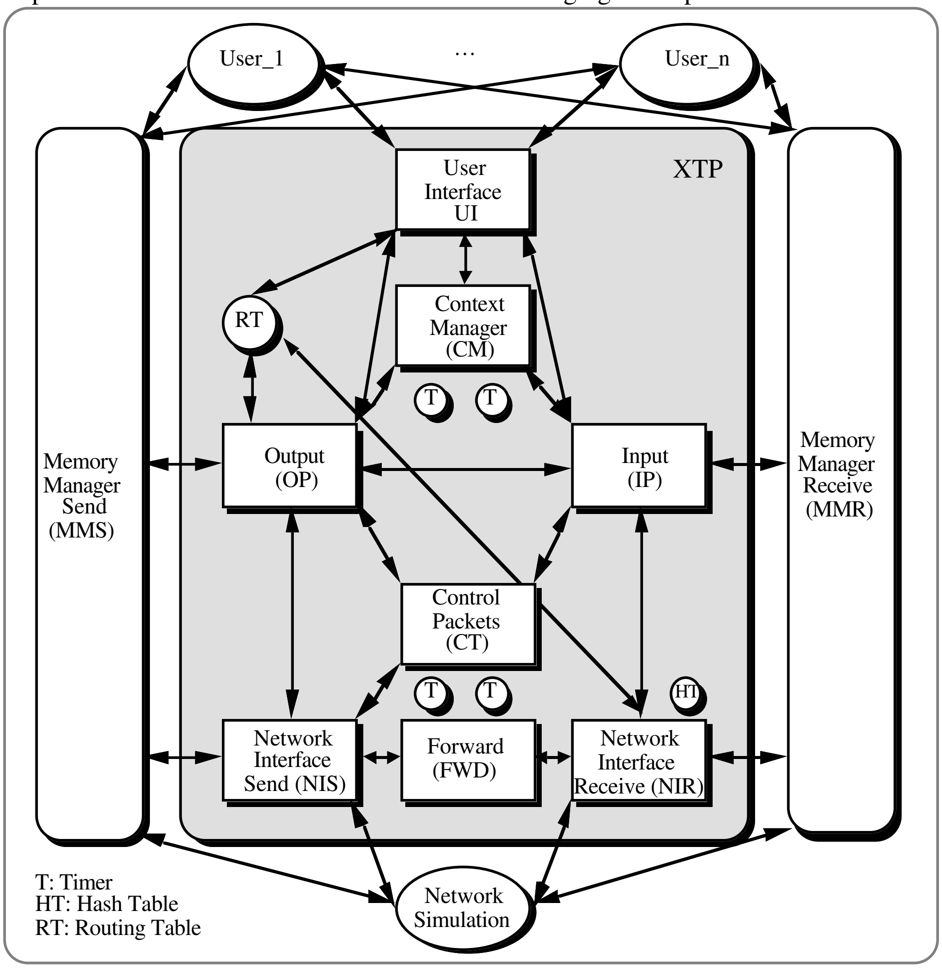

Figure 5 Process Net of the Parallel XTP Implementation

Routing tables are mapping incoming messages to an output channel by means of interpreting the destination process identifier. This mechanism is only used if necessary because it causes additional overhead.
Every process also starts a subprocess, the so called communication-process, that is doing interprocess communication across channels.
A communication-process receives incoming messages from another main process and stores the messages in a queue. The communication-process cooperates with the according main process by this queue and wakes up the main process whenever a message arrives. Then, the main process performs computations dependent on the incoming message. If interprocessor communication is necessary, the main process can do the computations while the communication-process performs interprocessor communications. Additonally, these communication-processes provide asynchronous interprocess communications, deadlocks are avoided.
The design of the multiprocessor implementation is supported by a configurator program. The implementor has to describe the configuration in a description file only. The file must contain the mapping of processes to processors and the connections of the processes' input and output channels. Generation of boot programs to initialize the Transputer network and generation of routing tables mentioned above is based on the implementor's description and is done automatically by the configurator.

3.4 XTP Performance Results

Figure 6 shows first performance results of the current implementation [11]. The results show how much time the processes spend for sending and receiving a packet. After establishing a connection nine packets are sent. Processing times for UI, CM, and RT are independent of the number of packets.
The network interface processes, NIR and NIS, are the bottlenecks with about 750 µs (NIR) and 840 µs (NIS) processing time per packet including communications. These processing times reduce the maximum achievable performance on a multiprocessor configuration with one processor for a single process to 1330 and 1190 packets/second. Configurations of nine T414 Transputers achieve about 800 packets/s. T800 Transputer configurations with no-wait-state-memories are expected to run at least 30 % faster.
Summation of process times without communications comes up to 1900 µs on receiving and to 1350 µs on sending. Parallel execution will have the most advantages on receiving. The communication overhead is still high: it comes up to 25 %. Efforts to reduce communication overhead, e.g., by advanced communication links and mechanisms, will be necessary. Several Transputer network configurations will be examined and evaluated.
Especially, parallel processing of control packets and information packets is very efficient. Generation and parsing of control packets can be done nearly independent of information packet processing. Identifying more independent data streams seem to be a promising approach for protocol design to support parallelism more efficiently.

3.5 Implementations of OSI-TP4 and CLNP

In [9] an implementation of the OSI connectionless network protocol (CLNP) is described. It is shown that the throughput of CLNP implementation on eight processors is nearly four times higher than the throughput of the implementation on one processor. With this implementation a throughput of more than 2000 packets/s is achieved. The transport protocol class 4 (TP4) is implemented on a Transputer network also. The achieved performance (about 400 packets/s) was much lower, because the protocol is much more complex than CLNP. But the speed-up of a multiprocessor implementation in comparison with the monoprocessor implementation was in the range of the speed-up achieved with CLNP implementations. Similar to XTP this implementation will further be optimized. Implementations of the OSI protocols showed that mainly pipelined parallelism leads to performance gain. Spatial parallelism is mainly achievable by separating protocol processing in send and receive parts.

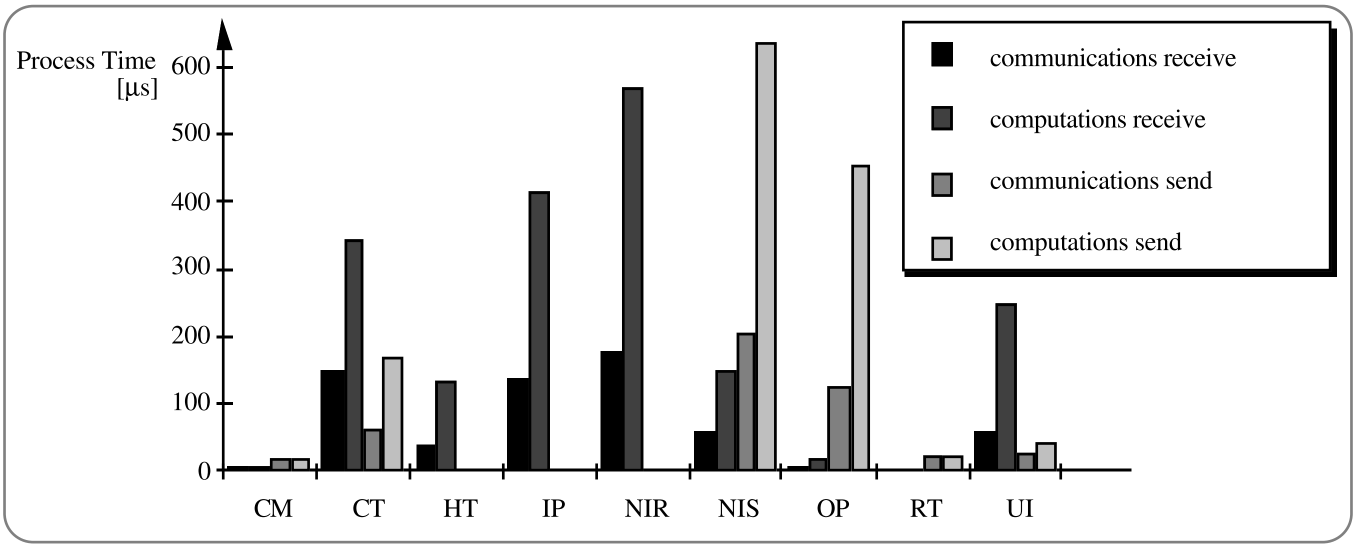

Figure 6 Performance Results of the XTP Implementation

4. Comparison

By the use of parallelism considerable performance gains can be achieved. One reason for differences in performance values of high speed protocols and OSI protocols is the use of different implementation techniques. Another reason is that XTP is suitable for parallelism, e.g., by XTP's concept of parallel finite state machines. A parallel implementation of XTP is easier to obtain than a parallel implementation of OSI protocols, because a parallel decomposition of XTP is given by the parallel FSMs. Moreover, in XTP information packet and control packet processing is separated. Furthermore, XTP is designed for implementation in VLSI. XTP headers and trailers have fixed size to support easy access to the data fields. The main difference, however, is the protocol architecture like support of parallelism as mentioned above. In addition, XTP combines functionality of layer 3 and 4. There are also approaches to integrate LLC type 1 functions into XTP. Integration of several layers reduces layering overhead.

Compared to OSI protocols high speed protocols such as XTP have similar functionalities. They differ in a few adapted mechanisms only. Some mechanisms are added to XTP in comparison with OSI protocols: The delay of connection establishment can be reduced by implicit connection set-up. Datagrams and connections are handled uniformly. Rate control can support the flow control function. Several retransmission methods can be selected. Furthermore, new protocols have functions such as multicast that are required by new applications. But the different protocol mechanisms and methods often have poor effect on performance increase only. The bottleneck is mostly protocol processing itself.

The suitable parallelism for the VMTP protocol should be seen first between the receive and send processes. Second, the duplicating of specific slow functions, e.g., the ChecksumSend or ChecksumReceive functions, will lead to a better performance of the protocol. The impacts on the throughput considering different situations depends on the transmitted message size. The implicit connection management allows for request-response traffic a higher throughput if the CSR already exists. In case of bursty traffic there is no difference between existing or non-existing CSRs - the probing mechanism to set-up the connection is done in parallel to the data transfer. To achieve low delays the CSRs have to be stored for a long time. This requires some memory at every communication node. The facility to process runs reduces the protocol processing overhead due to fewer acknowledgements. The common header format of request- and response-messages forces a more simple and faster header analysis.

Comparing the VMTP to the OSI-Basic Reference Model the functionality of VMTP will be similar to the transport layer. In detail, the (de-)multiplexing function is missing in VMTP as well as the split and recombine function. Other functions are used with just different mechanisms.

5. Conclusions

In general, the protocol XTP is an all purpose protocol. That is, no special optimization for a specific traffic type has been done and therefore, no functional reduction has been dealt with. However, VMTP has been designed having the request and response traffic in mind. But it is not limited to this traffic type. In comparison with OSI protocols they offer similar services and functionality. Main difference to OSI is the protocol architecture supporting efficient implementation on appropriate processing elements.

Especially the VMTP and the XTP are more suitable for parallel processing, but the offered functionality is not restricted. The influence of the protocol processing elements is more and more important to achieve high performance. Protocols must be appropriate for fast processing elements, which can be realized in VLSI or by multiprocessor architectures.

6. References

[1] Protocol Engines Inc.: "XTP Protocol Definition", Revision 3.5, 10 September 1990.

[2] D.R.Cheriton; *VMTP: Versatile Message Transaction Protocol - Protocol Specification Version 0.7*; Computer Science Department, Stanford University, 1988.

[3] H.Kanakia, D.R.Cheriton; *The VMP Network Adapter Board (NAB): High Performance Communication for Multiprocessors*; ACM Sigmetrics, October 1988.

[4] G.Chesson: "The Protocol Engine Chipset", Transfer Newsletter, Vol. 4, No. 3, May/June 1991.

[5] Protocol Engines Inc.: "XTP Protocol Definition", Revision 3.4, 17 July 1989.

[6] International Organization for Standardization ISO: Information Processing Systems, OSI - *Connection Oriented Transport Protocol Specification* ISO IS 8072, 1986.

[7] International Organization for Standardization ISO: Information Processing Systems, OSI - *Protocol for Providing the Connectionless-Mode Network Service,* ISO IS 8473, 1988.

[8] J.Weigl; *Ein VMTP Simulationsmodell mit SIMSCRIPT II.5*; Diplomarbeit, Universität Karlsruhe, Institut für Telematik, March 1991.

[9] M.Zitterbart: "High Speed Transport Components", IEEE Network, January 1991.

[10] A.Schörk: "Parallele Realisierung von XTP auf Transputern", Diplomarbeit, Universität Karlsruhe, Institut für Telematik, October 1990.

[11] Th.Weber: "Untersuchung und Adaption des XTP-Protokolls auf Transputer-Netzwerken", Diplomarbeit, Universität Karlsruhe, Institut für Telematik, July 1991.