A Parallel Transport Subsystem Implementation for High Performance Communication

Torsten Braun, Claudia Schmidt
Institute of Telematics, University of Karlsruhe, D-76128 Karlsruhe, Germany

Abstract

Requirements of emerging applications together with rapid changes in networking technology towards gigabit speeds require new adequate transport systems. Integrated designs of transport services, protocol architecture, and implementation platforms are required by forthcoming applications in high-speed network environments. The transport subsystem PATROCLOS (parallel transport subsystem for cell based high speed networks) is designed with special emphasis on a high degree of inherent parallelism to allow efficient implementations on multiprocessor architectures combined with specialized hardware for very time critical functions. The paper presents the new parallel protocol architecture of PATROCLOS, an appropriate implementation architecture based on transputer networks, and performance evaluation results, which indicate high throughput values.

1. Introduction

During the last years, the characteristics of communication networks changed dramatically. The networks provide an increasing variety of services and their bandwidths are growing into the gigabit range. However, there is still a lack of adequate *transport systems*, which combine the functions of the four lower layers according to the OSI Reference Model, while t*ransport subsystems* cover only a subset of a complete transport system. Transport subsystems deliver the performance of the underlying networks to the application layers, transport subsystem *entities* use *protocols* for communication.

Parallelism in transport subsystem should be based on a relatively fine granularity together with low overhead (e.g., communication overhead among parallel building blocks) to support their efficient implementation on multiprocessor architectures and to increase their performance significantly [1]. Several parallel protocol implementations have already shown performance advantages [2, 3, 4, 5, 6, 7, 8, 9, 10, etc.]. Because parallelising standard protocols (e.g., OSI or TCP/IP) suffers from their inherent sequential structure, PATROCLOS realizes a parallel protocol architecture with fine granularity adequate for parallel implementations on hybrid multiprocessor platforms.

In addition, several characteristics of emerging cell based networks such as ATM [11] and DQDB [12] influenced the protocol design for the transport subsystem [13] and a number of protocol mechanisms (e.g., connection management, error control, flow control) needed to be changed in order to take advantage of the provided network bandwidth [10]. Moreover, since a large amount of data may be traveling between communicating end systems, it is impractical to maintain a consistent view of the state in both systems. The exchange of connection state information piggy-backed in user data can not be recommended for high-speed communications. Mechanisms based on state exchanges after dedicated requests (e.g., XTP [14]) or periodic state exchanges (e.g., SNR [3]) seem to be more suitable.

The parallel transport subsystem PATROCLOS presented in this paper attempts to overcome the performance limitations by the integration of protocol and implementation issues and by providing the required functionalities between the media access (MAC) layer interface of cell-based networks and the transport layer interface in a single modular protocol component.

2. Design Guidelines for Parallel Transport Subsystems

Considering processing speeds and memory bandwidth as major bottlenecks for high speed communication subsystems, the use of multiprocessor platforms is an adequate approach to increase their performance. Different levels of granularity have been chosen for several approaches in parallelising standard protocols. Most of them remain at a level that is based on protocol entities or even complete protocol stacks. The approach followed in the PATROCLOS design uses a finer granularity with a protocol function oriented decomposition into basic building blocks.

A sophisticated protocol example supporting parallelism is XTP [14]. Parallelism has been considered in the design process and has resulted in the specification of multiple concurrent extended finite state machines (FSMs) [15]. Several XTP features, such as the integration of transport and network layer, the fixed header format etc., are extremely helpful in designing a parallel high performance implementation on multiprocessor architectures such as transputer networks. The XTP implementation developed at the Institute of Telematics is able to process send and receive part in parallel without significant interactions [4]. Processing of information and control packets can be performed partly in parallel. Communication overhead caused by event signalling among the FSMs and by context information updates is the major limiting factor.

Based on the analysis of the XTP FSMs and the performance results of the corresponding parallel implementation, we derived several guidelines for protocol architectures appropriate for parallel processing [13]:

- *Reducing data dependences among protocol functions*
- *Decoupled control and user data processing*
- *Modular function-oriented decomposition*
- *Reducing event signalling among parallel function units*
- *Minimizing common resources among function units*

3. Transport Subsystem Architecture

The presented design issues influenced the architectural concept of PATROCLOS, which combines a modular design and the avoidance of layering to achieve a high degree of parallelism. Another reason to avoid layering is to minimize redundant protocol functions in different layers. The transport subsystem covers the functionality of the OSI layers between MAC service interface and transport service interface (e.g., in IEEE 802.x LANs: layers 2b-4, in ATM: AAL and higher layers below transport service interface). This section describes the decomposition of the protocol into autonomous function units to support the implementation on a parallel architecture. It introduces the concept of parallel finite state machines, presents the specification based on these FSMs, and describes the functionalities of the single FSMs.

3.1. Classification of Protocol Functions

PATROCLOS is mainly designed around a set of independent protocol functions [13]. There is a high degree of independence among functions involved in send and receive processing. Another approach is the parallel execution of control and user data processing functions. For this reason, the protocol functions are classified as transfer control, data access and data control functions (cf. Figure 1). A similar classification is presented in [16]. ***Data access functions*** process user data and can be divided into the subclasses *data manipulation functions* and *data analysis functions*. ***Data control functions*** make decisions to select the operations, which have to be performed on user data. Data control functions can also be divided into two subclasses *send control functions* and *receive control functions*. Data control and data access functions are called ***data transfer functions***. ***Transfer control functions*** (e.g., connection management,

flow control) control the connection and its user data flow and exchange control information for control and management purposes.

<table>
<tr><td colspan="4">protocol functions</td></tr>
<tr><td colspan="3">data transfer</td><td rowspan="3">transfer control</td></tr>
<tr><td colspan="2">data access</td><td colspan="2">data control</td></tr>
<tr><td>data manipulation</td><td>data analysis</td><td>send control</td><td>receive control</td></tr>
</table>

Figure 1: Classification of protocol functions

3.2. FSM System Definition

The components of the PATROCLOS architecture are realized by parallel FSMs. FSMs belonging to the same PATROCLOS entity exchange messages for synchronisation and communication purposes. Together, they form a system of extended finite state machines (FSM system) with the functionality of a transport subsystem. A FSM is defined by the quintuple [17]:

$$\langle A_e, A_a, S, s_0, \Delta \rangle$$

A_e: a finite set of input characters
A_a: a finite set of output characters
S: a finite set of states with main states and variables
s_0: a special element of S (initial state)
Δ: set of transitions, i.e., relations of $(S \times A_e \cup \{\varepsilon\}) \times (A_a \cup \{\varepsilon\}) \times S$, with the null element ε

Several FSMs, which interact by message exchange over channels, can be combined to a FSM system (cf. Figure 2). The input and output characters encode the message types and the channels, on which the interactions take place. A channel addressing function performs the mapping of messages to channels. The FSM system has two interfaces, an input interface allows external influence on the system, the output interface indicates the reactions of the system. A FSM system is defined by [17]:

$$\langle D, C, f_k, N_e', N_a', A_e, A_a \rangle$$

D: a finite set of FSMs
C: a finite set of channels with unlimited capacity. The channels are based on a FIFO service discipline.
N_e: union of the sets of input characters of the FSMs in D
N_a: union of the sets of output characters of the FSMs in D
f_k: channel addressing function, i.e., a partial mapping from $N_e \cap N_a$ to C
N_e': input interface message types
N_a': output interface message types
A_e: input mapping function, i.e., a partial mapping from N_e to N_e'
A_a: output mapping function, i.e., a partial mapping from N_a to N_a'

PATROCLOS can be characterized as a FSM system according to this definition. The FSMs belonging to the same entity exchange messages, which are classified as internal messages and external messages. Internal messages are exchanged between FSMs inside a FSM system. A special case of internal messages are timer messages. Timers are regarded as small FSMs, which receive commands to start or stop a timer and send time-out messages to other FSMs. External messages, which represent the interface messages of the FSM system definition, are used for the communication with other FSM systems, i.e., the external messages in PATROCLOS are messages from/to the transport subsystem user or the underlying network. The functions A_e and A_a perform together the mapping between the internal message types of a FSM system and the message types used at the interface.

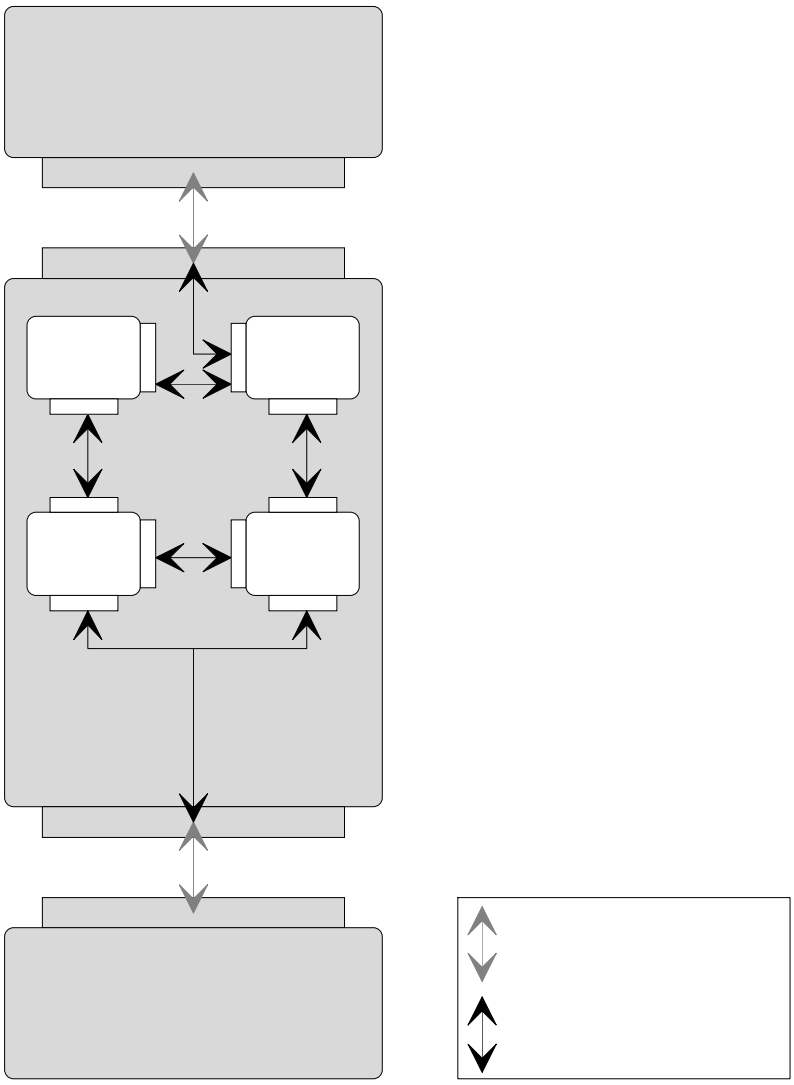

Figure 2: FSM systems

3.3. Parallel Protocol Architecture

PATROCLOS consists of two types of FSMs:

- *Interface FSMs* for interactions with neighbored layers, which are located at the interfaces of the transport subsystem.
- *Protocol FSMs* communicating by separate protocols with peer FSMs at the peer entity.

The interface of the PATROCLOS system is subdivided in two interface FSMs. These interface FSMs support interactions of PATROCLOS with FSMs, which represent the transport service user and the underlying network unit. The interface FSMs are only involved in local communications inside one entity, and they do not have to communicate with FSMs at the peer entity. The main task of the transport interface FSM is the mapping between service primitives and internal messages. It exchanges TSDUs with the transport service user, triggers actions of the protocol FSMs dependent on the received TSDUs, delivers TSDUs to the transport service user, and is involved in performing graceful connection termination. The network interface FSM controls and schedules transmission of PDUs (P-frames) to the underlying network unit (data control functions).

The protocol FSMs are designed according to the classification of protocol functions to decouple connection state information exchange (protocol FSMs for transfer control, cf. Figures 1, 3) from user data exchange (protocol FSMs for data transfer) effectively. Acknowledgments

and connection state information exchange are triggered by expired timers, explicit requests, or error situations. Protocol FSMs communicate directly with the corresponding peer FSMs. Each communication relation has its own protocol (FSM protocol, cf. Figure 3) and separate PDUs (P-frames). Multiplexing of P-frames by different FSMs is avoided to support a high degree of parallelism. P-frames containing user data are called P-data-frames in contrast to the P-frames containing control informations, which are called P-control-frames.
The FSM protocols use individual error recovery mechanisms and timers. The decomposition into independent FSMs permits parallel execution, while common connection state data bases are avoided. The transport subsystem has the benefit of a highly modular system, which can be configured easily dependent on the application requirements and the network characteristics. Configuration issues of transport systems are discussed in [18].
Furthermore, the transport subsystem is designed to minimize interactions among FSMs of the same entity, i.e., all FSMs are designed as autonomous as possible [19]. Because bi-directional communications reduce the achievable parallelism considerably, the communication among FSMs is uni-directional to allow a higher degree of parallelism. If possible, the necessary information exchange is performed periodically to reduce the communication overhead.

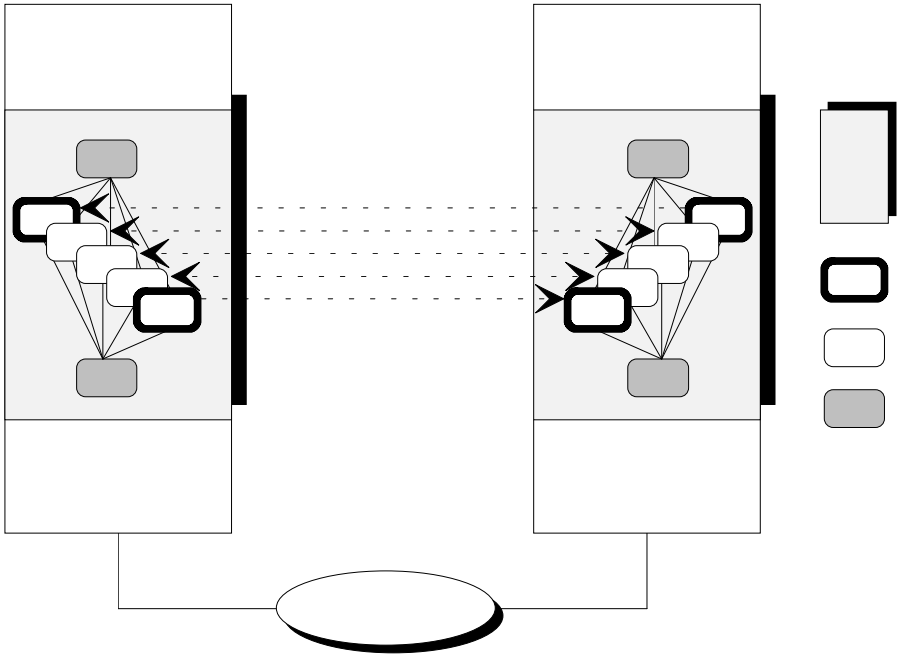

Figure 3: Architecture of PATROCLOS

3.4. Protocol FSMs

The PATROCLOS protocol is splitted into several FSM protocols. Each FSM protocol is performed by communicating peer FSMs, which use P-frames containing only the necessary information for their functions. This results in a parallel execution of different protocol functions. Moreover, most of the functions allow parallelism between send and receive part. All FSMs can be configured dependent on application and network requirements. Several strategies, mechanisms, and parameters are offered by the FSMs [20]. The formal specification of all PATROCLOS FSMs, which have been described in the PROMELA [21] language, has been simulated and validated with SPIN (simple PROMELA interpreter) [21]. The protocol FSMs are discussed more in detail in the following.

Connection management
A single connection management FSM manages the complete duplex connection. It supports establishment, monitoring, maintenance, and correct connection termination. Therefore, this

FSM 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. Handshake-based or implicit establishment procedures are provided according to the required service. To provide a graceful connection release, the connection management FSM has to ensure by co-operation with several other FSMs that all data, which are delivered to the transport system before initiating connection termination are sent and acknowledged.

User data transfer

Data transfer can be divided into a send path and a receive path. The send path includes most of the data manipulation functions, the receive path consists of most of the data analysis functions. The main tasks of the two 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, which results in two pipelines for the send and the receive path. The concept of pipelining increases the throughput of user data processing, which is the most time-critical part.

On send 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. There is another FSM in intermediate systems performing functions such as routing & relaying, route management, and monitoring.

Acknowledgment and retransmissions

Because data transfer often performs the critical path in communication protocols, retransmission and acknowledgment are handled by two special FSMs. The FSM on send side (retransmission FSM) receives acknowledgments from the peer FSM on receive side (acknowledgment FSM). The retransmission FSM may also request acknowledgments, or send the state of acknowledged data to the peer's acknowledgment FSM. Retransmission strategy (go-back-N strategy or selective repeat) and management of unacknowledged data are implemented in this FSM. Retransmissions depend on incoming acknowledgments from the peer's acknowledgment FSM.

Both FSMs co-operate with user data FSMs of the same FSM system. The data send FSM gives control over previously sent data packets to the retransmission FSM, which is responsible for further retransmissions to the peer's user data receive FSM. Communication between the FSMs is reduced to a minimum to avoid unnecessary overhead. Therefore, the data send FSM gives the control over the sent packets to the retransmission FSM periodically. The time period must be lower than one round trip time to be sure that the retransmission FSM receives the informations about a sent packet before the corresponding acknowledgment.

The data receive FSM signals informations about received data to the acknowledgment FSM, which generates acknowledgments based on these informations. In the case of error-free communication, the FSMs in one entity communicate periodically instead of exchanging messages after each packet. To avoid delays in error cases, the information is sent immediately after the error is detected.

Flow and rate control

Stop-and-go, window-based, and rate-based mechanisms can be selected or combined for flow control. Two flow control FSMs exist in each entity, one FSM for each data flow direction (cf. Figure 4). These FSMs negotiate flow control parameters with the remote peer and use these parameters to control the data flow.

On receive side, the user data receive FSM has to signal buffer state information to the flow control FSM. The flow control receive FSM calculates from these informations (e.g., buffer sizes) new flow control parameters (window sizes and rate values) and sends them periodically

to the remote peer's flow control send FSM, which controls the FSMs of the user data send functions by the received informations. The data send FSM is responsible for the window-based part of the flow control and, therefore, receives the new window limits from the flow control send FSM. The rate-based part of the flow control is performed by the flow control send FSM and the send control FSM. The flow control send FSM is interrupted periodically by the rate control timer to send a new credit value to the send control FSM, which decrements the credit value, whenever a new data packet has been sent. In addition, the congestion control send FSM may decrease the rate in the case of congestions. To minimize the communications among FSMs, messages are sent periodically if possible, e.g., buffer state information from the user data receive FSM to the flow control receive FSM. Other messages are exchanged only in exceptional cases, e.g., the user data send FSM requests a new credit, if it is blocked on the sliding window.

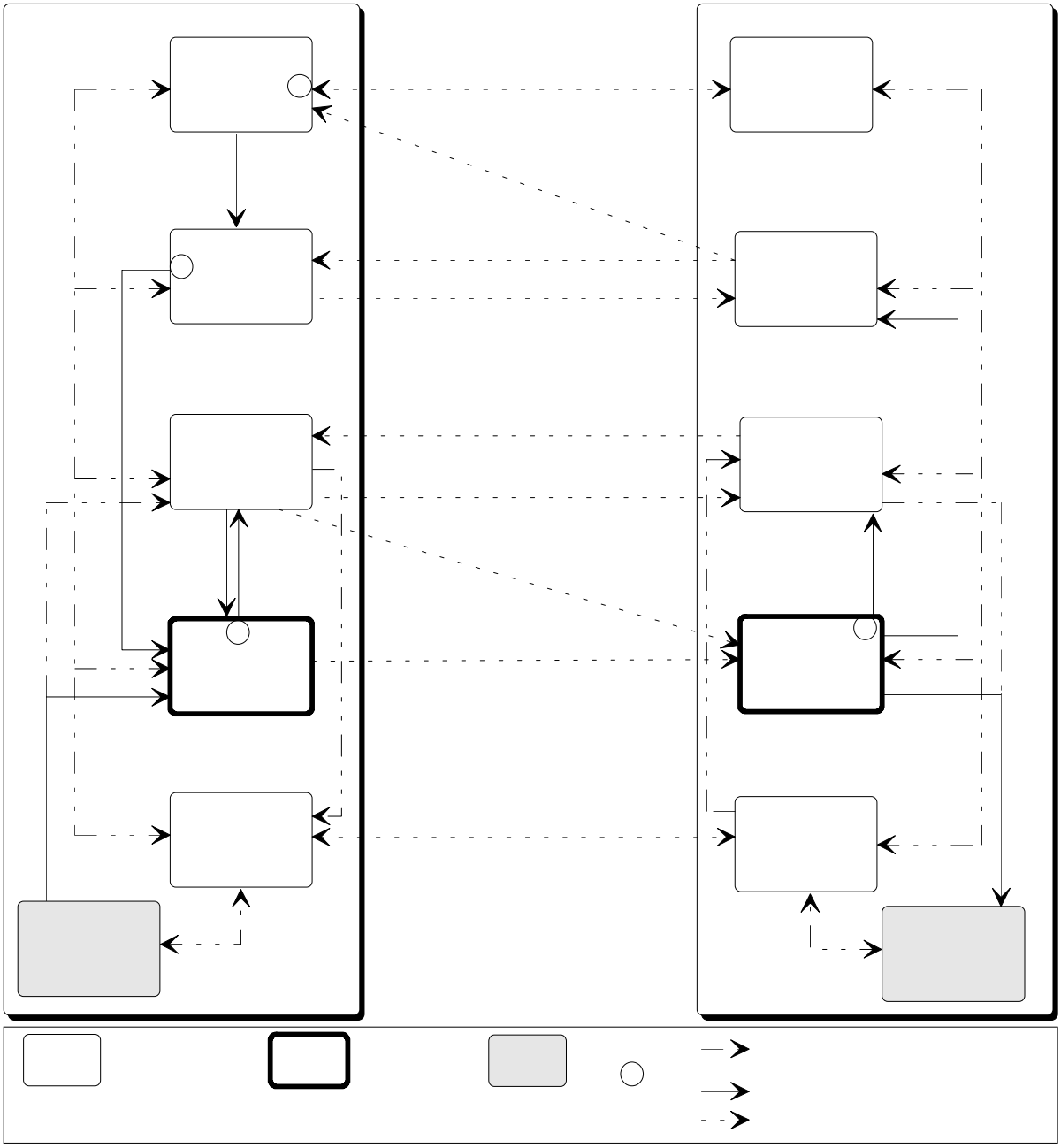

Figure 4: Co-operation among FSMs and FSM protocols

Congestion control

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 packets. The FSMs exchange test P-frames to identify congestions or to estimate round trip times. Based on test packet delays and inter-arrival times, congestion situations are detected and the rate of the sender can be decreased. The adaptive admission congestion control [22] and the packet pair algorithm [23] are used to detect congestions. The congestion control FSM may also react to the feedback signals of the network nodes.

4. Implementation on a Hybrid Multiprocessor Architecture

PATROCLOS is appropriate but not limited to be implemented on a hybrid multiprocessor architecture. Parallel architectures for communication subsystems are successfully built with transputer networks [2, 4, 5, 7, 8]. This section describes the mapping of the protocol specification based on the set of parallel FSMs to appropriate implementation building blocks and the developed hybrid multiprocessor architecture.

4.1. Process Implementation of FSMs

The FSMs have been implemented as processes (protocol processes). Each process may be mapped to a single processor of a suitable multiprocessor configuration. 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 be performed in parallel. A protocol process has to send command messages to the appropriate timer process to start, modify or stop timers. The timer process notifies the protocol process for each time-out by sending a time-out message.
Each protocol or timer process is supported by a subprocess (communication process), which allows asynchronous inter-process communication across synchronous channels and avoids deadlocks. The communication process receives incoming messages from another protocol or timer process over a synchronous inter-process channel and stores the received messages in a message queue. The receiving protocol or timer process can access to this message queue asynchronously and performs computations dependent on the contents of the received message.

```
process data_send () {
 while (TRUE) {
   read_message_from_queue();            /* receive a new message */
   switch (message_type) {
     case FRAME_FROM_SEGMENTATION:       /* processing step 1     */
       format_frame ();
       deliver_frame_to_send_control ();
       break;
     case MSG_FROM_FLOW_CONTROL:         /* processing step 2     */
       set_max_sequence ();
       break;
     case TIME_OUT:                      /* processing step 3     */
       send_info_to_retransmission ();
       break;
     ...
       break;
 }}}
```

Figure 5: Process implementation

For implementation, a parallel C language [24] with additional elements for message passing and process management has been used. The program in Figure 5 shows the structure of the data send process as an example for the implementation of protocol processes. Different proc-

essing steps have to be performed dependent on the incoming message from another process or a timer process. After each processing step, the process waits for the arrival of new messages (`read_message_from_queue`). In processing step 1 the data send process receives a message describing a segment from the segmentation process, formats the P-data-frame header and sends it to the send control process. In processing step 2, the maximal sequence number to be sent is received from the flow control process. After a time-out several informations about P-data-frames already sent are calculated and delivered to the retransmission process in processing step 3.

4.2. Multiprocessor Based Implementation Architecture

Each protocol or timer process derived from the PATROCLOS protocol architecture can be mapped to a separate processor on a multiprocessor-based implementation architecture, which uses transputers as universal processors (Figure 6). Processes on the same processor are interconnected by software channels, processes on different processors are interconnected by hardware channels (links). Transputers provide special capabilities to perform inter-processor communications concurrently with computations of any process. The implementation architecture will be connected via a DMA unit to a host system.
A demultiplexing unit to demultiplex incoming P-frames to the appropriate processor module, access control units to control the access of several processors to the shared memory busses, and protocol functions on cell level (SAR processing according to AAL3/4) have been implemented by dedicated hardware units to support the high network bandwidth. A processor module consists of one or more transputers, a reassembly hardware unit, and a shared memory with an access control to the memory bus. The combination of universal processors and specialized processors results in a hybrid multiprocessor architecture.

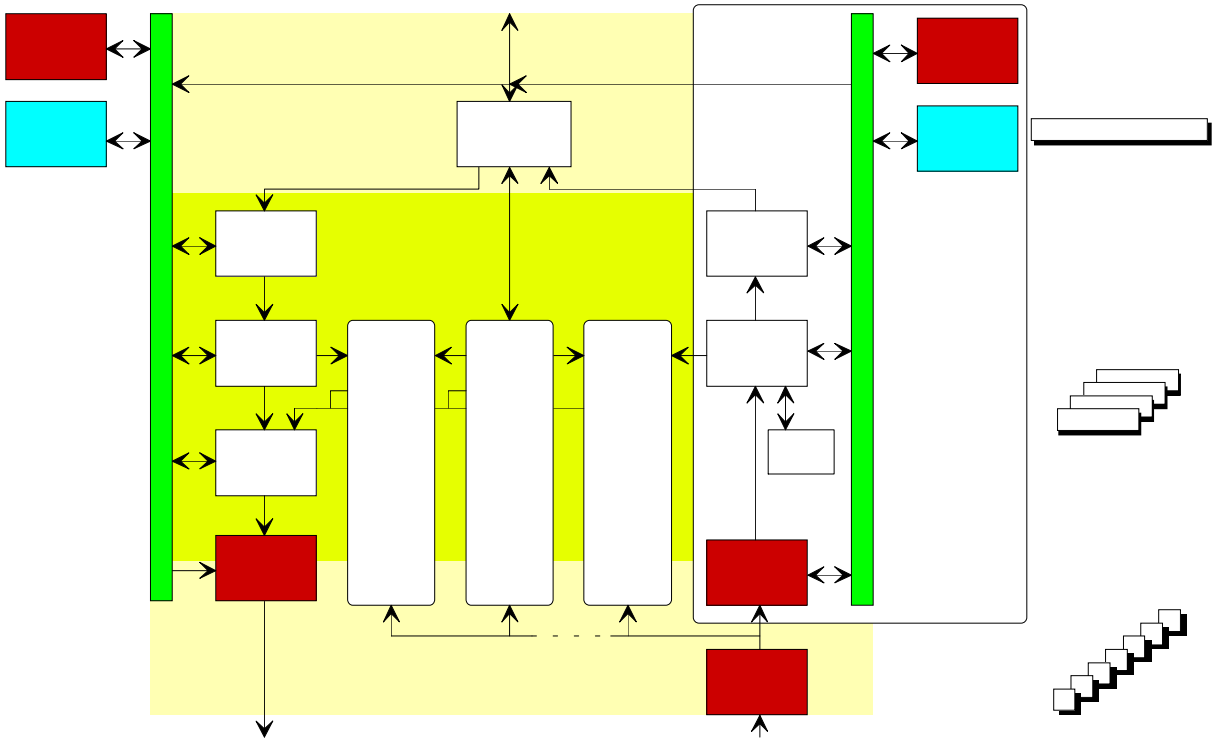

Figure 6: Implementation architecture

On send side, TSDUs are segmented by the segmentation processor and delivered to the data send processor. The send control processor receives P-frames from the data send processor and from the processor modules, determines the next P-frame to be sent, and delivers corre-

sponding P-frame informations to the segmentation hardware, which performs cell segmentation for a cell-based network. The use of shared memories avoids copying of user data between different processors. Memory management tasks are distributed among the processors of the send pipeline. Special data structures are required, which are designed in order to reduce copying of data to a minimum. They permit efficient memory management and access by several processors without any inconsistencies and conflicts. The same concept is used on the receive pipeline.
On receive side, a demultiplexing hardware distributes incoming cells to the reassembly hardware units of the processor modules. The processes of the user data receive pipeline and the transfer control functions are mapped to processor modules. The reassembly hardware units of the processor modules reassemble incoming cells to P-frames and signal completely received P-data-frames and errors (e.g., bit errors, cell loss, and missequenced cells). The received P-data-frames are delivered via the data receive processor to the reassembly processor, which generates TSDUs. The TSDUs are delivered via the transport service interface to the transport service user.

5. Performance Evaluation

For the performance analysis a protocol configuration with window-based flow control, rate control, and selective retransmission has been selected. Congestion control is switched off, because the lack of congestion control does not influence the achievable throughput. Figure 7 shows the communication among the protocol processes in the case of regular, and error-free data flow. The processes exchange window and credit values, and information about P-data-frames (packets) received or to be sent. Most of the processing steps of a process are triggered by received information from another protocol process. These messages are depicted by an arrow. Time-out messages from timer processes, are indicated by a circle near the beginning of an arrow.

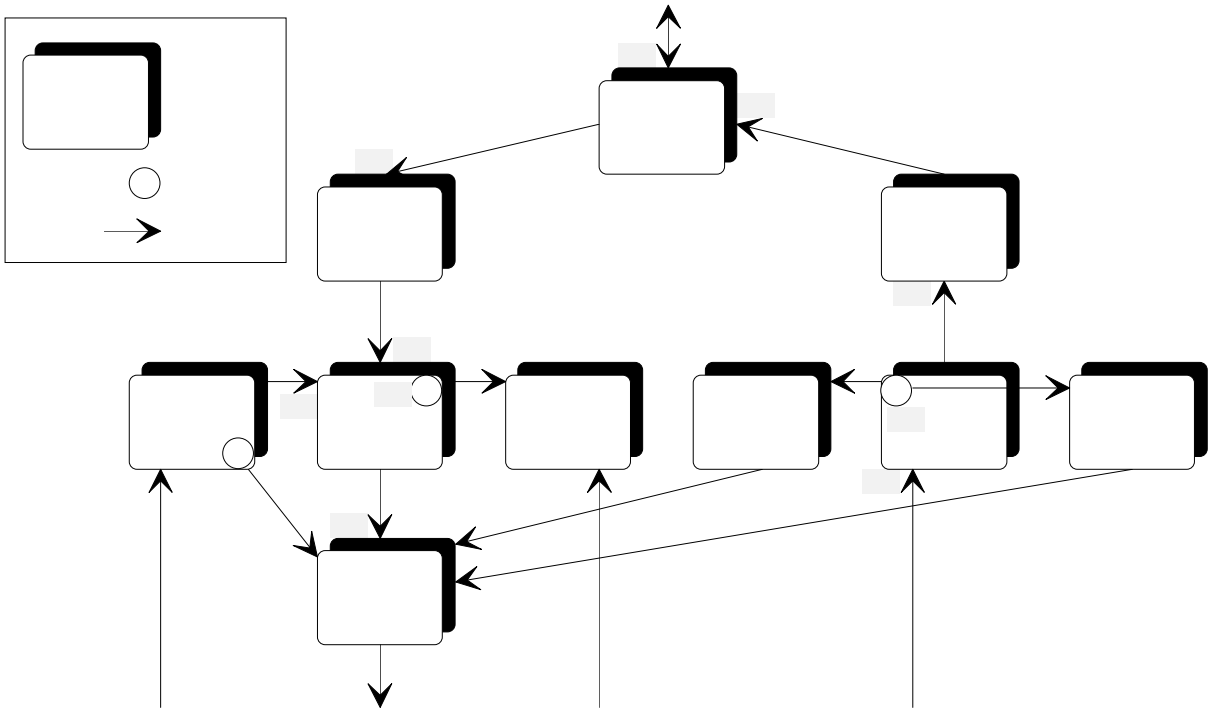

Figure 7: Process net and processing times on transputers

The time periods needed by the processing steps are presented close to the arrowheads of the processing steps triggered by received messages and close to the time-out symbol of the proc-

essing steps triggered by time-outs. E.g., the data send process sends messages to the retransmission process periodically. The processing time for the data send process to react on the time-out by sending a message to the retransmission process and to restart the timer is 89 µs. The retransmission process needs 82 µs for processing the incoming message.

5.1. Send Throughput

Because P-control-frames are sent and received periodically and, therefore, usually less frequently than P-data-frames, processing of user data to be sent and received will be the critical path. The send path consists of the processes

transport interface → segmentation → data send → send control,

the receive path consists of the processes

data receive → reassembly → transport interface.

The processing times of the send and receive pipeline are highlighted in Figure 7. Because both paths may be realized by pipelines, the throughput values depend on the processes with the largest processing times of the pipeline. The so-called *bottleneck processes* are the data send process on the send pipeline with more than 126 µs, and the data receive process on the receive pipeline with more than 159 µs.

To calculate the average processing time per P-data-frame of the data send process, we have to take into account that the data send process is triggered by three events during the regular data flow:

- The data send process gets P-data-frame information from the segmentation process, formats the P-data-frame header, and delivers the P-data-frame to the send control process, if the credit is large enough. The time needed for this procedure is called t_{seg}.
- The flow control send process controls the available credit of the data send process and sends new credit values to the data send process periodically. Processing an incoming credit value from the flow control send process takes the time t_{fcs}. Assuming that this message arrives every *w* P-data-frames, the data send process requires the time $\frac{t_{fcs}}{w}$ per P-data-frame to process the incoming new credit values.
- The data send process has to send informations about sent P-data-frames (t_{ret}) to the retransmission process, which monitors missing acknowledgments. The time period between two messages must be lower than the minimum round trip time (*rtt*), which is the minimum time between sending a P-data-frame and receiving its acknowledgment. With *p* as the packet rate [packets/s], the processing time per P-data-frame for sending information about sent P-frames is $\frac{t_{ret}}{p \cdot rtt}$.

The maximum achievable packet rate can now be calculated by the following formula:

$$p_S = \frac{1}{t_{seg} + \frac{t_{fcs}}{w} + \frac{t_{ret}}{p \cdot rtt}} \Rightarrow p = \frac{1 - \frac{t_{ret}}{rtt}}{t_{seg} + \frac{t_{fcs}}{w}}$$

Figure 8 shows the throughput in packets (P-data-frames) per second dependent on the parameters w and rtt (t_{seg} = 126 µs, t_{fcs} = 141 µs, t_{ret} = 89 µs). Increasing the values of rtt and w results in a higher performance. The achievable throughput is higher than 6000 packets/s for rtt > 5 ms and w > 5, but is limited to 8000 packets/s by the time of 126 µs of the data send process to format a packet.

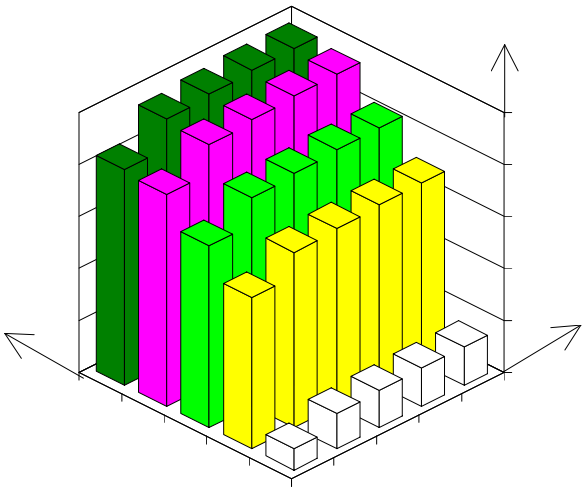

Figure 8: Throughput of the send path

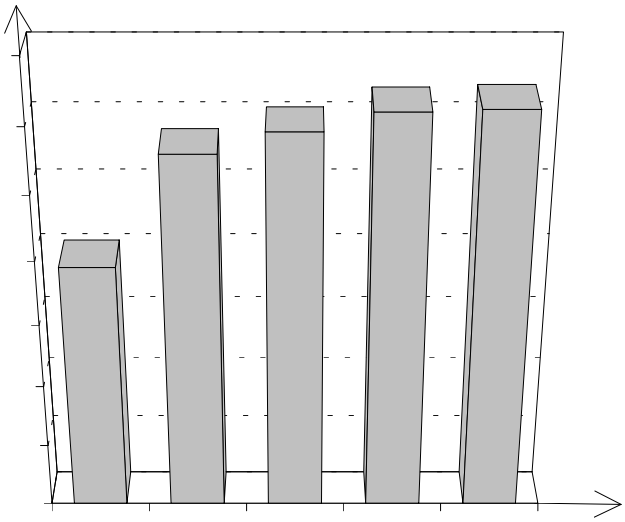

Figure 9: Throughput of the receive path

5.2. Receive Throughput

Assuming a regular data flow, two actions of the data receive process have to be considered to get the average processing time per P-data-frame of the data receive process:

- The data receive process gets P-data-frames from the network, analyzes the lifetime, checks on duplicates etc., and delivers the P-data-frames to the reassembly process. This procedure takes the time t_{net}.
- The data receive process has to inform the acknowledgment process periodically about P-data-frames correctly received (t_{ack}). Also, the flow control receive process, expects buffer state information within certain periods of time. If the generation and the distribution of these messages is performed every q P-data-frames, it costs $\frac{t_{ack}}{q}$ per P-data-frame.

The maximum throughput of the receive pipeline is calculated by the following formula:

$$p_r = \frac{1}{t_{net} + \frac{t_{ack}}{q}}$$

Figure 9 shows the throughput dependent on the parameter q (t_{net} = 159 µs, t_{ack} = 97 µs). The throughput grows with an increasing value of q, but is limited by t_{net} to 6300 packet/s and is higher than 5600 packets/s for q > 5.

5.3. Speed-up and Discussion

The speed-up is a measure for the performance gain of a parallel implementation compared with a sequential one. The speed-up values of the PATROCLOS implementation are between 2 and 2.5 for sending and between 2.5 and 3.5 for receiving. The values vary dependent on the parameters w, rtt, and q. The speed-up is higher for an increasing ratio of control functions to data processing functions [25].
Compared with other implementations, the PATROCLOS implementation achieves very high performance. The XTP implementation [4], which, however, uses an other memory management scheme, accomplishes a maximum throughput of less than 1330 packets/s. A TCP/IP implementation [5] running on a transputer architecture (parallel protocol engine, PPE) with the same transputers and memories as used for PATROCLOS achieves up to 2800 packets/s.

6. Conclusions

The general goal of PATROCLOS, a parallel transport subsystem appropriate for high performance communication subsystems, was an integrated design of protocol and implementation architecture issues. The presented design is highly influenced by guidelines for parallel protocol design. The key issue is the decomposition of a protocol entity into concurrently executable FSMs. Therefore, the protocol consists of multiple modular FSMs operating highly independent from each other. The design comprises separate FSMs for each control function and for user data processing. With respect to concurrent processing, synchronisation between FSMs of a single protocol entity is reduced by associating each FSM with an individual protocol to exchange messages with FSMs residing in peer entities.
A prototype version of PATROCLOS has been implemented on a multiprocessor architecture using transputers as general purpose processors and dedicated hardware for time-critical protocol functions. Performance evaluations show significant improvements compared to similar implementations of other protocols. The results show that multiprocessor systems are applicable candidates for future gigabit rate networks, if protocols are well designed for efficient implementation architectures.
An advanced implementation architecture to achieve higher performance has been designed based on VLSI components. The architecture, which has been developed for parallel protocols like XTP or PATROCLOS, has been described in VHDL. Simulation and implementation are currently being performed [26].

References

[1] Zitterbart, M.
Parallelism in Communication Subsystems
in: Tantawy, A. (ed.): High Performance Communications, Kluwer Academic Publisher, 1993

[2] Zitterbart, M.
High Speed Transport Components
IEEE Network Magazine, Vol. 5, No. 1, January 1991, pp. 54-63

[3] 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

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

[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] Sterbenz, J.P.G.; Parulkar, G.M.
AXON Host-Network Interface Architecture for Gigabit Communications
in: Johnson, M. J. (ed.): Protocols for High-Speed Networks, II, North-Holland, 1991, pp. 211-236

[7] Diot, C. ; Dang, M.
A High Performance Implementation of OSI Protocol Class 4; Evaluation and Perspectives
15th Annual Conference on Local Computer Networks, September 30 - October 2, 1990, Minneapolis, U.S.A., pp. 223-230

[8] Ulrich, R.
Untersuchungen an einem OSI-Kommunikationswerk auf Transputer-Basis
Dissertation (in German), Technical Report, Vol. 26, No. 11, University of Erlangen, Germany, September 1993

[9] Jain, N.; Schwartz, M.; Bashkow, T.R.
Transport Protocol Processing at Gbps Rates
Proceedings of the ACM SIGCOMM ´90, September 24-27, 1990, Philadelphia, USA, pp. 188-199

[10] La Porta, T.F.; Schwartz, M.
A High-Speed Protocol Parallel Implementation: Design and Analysis
in: Danthine, A.; Spaniol, O. (eds.): High Performance Networking, IV, IFIP, North-Holland, 1993, pp. 135-150

[11] de Prycker, M.
Asynchronous Transfer Mode: Solution For Broadband ISDN
Ellis Horwood, New York, London, Toronto, Sydney, Tokyo, Singapore, 1991

[12] Kessler, G.C.; Train, D.A.
Metropolitan Area Networks - Concepts, Standards and Services
McGraw-Hill, 1992

[13] 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

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

[15] Protocol Engines Inc.
XTP Protocol Definition
Revision 3.4, July 17, 1989

[16] Clark, D.; Tennenhouse, D.
Architectural Considerations for a New Generation of Protocols
ACM SIGCOMM ´90, September 24-27, 1990, Philadelphia, USA, pp. 200-208

[17] Krumm, H.
Funktionale Analyse von Kommunikationsprotokollen
Informatik-Fachberichte 247 (in German), Springer-Verlag, 1990

[18] Zitterbart, M.; Stiller, B.; Tantawy, A.
A Model for Flexible High-Performance Communication Subsystems
Journal of Selected Areas in Communication, May 1993, pp. 507-518

[19] Schmidt, C.
Entwurf eines Transportsystems auf der Basis paralleler Automaten
Diploma thesis (in German), University of Karlsruhe, Germany, July 1992

[20] Ludwig, J.
Integration von Protokollmechanismen in ein paralleles Transportprotokoll
Diploma thesis (in German), University of Karlsruhe, Germany, June 1993

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

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

[23] 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

[24] INMOS Ltd.
ANSI C Toolset Language Refernce
INMOS Ltd., Bristol, U.K., 1990

[25] Braun, T.
Ein paralleles Transportsubsystem für zellenbasierte Hochgeschwindigkeitsnetze
Dissertation (in German), University of Karlsruhe, Germany, VDI-Verlag, Düsseldorf, 1993

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