PATROCLOS: A Flexible and High-Performance Transport Subsystem

Torsten Braun

Institute of Telematics, University of Karlsruhe, Zirkel 2, 76128 Karlsruhe, Germany
Phone: +49 721 608-3982, Fax: +49 721 388097
E-mail: braun@telematik.informatik.uni-karlsruhe.de

ABSTRACT

Increasing application demands together with the modern network technology towards gigabit speeds require new adequate transport systems. The transport subsystem PATROCLOS (parallel transport subsystem for cell based high speed networks) is based on a modular protocol architecture with a high degree of inherent parallelism. The primary goal of PATROCLOS was to allow efficient implementations of transport oriented protocols on multiprocessor architectures combined with specialized hardware for very time critical functions. Furthermore, the protocol architecture should offer an enhanced and flexible transport service to support a wide variety of applications providing a large set of different configurable protocol functions. Other important aspects of the transport subsystem are its efficient operation on top of modern cell-based high-speed networks such as ATM.
This paper describes the PATROCLOS protocol architecture with special emphasis on the integrated protocol functions and mechanisms. The protocol architecture allows an easy integration of several kinds of transport service interfaces. Two different types of service interfaces (OSI95 [1], F-CSS interface [2]) are discussed. Moreover, the paper shows the advantages of the modular approach for parallel protocol processing by performance results of a multiprocessor implementation. The performance results show significant better throughput and speed-up values than other parallel protocol implementations.

Keyword Codes: B.4.1; C.1.2; C.2.2
Keyword: Data Communication Devices; Multiprocessors; Network Protocols

1. INTRODUCTION

PATROCLOS provides the functionalities of the layers between the media access layer interface of cell-based networks and the transport service interface in a single protocol component (functionality of OSI layers 2b-4). To support the increasing variety of applications, it is necessary that transport systems provide new service types, e.g. according to the OSI95 service definition [1], together with an enhanced set of quality-of-service parameters as proposed in [2]. Dependent on the type of service or the quality-of-service parameters, different protocol functions have to be selected. E.g., the type of retransmission (selective repeat, go-back-N, or even no retransmission) may have significant impact on the achievable delay. For that reason, future transport subsystems should provide a lot of different protocol functions, which have to be configured dependent on the application requirements and network characteristics. For example, the selection of go-back-N retransmission strategy may cause high delays in networks

with a large bandwidth-delay-product, while selective repeat strategy or forward error correction can significantly reduce the delays [3]. Therefore, configuration facilities will play an important role in future protocols to allow their adaptation to the wide variety of application requirements and network environments [2, 4].
PATROCLOS is based on a modular design concept to simplify the application-driven and network-dependent configuration. The key feature of the protocol is that it consists of a set of extended finite state machines (FSMs), where each FSM performs a dedicated protocol function such as connection management or acknowledgment. The modular approach is also useful to increase the performance of transport subsystems significantly by concurrent processing of the different FSMs. Parallel implementations of standard protocols like OSI TP4/CLNP or TCP/IP suffers from their inherent sequential structure. Therefore, parallel implementations of standard protocols use pipelining [5] or parallelism on a per-packet basis [6]. The transport subsystem PATROCLOS realizes a parallel protocol architecture with fine granularity adequate for parallel implementations and overcomes performance limitations by the integration of protocol and implementation issues. In contrast to other approaches of parallel protocols [7, 8, 9], PATROCLOS is based on a decomposition into building blocks, which are oriented on protocol functions. Therefore, the building blocks are the atomic units for implementation on a parallel implementation architecture and for configuration of the transport subsystem, simultaneously.

2. THE PATROCLOS ARCHITECTURE

The PATROCLOS approach combines a fine granularity with a protocol function oriented decomposition into basic modular building blocks to simplify protocol configuration and parallel implementation, simultaneously. Furthermore, it avoids layering to achieve a high degree of parallelism. Redundant placement of protocol functions such as segmentation and reassembly functions, which occur in layer 3 and 4 of the OSI Reference Model, in several layers can be avoided. Mainly, the characteristics of emerging cell-based networks such as ATM and DQDB influenced the PATROCLOS protocol design [10].

2.1. Parallel Protocol Architecture

The protocol architecture has been developed based on the analysis of a XTP [8] specification, which consists of a set of parallel FSMs, and the performance results of the corresponding multi-processor implementation [11]. PATROCLOS is also based on the concept of separating data transfer and transfer control functions as proposed in the integrated layer processing approach [12].
Data transfer and control functions are separated within the PATROCLOS protocol specification to allow a higher degree of parallelism. The separation of data and control path processing is also performed in the AXON approach [13] PATROCLOS has been specified with the specification language Promela [14], which has several advantages describing a protocol as a set of parallel FSMs. Those parallel FSMs are the building blocks of the protocol architecture. Together, all FSMs form a FSM system. FSMs belonging to the same PATROCLOS entity exchange internal messages for co-operation. Internal messages consist of a locally unique connection identifier and the message body. The message body may contain information such as sequence numbers, buffer state information, etc. (cf. Figure 2). All FSMs are designed as autonomous as possible to minimize interactions with other FSMs of the same entity. The communication among FSMs is uni-directional to allow a high degree of parallelism. If possible, the necessary information is exchanged periodically to reduce the communication overhead.

The modular protocol design supports an easy mapping on multiprocessor architectures, if the FSMs are implemented as processes, which can run on separate processors.
PATROCLOS consists of interface and protocol FSMs (cf. Figure 1, 2). ***Interface FSMs*** are located at the interfaces of the transport subsystem to the application and the network. They support interactions of PATROCLOS with the transport service user or with the underlying network unit and are only involved in local communications inside one entity. In contrast to the protocol FSMs, they do not have to communicate with FSMs of the peer entity. The network interface FSM (send control FSM) controls and schedules transmission of *all* protocol data units (PDUs, PATROCLOS-frames, P-frames) to the underlying network unit. Note that this FSM is missing in Figure 2. The transport service interface FSM maps service primitives to signals for the protocol FSMs, exchanges transport service data units (TSDUs) with the transport service user in both directions and is involved in performing graceful connection termination.

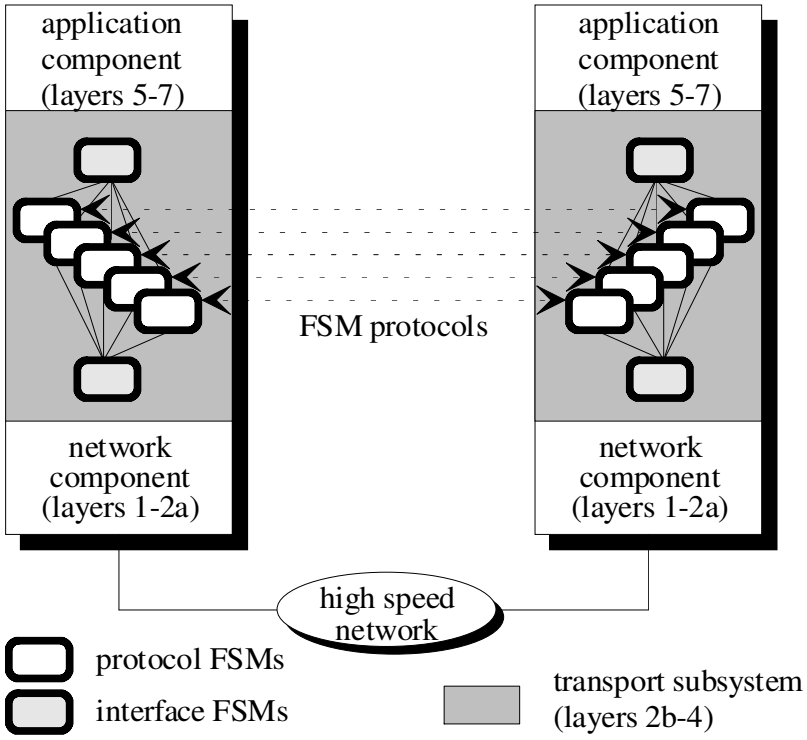

Figure 1: Protocol Architecture

The key feature of ***protocol FSMs*** is that they communicate directly by separate, so-called ***FSM protocols*** with the corresponding FSMs at the peer entity. The protocol FSMs are designed to allow parallelism between send and receive part and to decouple the connection state information exchange from user data exchange. Moreover, since a large amount of data may be traveling between communicating end systems, state exchange (e.g., acknowledgments and connection state information exchange) after dedicated requests and errors (e.g., XTP [8]) or periodic exchanges (e.g., SNR [7] and Delta-t [15]) seem to be more suitable than state exchange for each data packet. All FSM protocols use separate P-frames (external messages), individual error recovery mechanisms, and timers. P-frames containing user data are called P-data-frames or packets in contrast to the P-control-frames, which contain only control information. Each FSM protocol is performed by communicating peer FSMs, using P-data-frames containing only the necessary information for their dedicated protocol function. Multiplexing

of P-frames by different FSMs is avoided to support parallelism. Together, the set of FSM protocols builds the complete PATROCLOS protocol. The decomposition into independent FSMs permits parallel execution and has the benefit of a highly modular system, which can be configured easily depending on the requirements of the applications and the characteristics of the networks as suggested in [2]. There are the following protocol FSMs:

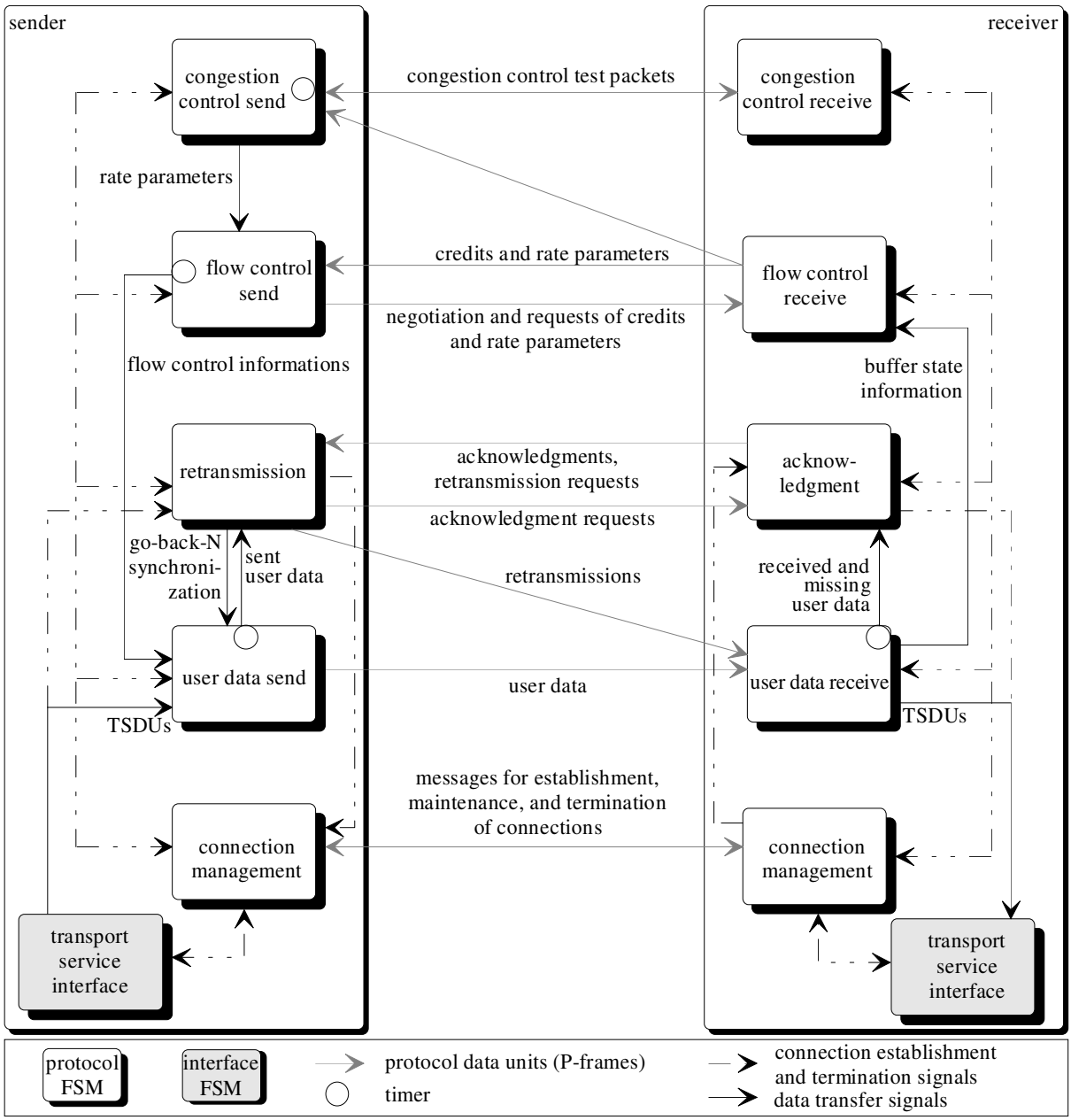

Figure 2: Communication Among FSMs

The ***connection management*** FSM is responsible for establishment, management, and termination of a full duplex connection. Therefore, it has to activate and initialize the other FSMs of the same entity for each new connection and has also to deactivate these FSMs after connection termination. The peer connection management FSMs exchange P-control-frames for connection establishment, initialize the connection context, and forward quality-of-service require-

ments to the other FSMs of the entity. The connection management FSMs also guarantee the unique use of connection identifiers and perform signalling tasks, e.g. in B-ISDN networks.
User data transfer can be divided into a send path and a receive path. User data transfer FSMs send and receive user data. The main tasks of both user data transfer FSMs (user data send and user data receive FSM) are formatting and analyzing P-data-frames. Additional functions are segmentation and reassembly. Therefore, each user data transfer FSM consists of two subordinate FSMs. On sending side, the segmentation FSM segments TSDUs into P-data-frames. The data send FSM formats the P-data-frame headers to describe the user data (e.g., sequence numbers). On receive side, the data receive FSM operates on these informations to detect duplicates or errors and checks the packet lifetime. The reassembly FSM reassembles P-data-frames into TSDUs. Another FSM in intermediate systems performs functions such as ***routeing & relaying***, route management, and monitoring.
Because data transfer mostly performs the critical path in communication protocols, ***retransmission and acknowledgment*** are handled by two special FSMs. The FSM on the send side (retransmission FSM) requests and receives acknowledgments from the peer's FSM on the receive side (acknowledgment FSM). The retransmission FSM analyzes the acknowledgment P-frames, initiates retransmissions dependent on the acknowledgment information and the selected strategy, and releases the buffers of acknowledged data packets. Retransmissions are sent to the peer's user data receive FSM. The retransmission FSM receives the control over sent user data packets from the user data send FSM periodically with an interval lower than one round trip time (rtt). The user data receive FSM signals informations about received data to the acknowledgment FSM periodically. The acknowledgment FSM sends negative selective and positive cumulative acknowledgments to the peer's retransmission FSM. Communication between the FSMs participating in user data transfer is reduced to a minimum. In the case of error-free communication, the FSMs in one entity communicate periodically instead of exchanging messages after each packet. In error cases, the informations are sent directly to avoid unnecessary delays.
Two ***flow control*** FSMs exist in each entity, one FSM for each data flow direction. They negotiate flow control parameters with the remote peer and use these parameters to control the data flow. The flow control receive FSM transmits rate parameters or credits according to the current traffic and available buffer space, which is signaled by the data receive FSM.
The ***congestion control*** FSMs control data flow by the interpretation of feedback signals from the network or the peer entity, e.g., congestion indications, round trip times, and inter-arrival times of test P-frames, which are exchanged between peer FSMs to detect congestions. The FSM may also react upon the feedback signals of intermediate systems.

2.2. Transport Services

2.2.1. Enhanced OSI Transport Service

At first, PATROCLOS provided a transport service similar to the transport service definition developed within the OSI95 project. The OSI95 transport service definition comprises a normal connection-mode, a fast connection-mode, an unacknowledged connectionless-mode, an acknowledged connectionless-mode, and a request/response connectionless-mode transport service. These services are grouped into three classes as shown in Table 1. Additional options and parameters are offered for the connection-mode service. The connection establishment may be performed by a 2-way- or a 3-way-handshake. Furthermore, an implicit connection establishment (fast connect) is provided.

Table 1: Enhanced OSI Transport Service

Service Type	Service Parameters	
Connection-oriented Service	establishment parameter	termination parameter
	2-way-handshake acknowledged 2-way-handshake unacknowledged 3-way-handshake acknowledged 3-way-handshake unacknowledged fast connect (implicit)	graceful normal (abrupt) timer-controlled
Connectionless Service	acknowledgment parameter	
	acknowledged unacknowledged	
Request/Response Service		

For connection termination the transport service user can select a graceful release, which makes sure that all in-transit data are delivered, in contrast to the normal (abrupt) termination, which is used in OSI-TP4. To provide a graceful connection release, the connection management FSM has to ensure that all data, which are delivered to the transport system before initiating connection termination, have been sent and acknowledged. The graceful release requires co-operation with other FSMs such as the retransmission FSM to be sure that all transmitted user data are correctly received and delivered at the receiver.

Another option is the timer-controlled termination. A connection is terminated, if there is no user data exchange for a certain inactivity interval. The data send and the data receive FSM send inactivity signals to the connection management FSM, if there is no user data exchange for the time period t_{send} or $t_{receive}$ respectively. Otherwise, they send activity signals. The transport service user has to specify a time value t_{idle}. If the connection management FSM receives $\lceil \frac{t_{idle}}{t_{send}} \rceil$ inactivity signals successively from the data send FSM or $\lceil \frac{t_{idle}}{t_{receive}} \rceil$ inactivity signals from the data receive FSM, the connection is terminated. The inactivity interval is bounded by the following formula:

$$t_{idle} < \text{inactivity_interval} < \max(\lceil \frac{t_{idle}}{t_{send}} \rceil \cdot t_{send} + t_{send}, \lceil \frac{t_{idle}}{t_{receive}} \rceil \cdot t_{receive} + t_{receive})$$

2.2.2. F-CSS Transport Service

Because of the modular structure of PATROCLOS the required enhancements to offer other transport services are very low. PATROCLOS offers a number of different protocol functions. Especially, mechanisms appropriate for high-speed network environments with a large bandwidth-delay-product have been integrated. Because of the ability of PATROCLOS to be configured, we integrated the PATROCLOS protocol architecture and the F-CSS approach, which provides a framework for communication subsystems to offer a flexible transport service dependent on application and network requirements. F-CSS [2] provides an application interface specifically designed to support service flexibility. It relieves the application from dealing with a large number of available protocol functions by providing a small number of pre-defined service classes. The application interface provides the application with the possibility of formulating its individual requirements in terms of service parameters and protocol functions.

The F-CSS session model allows to comprise different types of communications. A session is an association between two or more communicating applications and may consist of several application data streams, each of which is being processed by a separate protocol machine. F-CSS configures different protocol machines, each being tailored to the requirements in one

data flow direction. For configuration and management purposes, F-CSS provides several tools. A session configuration component receives session-setup requests from the local or the remote application and delivers a message to a protocol configurator to start the configuration of a suitable protocol machine for each individual data stream requested in the session-setup request. To allow a configuration based on protocol functions, a protocol description of PATROCLOS in a special description language (F-PDL), which has been developed within the F-CSS project, is required. The output of the F-CSS tools is directly passed to the connection management FSM of PATROCLOS, which initializes all other FSMs and delivers the selected protocol functions and parameters. Figure 3 shows the integration of PATROCLOS and the F-CSS components.

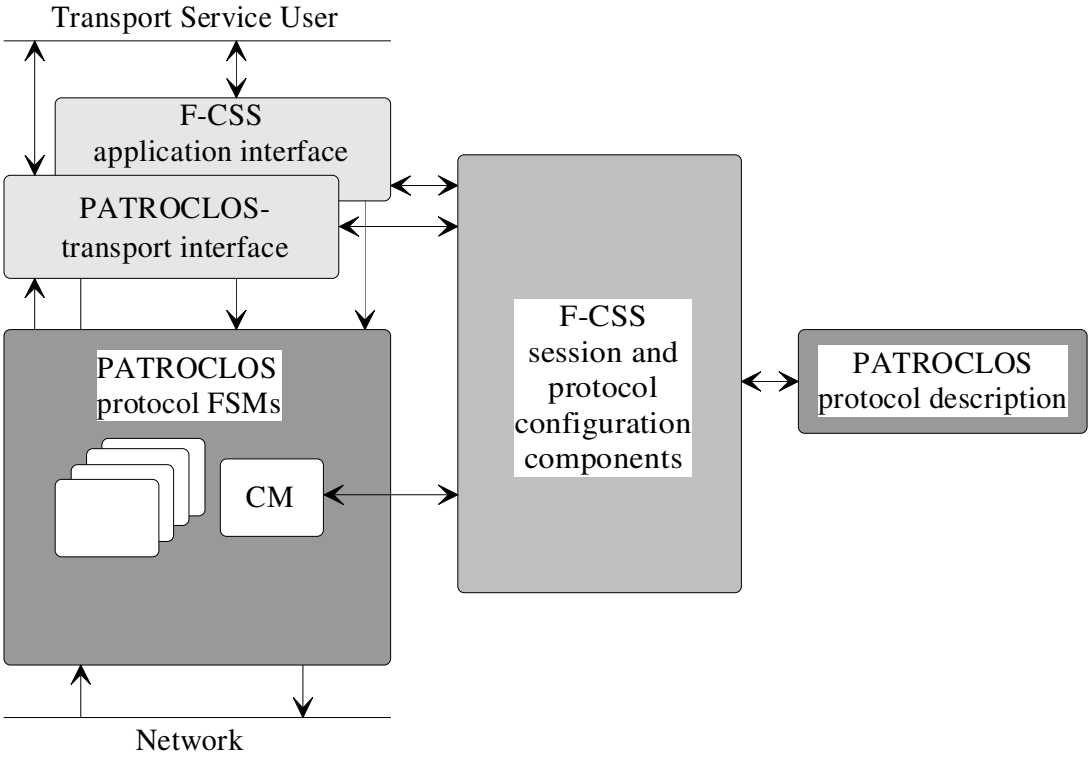

Figure 3: Integration of PATROCLOS and F-CSS configuration components

2.3. Protocol Functions and Mechanisms

In addition to several types of connection establishment and termination, the protocol offers several options for data transfer. These options may be specified by the transport service user or may be calculated by the F-CSS configuration tools. Table 2 gives an overview of the offered protocol functions.

User data may be delivered to the transport service user as TPDUs (P-frames), i.e. all received TPDUs are delivered in the same order they arrived, or as TSDUs. For delivery on a TSDU basis a sequenced or an unsequenced delivery can be selected.

For unreliable data transfer, retransmission can be switched off. For reliable data transfer, selective or go-back-N strategy can be selected. While selective retransmission has benefits in networks with a large bandwidth-delay product, it requires significantly more buffer memory at the receiver. For go-back-N strategy, however, the data send and the retransmission FSM are not able to work as independent as for selective retransmission strategy. When the retransmis-

sion FSM detects that a go-back-N retransmission is required, it first stops the data send FSM by sending a message to it. The response of the data send FSM contains a sequence number, which indicates the user data already sent. The retransmission FSM retransmits the user data already sent sends a signal to the data FSM to indicate that all retransmissions have been done and that the normal data transfer can proceed.

Table 2: Protocol functions and parameters

Protocol Functions	Protocol Function Parameters
Reassembly	delivery of TSDUs (reassembly) delivery of TPDUs (no reassembly)
Sequencing	no sequenced delivery of TSDUs to the transport service user
Retransmission	no selective repeat Go-Back-N
Flow control	no Stop-And-Go window-based rate control Stop-And-Go and rate control window-based and rate control
Congestion control	no 2P algorithm AACC algorithm

Window-based, rate-based, and stop-and-go mechanisms are provided for flow control. The flow control send FSM controls data flow by the interpretation of the peer's feedback signals. It signals periodically the maximum sequence number to be sent (limited by window-based flow control) to the data send FSM. For rate-based flow control, it starts a rate timer, calculates the maximum sequence number of user data to be sent for the next time period, and signals this value to the send control FSM, which controls that no user data with an illegal sequence number are sent.

Based on test P-frame delays and inter-arrival times, congestions are detected and the rate can be decreased by sending signals to the flow control send FSM. Currently, we use the adaptive admission control (AACC) algorithm [16] and the packet pair algorithm (2P) [17]. Both algorithms have slightly been modified. The original 2P algorithm sends packet pairs with a certain inter packet gap. Dependent on the gap between the corresponding acknowledgments, the inter packet gaps are decreased or increased. PATROCLOS does not use user data packets for this algorithm but special test packets, which take the same path as the user data packets. Furthermore, PATROCLOS does not consider the acknowledgment gaps, but the gaps of the test packets at the receiver. Therefore, the congestion control receive FSM has to inform the congestion send FSM about the detected inter arrival times of the test packets. That means that in PATROCLOS possible congestions in the reverse direction do not influence congestion detection in the other direction. Another difference is that the congestion analysis results in a modified rate in contrast to the original algorithm, which modifies the inter packet gap. The AACC algorithm is based on a calculation of a virtual delay between sender and receiver. The virtual delay is the difference between the sending time at the sender and the receiving time at the re-

ceiver. Dependent on increasing or decreasing virtual delays the inter packet gaps are increased or decreased. Again, in PATROCLOS we calculate and modify rate parameters. Therefore, the flow control receive FSM sends the flow control parameters not only to the flow control send FSM but also to the congestion send FSM. This is done with a single P-control-frame, which is addressed to both FSMs. A demultiplexer component has to deliver the P-control-frame to both FSMs.
Because of the high degree of modularity, the FSMs can be configured highly independent of each other. There are only a few protocol functions, which have influence to the behavior of other FSMs. Most of the functions, which can be configured have influence on one or two FSMs.

2.4. Other Parallel Transport Protocols Approaches

In addition to XTP, there are other transport protocols supporting parallel implementations. SNR [7] is a transport protocol to support high-speed communication in datagram and connection oriented networks. SNR has been developed at AT&T Bell Laboratories for parallel implementation on a M68030 based multiprocessor architecture. Control and user data transfer have been separated by exchanging control information periodically and independently from user data exchange. The protocol processors on send side and on receive side have to share a common memory. This fact allows an implementation with two memory busses, where control and user data packets are flowing across the same memory bus. Another difference to PATROCLOS is that retransmission processing and regular user data transfer are performed by the same process.
The MultiStream Protocol (MSP) [18] is also decomposed into several units, which are called protocol machines. MSP is based on concepts introduced by the HOPS architecture (HOPS: horizontally oriented protocol structure) [4]. The transport service user can switch off or on protocol functions for acknowledgment, retransmission, and delivery to the transport service user by selection of one of eight so-called streams. The offered protocol functions can not be selected independently of each other and user data transfer is not fully separated from control functions, e.g., acknowledgments are exchanged by piggy-backing.
TP++ [19] is an approach of high-speed protocol designed at Bellcore. Special data formats shall enable independent user data and control processing. Furthermore, most of the protocol functions can process incoming packets in an arbitrary order.
HTPNET [9] is another protocol, which has been strongly influenced by SNR. HTPNET has been developed at the University of New South Wales (Australia) and consists of four finite state machines, which are the result of separating a protocol into send/receive and data/control part. Error control, flow control, and rate control can be switched on or off. The combinations result in eight protocol classes.

3. IMPLEMENTATION

The main advantage of the PATROCLOS protocol architecture with its modular FSMs is its support for parallel implementation. This section describes the PATROCLOS implementation for an ATM network on a hybrid multiprocessor architecture, which uses transputers as universal processors and several specialized hardware units. Another implementation architecture for PATROCLOS consists of VLSI components only [20]. In contrast to other approaches (e.g., [21]), the transport protocol is processed outboard to prevent the host from transport protocol processing. Another reason is that with outboard processing no control data such as acknowledgments must cross the host bus.

3.1. Mapping of FSMs to Parallel Processes

The interface and protocol FSMs of PATROCLOS have been implemented as processes. For each FSM there is only one process, which can handle several connections. Processes of the same entity communicate with each other via asynchronous message passing across interconnecting inter-process channels. Timer processing is supported by special timer processes, which do not need to run on the same processor as the protocol process, and, therefore, may run in parallel. Protocol and timer processes co-operate by message passing, too.
A parallel C language with additional elements for message passing and process management has been used for implementation. Figure 4 shows the data receive process as an example for the general process implementation structure. Other processes have the same process structure.

```
  process data_receive(){
  while (TRUE){
   read_message_from_queue();  /* receive message
*/
   switch (message_type){
    case FRAME_FROM_NETWORK:   /* processing step 1
*/
     analyze_frame();
     deliver_frame();
     break;
    case TIME_OUT:             /* processing step 2
*/
     send_info_to_ack_and_flow_control();
     restart_timer;
     break;
    ...
  }}}
```

Figure 4: Data Receive Process Implementation

Different processing steps have to be performed dependent on the incoming message type, which is received by the function `read_message_from_queue`. Processing step 1 includes all operations for an incoming packet from the network. The packet is analyzed (e.g. sequence control, lifetime control, address analysis etc.) and delivered to the reassembly process, if it is error-free. Because the acknowledgment process must have some knowledge about correctly received or missing packets, the data receive process has to send informations to that process. Usually, the data receive process submits these informations periodically. To control the information exchange, the data receive process starts a timer by sending a message to a timer process. The timer process sends a message back to the data receive process to signal a timeout. In processing step 2, the data receive process reacts upon the timeout message from the timer process by sending informations about received packets to the acknowledgment process. In the same processing step buffer state informations are sent to the flow control process. Finally, the data receive process restarts the timer. In addition to temporal parallelism (pipelining), a high degree of spatial parallelism by parallel user data and control processing is achieved. Furthermore, timer processing is an appropriate candidate to be performed in parallel.
As mentioned above the protocol architecture supports a variety of different protocol functions. The selected protocol functions and parameters are delivered during the initialization phase of a connection by the connection management FSM to all other processes. Each process extracts the relevant informations and stores them in the local context. The process code is constructed in a way, that a process can perform all possible combinations of mechanisms and parameters. The required operations are selected by if-then-else elements or by function point-

ers, which are also stored in the local context. Because only a few protocol functions influence the behavior of a FSM, only a few if-then-else statements or function pointers in the implementation are required. This strategy has the advantage that we can share the same code for several connections, which require different protocol functions and parameters. The amount of code and the local memory space is considerably reduced.

3.2. Hybrid Multiprocessor Architecture

Protocol functions on P-frame and TSDU level are mostly implemented in software, while hardware is required for cell level functions. The combination of universal processors and dedicated hardware results in a hybrid multiprocessor architecture. The complete implementation architecture has to be connected via a DMA unit to a host system. Figure 5 shows the implementation architecture and the processor module for the user data receive FSM.

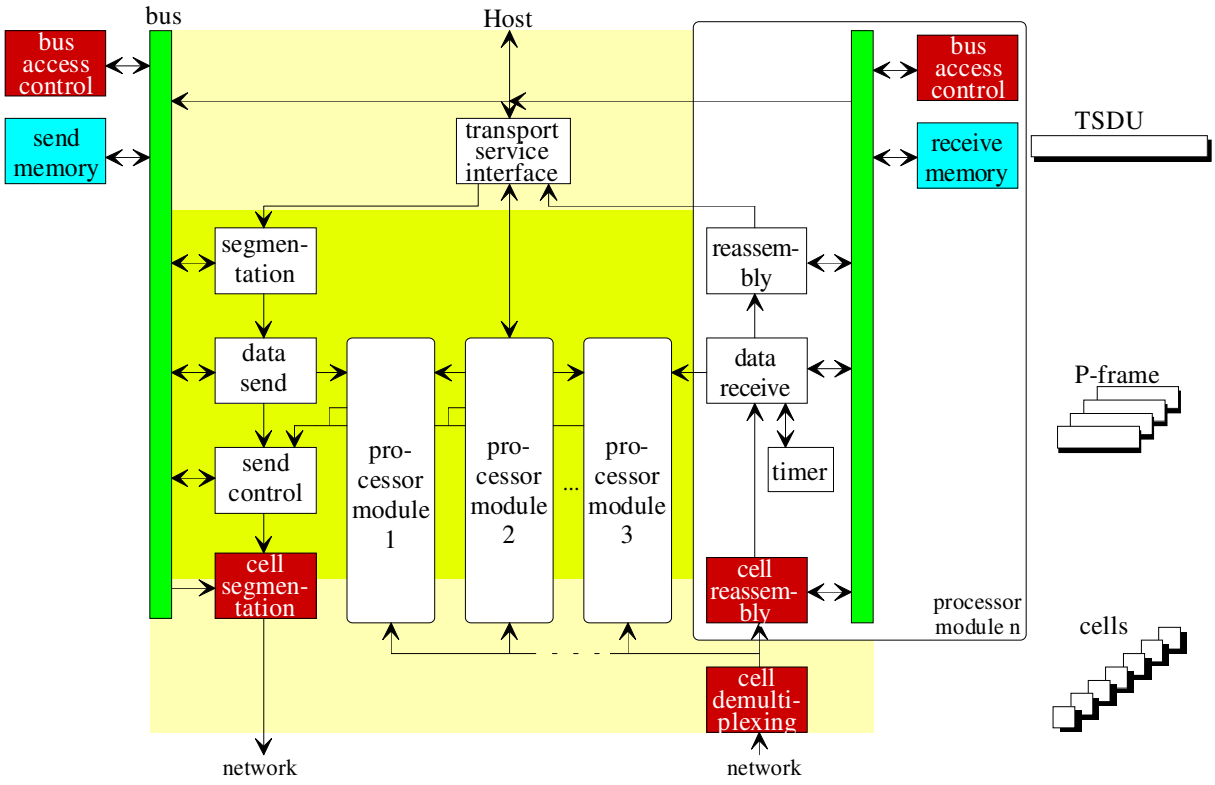

Figure 5: Hybrid Implementation Architecture

Each process derived from the protocol FSMs of the PATROCLOS protocol architecture may be mapped to a processor module of a suitable multiprocessor configuration. Furthermore, the processes resulting from implementation of the F-CSS components can run on one or more separate processor modules. A processor module consists of one or more universal processors (in our case transputers) for FSM processing, a cell reassembly unit to reassemble ATM cells into P-frames, a shared memory, and a bus access control unit to control the concurrent access to the shared memory by the processors and the reassembly units (cf. Figure 5).
Transputers are interconnected by hardware channels for inter-processor communication, which is performed in parallel to other computations on transputers. Memory management is distributed among several processors and requires special data structures, which have been designed in order to minimize copying of data and to permit an efficient memory management and access by several processors without any inconsistencies and conflicts [22].

The bus access control units and the reassembly processing unit on cell level (according AAL 3/4) have been implemented by dedicated hardware components to support the high network bandwidth [22]. Other hardware components are a cell segmentation unit on send side and a demultiplexing component on receive side to perform demultiplexing below the cell reassembly level and to distribute very efficiently incoming cells to the reassembly hardware units of the processor modules. In contrast to other approaches (e.g., [23]), which use linked lists for reassembly, incoming cells are reassembled such that consecutive cells are stored in consecutive memory areas.

The demultiplexer is the interface between the network access unit and the processor modules. A similar component to support efficient demultiplexing has been proposed in [24]. The demultiplexer [22] has to distribute incoming P-frames to the appropriate processor module, but distribution is performed on the level of ATM cells. The corresponding processor module of a P-frame is identified by the FSM address, which addresses a single FSM and is an extension of the transport entity address. The address extension is a simple bit array. Because each bit represents a certain FSM, the address format allows a simple mapping of an FSM address to the processor modules. The FSM address is always located in the P-frame header of a PATROCLOS protocol data unit (P-frame), and therefore in the first cell of a P-frame (BOM cell, begin of message). All other cells of a P-frame do not contain the FSM address. ATM cells of a single P-frame are identified by a unique value of VCI (virtual channel identifier), VPI (virtual path identifier), and MID (multiplex identifier). The FSM address is determined dependent of the cell type and delivered to the select generator. The FSM address for BOM and SSM cells can be extracted directly from the cell body, COM and EOM cells require a table look-up in a look-up table, which is usually realized by a content addressable memory (CAM).

4. PERFORMANCE ANALYSIS

The demultiplexer implementation provides a high degree of parallelism: writing cells into the FIFO buffer, determination of the FSM address, and reading the cells from the FIFO buffer are performed in parallel. Therefore, the demultiplexer needs only 55 clock ticks to process a 53 byte ATM cell assuming a 40 MHz clock. This results in a throughput of more than 700 000 cells/s (300 Mbit/s). The throughput of the demultiplexer has to be higher than the throughput of the reassembly units, which can reassemble up to 470 000 cells/s in our implementation. Generally, the throughput values of the hardware components are limited by the used FIFO and shared memory technology.

The following analysis discusses the achievable software performance of the receive part, which requires the most processing time of all receive processes and is the bottleneck of the PATROCLOS implementation. To measure the processing times for receive processing (cf. Figure 6), we used a hybrid monitor system. We implemented the protocol with a parallel C language, but we did not yet optimize the code. For the performance measurements, we selected rate-based and window-based flow control, selective retransmission, and inactive congestion control. Furthermore, we did not perform segmentation and reassembly in the measurement scenario. However, the lack of congestion control and segmentation/reassembly does not influence the performance as shown below. For performance measurement we implemented additional test environment processes for the transport user and a loopback driver.

4.1. Receive Throughput

To measure the performance for receiving, all processes of the send part, which are inactive during receive processing, are mapped to a single processor. Each of the receive processes has

an own processor. Because P-control-frames are sent and received periodically and, therefore, less frequently than P-data-frames, processing of received P-data-frames will be the critical path. Figure 7 shows a more detailed analysis of the data receive process, which is the bottleneck of receive processing although running on a separate processor. In addition to analyze the received data packets, the data receive process informs the acknowledgment process periodically about correctly received P-data-frames and the flow control receive process about the buffer state. The data receive process sends messages to the acknowledgment process and the flow control receive process (processing step 2, cf. Figure 4) for every five P-data-frames (packets). Receiving a data packet is indicated by processing step 1. Each processing step is performed after receiving the corresponding message.

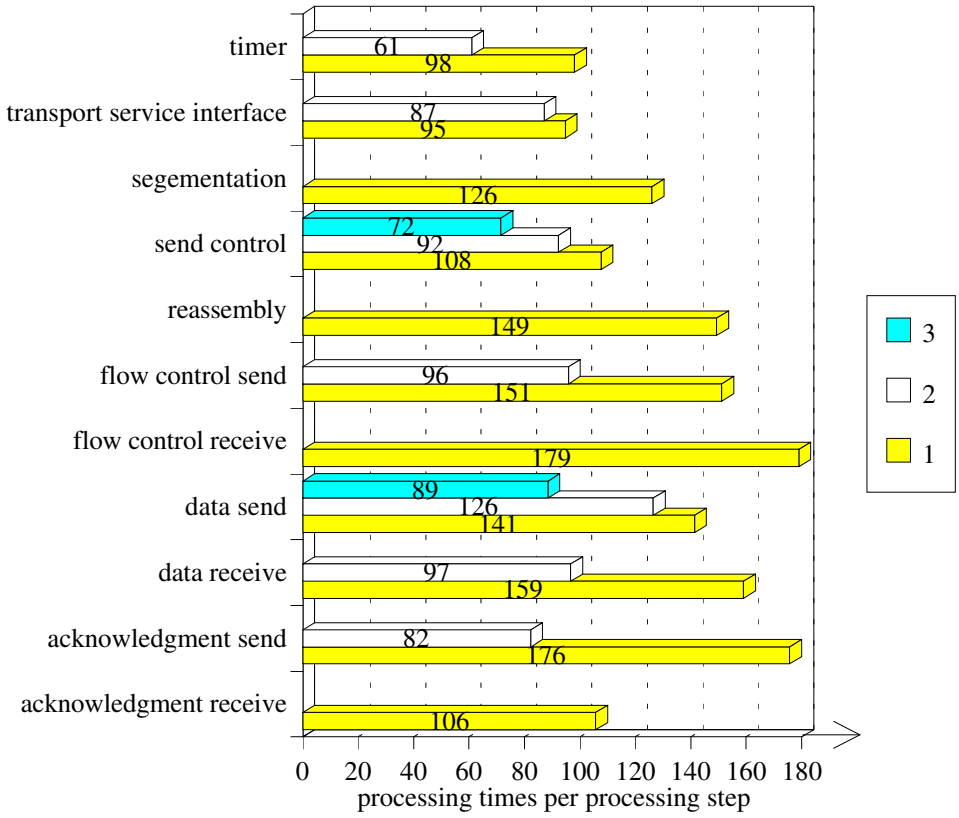

Figure 6: Receive Processing Times

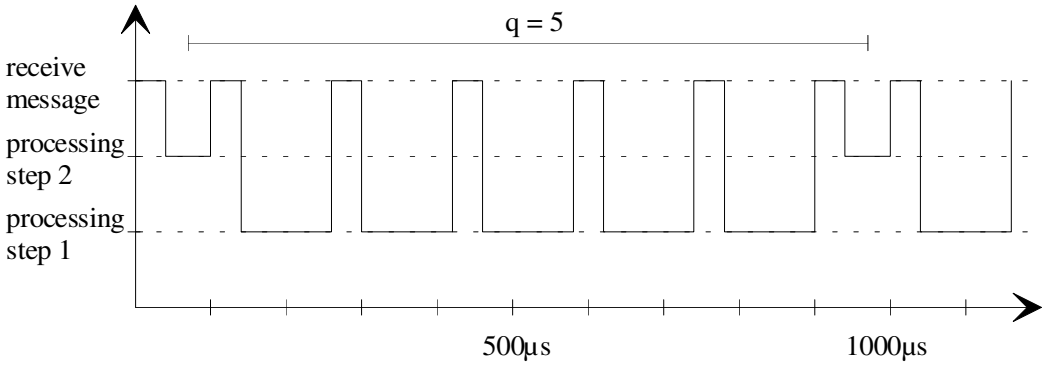

Figure 7: Analysis of the Data Receive Process

With q as the number of packets received per acknowledgment information, and $t_{dr,x}$ as the time of the data receive process for processing step x, the throughput P_r (receive performance) of the data receive process and, therefore, for receive processing is calculated by

$$P_r = \frac{1}{t_{dr,1} + q \cdot t_{dr,2}}$$

Increasing the value of q results in a higher performance. The throughput exceeds 5000 packets/s for q > 5 and 6000 packets/s for q > 10, but it is limited to 6300 packets/s by the time of the data receive process to analyze a P-data-frame (159 µs). The data receive process will also be the bottleneck, if packets have to be reassembled, because the reassembly process reassembles packets faster than the data receive process analyzes them. Congestion control has no influence on P-data-frame processing and does therefore not influence receive performance. The throughput values of the receive path have been calculated based on the processing times of the receive processes and have been verified by real measurements. The measured and the calculated results differ only by 2 %. Asynchronous inter-process communications allow very exact performance predictions.

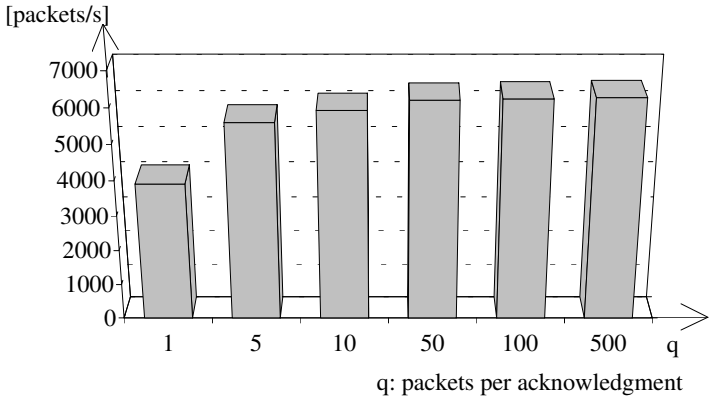

Figure 8: Throughput of receive path

4.2. Send Throughput

Similar to the analysis of the receive path, we evaluated the performance of the send path. Two parameters influence the performance of the data send process, which is the bottleneck process of the send path. The data send process has to signal periodically informations to the retransmission process, which describe the user data already sent (processing step 3). The time of this period must be lower than the round trip time, which includes propagation delay as well as processing of user data and acknowledgments at the sender and the receiver. Therefore, rtt will exceed 1 ms. Furthermore, the data send process receives a credit information message every q packets from the flow control receive process (processing step 2). Processing the P-data-frames received from the segmentation process is performed in processing step 1. Similar to the data receive process we can derive a formula for the throughput P_s (send performance) with $t_{ds,y}$ for the time of the data send process for processing step y:

$$P_s = \frac{1 - \frac{t_{ds,3}}{rtt}}{(q \cdot t_{ds,2} + t_{ds,1})}$$

The throughput depends on the parameters q and rtt as shown in Figure 9. For q > 5 we achieve more than 6000 packets/s. The throughput is limited to 8000 packets/s by the time of 126 µs required for processing step 1.

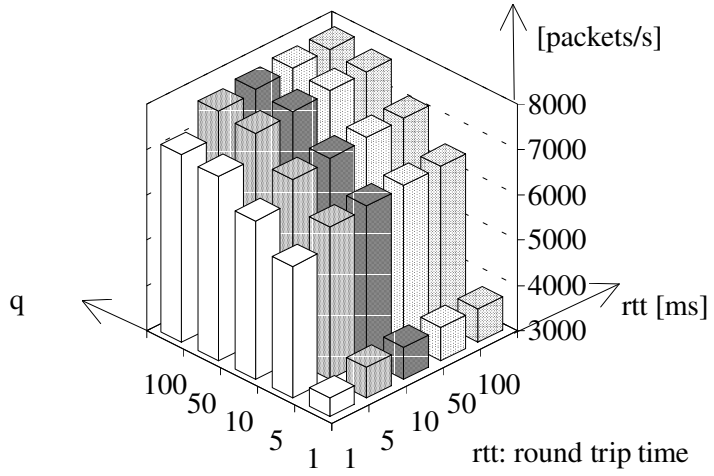

Figure 9: Throughput of Send Path

4.3. Speed-up

To compare the performance of the parallel implementation with an implementation on a single processor, we calculated the single processor performance by subtracting the overhead caused by interprocess communication from the parallel processing times. As illustrated in Figures 11 and 10, the parameters rtt and q have different effects on the performance. Decreasing rtt and q results in a lower performance because of the increasing overhead for control processing. The effects of control processing are stronger for an implementation on a single processor. In a parallel implementation, control processing has less impact on throughput performance, because control processing is performed by special processors. Therefore, speed-up increases with a high control processing overhead. This result is a major benefit of the presented parallel implementation. Usually, the speed-up of the discussed implementation will be 2-2.5 (receive path, cf. Figure 11) and 2.5-3.5 (send path, cf. Figure 10) dependent on the parameters q and rtt [22]. The cases, where the speed-up values are even higher, are unusual cases with an extremely frequent control information exchange.

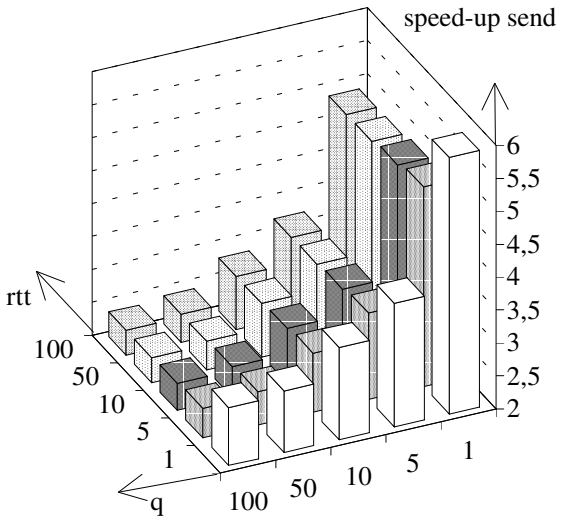

Figure 10: Send Speed-up

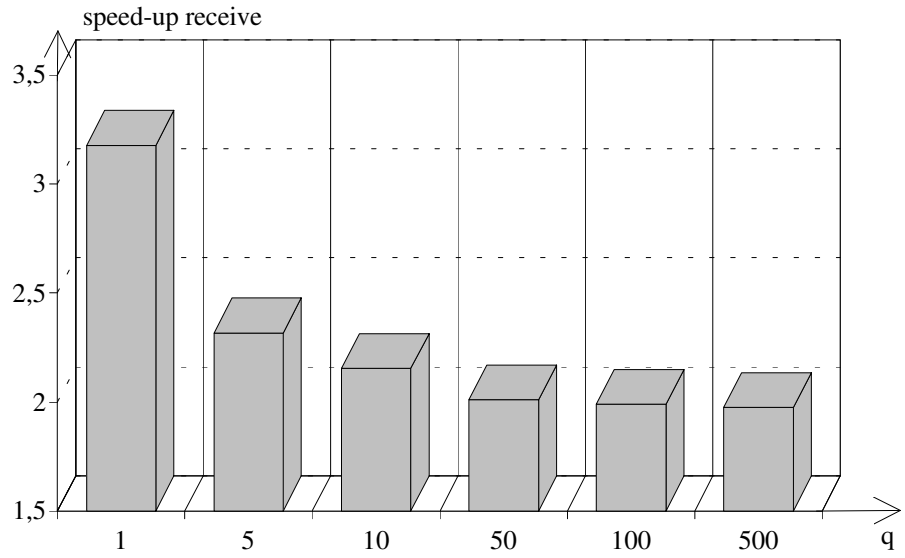

Figure 11: Receive Speed-up

4.4. Discussion

Compared to other transport protocol implementations on transputers, the PATROCLOS implementation achieves a very good performance. A TCP/IP implementation [5] running on the same platform as used for performance evaluation of PATROCLOS achieves less than 3000 packets/s and lower speed-up values. The bottleneck processes of that TCP/IP implementation require approximately the double processing time as the PATROCLOS bottleneck processes. One reason for this difference is that a lot of control functions such as acknowledgment processing or flow control, which are within the critical path of conventional protocols, have been moved to special FSMs of the PATROCLOS architecture and can be performed in parallel.
Our experiences cope with other researchers that it is difficult to separate a single packet into pieces and process these pieces in parallel efficiently [25]. However, the separation of control functions from user data processing functions is a very successful approach, because it improves the critical path of user data processing significantly and allows a more efficient parallel processing. In addition to control/data parallelism, pipelining is also a concept of parallelism that is applicable to transport protocol processing, especially if time-consuming functions as segmentation and reassembly are part of the critical path.

5. CONCLUSIONS

The general goal of PATROCLOS has been an integrated design of protocol and implementation architecture issues. Performance evaluations show significant improvements compared to similar implementations of other protocols. The good performance results, which we achieved without any code optimizations, indicate that a protocol design appropriate for efficient parallel implementation is very useful.
In addition to parallel processing, the presented implementation architecture has the advantage that the memory bottleneck, which usually occurs in network interfaces and is a very serious problem, can be reduced to a minimum, because we provide separate memory busses for incoming and outgoing user data. Because in contrast to other network interfaces, control information packets do not flow across the user data bus, the full bandwidth of the busses can be

used for user data transfer. Furthermore, implementing a transport subsystem with a transport service interface on outboard processors has the benefit that host interrupts and DMA transfers between network interface memory and host memory depend on the frequency of TSDUs. Generally, the results indicate that multiprocessor systems with several independent memory busses may be applicable candidates for future high-speed networks.

Although we developed a special protocol, which is well-suited for the parallel implementation architecture, some of the presented concepts, e.g., parallel processing of acknowledgments and user data, can also be applied to standard protocols such as TCP/IP, if we are able to extract control information from the data flow in the same efficient way as the demultiplexer of the PATROCLOS implementation. However, TCP/IP requires much more complex header parsing and composition functions, which are not easy to implement. Because of this fact, the drawbacks of TCP/IP in high-speed networks with large bandwidth-delay products [3], and the lack of the required flexibility to support a large variety of different applications, we developed the PATROCLOS transport subsystem. The PATROCLOS FSMs provide a lot of different protocol functions and can be configured dependent on application and network requirements. The modular protocol architecture simplifies the implementation of different kinds of transport service interfaces.

Recent work integrated advanced scheduling algorithms into the implementation to support quality-of-service guarantees and multicast extensions to be able to provide reliable and unreliable multicast services.

REFERENCES

[1] Danthine, A.; Baguette, Y.; Leduc, G.; Leonard, L.: *The OSI 95 Connection-mode Transport Service: The Enhanced QoS*, in: Danthine, A.; Spaniol, O.: High Performance Networking, IV, IFIP, North-Holland, 1993, pp. 397-412

[2] Zitterbart, M.; Stiller, B.; Tantawy, A.: *A Model for Flexible High-Performance Communication Subsystems*, Journal of Selected Areas in Communication, Vol. 11, No. 4, May 1993, pp. 507-518

[3] LaPorta, T.F.; Schwartz, M.: *Architectures, Features, and Implementation of High-Speed Protocols*, IEEE Network Magazine, Vol. 5, No. 3, May 1991, pp. 14-22

[4] Haas, Z.: *A Protocol Structure for High-Speed Communication over Broadband-ISDN*, IEEE Network Magazine, Vol. 5, No. 1, January 1991, pp. 64-70

[5] Rütsche, E.; Kaiserswerth, M.: *TCP/IP on the Parallel Protocol Engine*, in: Danthine, A.; Spaniol, O. (eds.): High Performance Networking, IV, IFIP, North-Holland, 1993, pp. 119-134

[6] Björkman, M.; Gunningberg, P.: *Locking Effects in Multiprocessor Implementations of Protocols*, ACM Sigcomm'93 Conference Proceedings, San Francisco, CA, September13-17, 1993, pp. 74-83

[7] Sabnani, K.; Netravali, A.; Roome, W.: *Design and Implementation of a High Speed Transport Protocol*, IEEE Transactions on Communications, Vol. 38, No. 11, November 1990, pp. 2010-2024

[8] Strayer, W.T.; Dempsey, B.J.; Weaver, A.C.: *XTP: The Xpress Transfer Protocol*, Addison-Wesley Publishing Company, 1992

[9] Chan, T.S.; Gorton, I.: *HTPNET: A New Transport Protocol for High-Speed Networks*, SCS&E Report 9402, School of Computer Science and Engineering, University of New South Wales, 1994

[10] Braun, T.; Zitterbart, M.: *Parallel Transport System Design*, in: Danthine, A.; Spaniol, O. (eds.): High Performance Networking, IV, IFIP, North-Holland, 1993, pp. 397-412

[11] Braun, T.; Zitterbart, M.: *A Parallel Implementation of XTP on Transputers*, 16th Annual IEEE Conference on Local Computer Networks, October 14-17, 1991, Minneapolis, USA, pp. 172-179

[12] Clark, D., Tennenhouse, D.: *Architectural Considerations for a New Generation of Protocols*, ACM SIGCOMM ´90, Philadelphia, U.S.A., September 24-27, 1990, pp. 200-208

[13] Sterbenz, P.G.; Parulkar, G.M.: *Design of a Gigabit Host-Network Interface*, Journal of High Speed Networks, Vol. 2, 1993, pp. 27-62

[14] Holzmann, G.J.: *Design and Validation of Protocols: A Tutorial*, Computer Networks and ISDN Systems, Vol. 25, 1993, S. 981-1017

[15] Watson, R. W.: *The Delta-t Transport Protocol: Features and Experiences*, in: Rudin, H. and Williamson R. (eds.): Protocols for High-Speed Networks, North-Holland, 1989, pp. 3-17

[16] Haas, Z.: *Adaptive Admission Congestion Control*, Computer Communication Review, Vol. 21, No. 5, October 1991, pp. 58-76

[17] Keshav, S.; Agrawala, A.; Singh, S: *Design and Analysis of a Flow Algorithm for a Network of Rate Allocating Servers*, in: Johnson, M.J. (ed.): Protocols for High-Speed Networks, II, North-Holland, 1991, pp. 55-72

[18] LaPorta, T. F.; Schwartz, M.: *The MultiStream Protocol: A Highly Flexible High-Speed Transport Protocol*, Journal of Selected Areas in Communication, Vol. 11, No. 4, Mai 1993, pp. 519-530

[19] Feldmeier, D.C.: *An Overview of the TP++ Transport Protocol Project*, in: Tantawy A. (ed.): High Performance Networks: Frontiers and Experience, Kluwer Academic Publishers, 1994

[20] Schiller, J.; Braun, T.: *VLSI-Implementation Architecture for Parallel Transport Protocols*, IEEE Workshop on VLSI in Communications, Stanford Sierra Camp, Lake Tahoe, CA, September 15-17, 1993

[21] Steenkiste, P.A.; Zill, B.D.; Kung, H.T.; Schlick, S.J.; Hughes, J.; Kowalski, B.; Mullaney, J.: *A Host Interface Architecture for High-Speed Networks*, Danthine, A.; Spaniol, O. (eds.): High Performance Networking, IV, IFIP, North-Holland, 1993, pp. 31-46

[22] Braun, T.: *A Parallel Transport Subsystem for Cell-Based High-Speed Networks*, Ph.D. Thesis (in German), University of Karlsruhe, Germany, VDI-Verlag, Düsseldorf, 1993

[23] Brendan, C.; Traw, S.; Smith, J.: *Hardware/Software Organization of a High-Performance Host Interface*, IEEE Journal of Selected Areas in Communications, Vol. 11, No. 2, february 1993, pp. 240-253

[24] Thekkath, C. A.; Nguyen, T.D.; Moy, E.; Lazowska, E.D.: *Implementing Network Protocols at User Level,* ACM Sigcomm'93 Conference Proceedings, San Francisco, CA, September13-17, 1993, pp. 64-73

[25] Partridge, C.: *Gigabit Networking*, Addison Wesley Publishing Company, 1994