Implementation of a Parallel Transport Subsystem on a Multiprocessor Architecture

Torsten Braun, Claudia Schmidt

Institute of Telematics, University of Karlsruhe, W-7500 Karlsruhe, Germany

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 needed for the requirements of 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 specialised hardware for very time critical functions. The paper presents transport system design guidelines based on experiences gained with parallel implementations of transport and network layer protocols on transputer networks, an implementation architecture for PATROCLOS based on transputer networks and results of a performance evaluation, which indicate promising throughput values.

1. Introduction

During the last years, the characteristics of communication networks changed dramatically. The bandwidths of the networks are growing into the gigabit range and are providing an increasing variety of services. 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 and deliver the performance of the networks to the applications. The concept of parallelism may significantly increase the performance of *transport subsystems*, which cover only a subset of a complete transport system. Transport subsystem entities use *protocols* for communication. Parallelism should be based on a relatively fine granularity together with low overhead (e.g., communication overhead among parallel building blocks) to support efficient mappings of transport subsystem protocols on multiprocessor architectures. Several parallel protocol implementations have already shown possible performance advantages [1, 2, 3, 4, etc.].

Because parallelising standard protocols (e.g., OSI or TCP/IP) suffers from their inherent sequential structure, PATROCLOS realises 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 and DQDB influence the protocol design for the transport system [5]. A number of protocol mechanisms (e.g., connection management, error control, flow control) need to be changed in order to take advantage of the provided network bandwidth [6]. Moreover, since a large amount of data may be travelling 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 performance communications. Mechanisms based on exchanges after dedicated requests (e.g., XTP [7]) or periodic exchanges (e.g., SNR [2]) 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 layer interface of cell-based networks and the transport layer interface in a single, modular designed protocol component.

2. Design Guidelines for Parallel Transport Subsystems

Considering processing speeds and memory bandwidth as major bottlenecks for high performance 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 [7]. Parallelism has been considered in the design process and has resulted in the specification of multiple concurrent extended finite state machines (FSMs) [8]. 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 [3]. This XTP implementation is able to process send and receive part in parallel without significant interactions. 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 [5]:

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

3. PATROCLOS

The presented design issues have influenced the architecture of PATROCLOS, which combines a modular design and the avoidance of layering to achieve a high degree of parallelism. The transport subsystem has the functionality of the OSI layers 2b-4 and is based on a specification as parallel finite state machines. This section describes the decomposition of the protocol functions into autonomous units to support the implementation on a parallel architecture and introduces the specifications of the FSMs.

PATROCLOS is mainly designed around a set of independent protocol functions [5]. There is a high degree of independence among functions involved in send and receive processing. Furthermore, the protocol functions are classified as transfer control, data access and data control functions. ***Data access functions*** process user data units and can be divided into the subclasses *data manipulation functions* and *data analysis functions*. ***Data control functions*** decide the actions to be performed on data units. Data control functions can also be divided into two subclasses called *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.

3.1. Parallel FSM Specification

The components of the PATROCLOS architecture are realised by parallel FSMs. FSMs belonging to the same PATROCLOS entity exchange messages for synchronisation and communication purposes. They form a system of extended finite state machines (FSM system) with the functionality of the transport subsystem.

3.2.1. FSM System Definition

A FSM is defined by the quintuple [9]:

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

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., a relation 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. 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 the messages to channels. The FSM system has two interfaces, an input interface allows an external influence on the system, the output interface indicates the reactions of the system. A FSM system is defined by [9]:

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

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 characterised 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 timeout 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 system 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. For formal FSM description we used charts and PROMELA [10]. The specification has been simulated and validated with SPIN (simple PROMELA interpreter) [10].

3.2.2. The PATROCLOS FSM System

PATROCLOS consists of two types of FSMs:

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

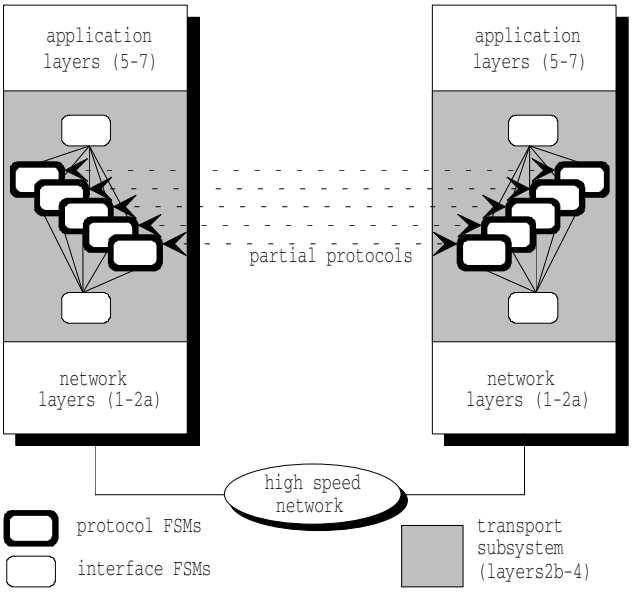

Figure 1 : Architecture of PATROCLOS

The system interface of PATROCLOS is subdivided in two interface FSMs. These interface FSMs support interactions of PATROCLOS with the transport service user or with the underlying network unit. They are only involved in local communications inside one entity, and they do not have to communicate with FSMs at the peer entity. The transport interface FSM 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 secure connection termination. The network interface FSM controls and schedules transmission of P-frames to the underlying network unit.

The protocol FSMs are designed to decouple the connection state information exchange from user data exchange effectively. Acknowledgements 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 and separate packets (P-frames). Multiplexing of P-frames by different FSMs is avoided to support parallelism. The FSM protocols have their individual error recovery mechanisms and timers. The decomposition into independent FSMs permits parallel execution and it 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. Configuration issues of transport systems are covered in [11].

3.2.3. Communication among Protocol FSMs

The independent protocol functions identified in the previous section are realised by autonomous protocols, which build together the PATROCLOS protocol. Each partial protocol is performed by communicating peer FSMs, which use data units 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. Several FSMs are discussed in the following:

- *Connection management*
  A single connection management FSM manages the complete duplex connection. It supports establishment, monitoring, maintenance, and correct termination of connections. Therefore, this FSM has to activate and initialise the other FSMs of the same entity for each new connection and has also to deactivate these FSMs after connection termination. To provide a correct 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. Handshake-based or implicit establishment mechanisms are provided according to the requested service.
- *User data transfer*
  User data transfer FSMs send and receive user data. The main tasks of the two FSMs are formatting and analysing P-data-frames. Therefore, the user data send FSM uses the header fields of the P-data-frames to describe the user data, while the user data receive FSM operates on these informations to detect duplicates and errors or it checks the lifetime of the packet.
- *Acknowledgement and retransmissions*
  The acknowledgement/retransmission functions can be divided into two FSMs. The FSM on the send side (retransmission FSM) receives acknowledgements from the peer's FSM on the receive side (acknowledgement FSM). The retransmission FSM may also request acknowledgements, or send the state of acknowledged data to the peer's acknowledgement FSM. Retransmission strategy and management of unacknowledged data are implemented in this FSM.
- *Flow and rate control*
  For flow and rate control, there exist two FSMs for each connection, i.e., one FSM for each transfer direction. The flow control send FSM controls data flow by the interpretation of the peer entity's feedback signals, e.g., window and buffer sizes, and rate values.

For rate control, both peer FSMs negotiate appropriate rate parameters.

- *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 can be decreased [12, 13]. The FSM may also react to the feedback signals of the network nodes.
- *Routing and relaying*
  The FSM performs functions required in intermediate systems such as relaying, routing, route management, and monitoring.
- *Delay jitter control*
  The jitter control FSMs support the timing relationships between sent and received data. Several applications require well defined data packet inter-arrival times. These FSMs send and receive time stamps for transferred user data. The receive FSM controls the orderly delivery of user data to the transport system user considering the received time stamps.

3.2.4. Co-operation among FSMs of one Entity

The system is designed to minimise interactions among FSMs of the same entity, i.e., all FSMs are designed as autonomous as possible [14]. 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. The co-operation among the PATROCLOS FSMs in the cases of flow control and data transfer with retransmission and acknowledgement is described in the following.

Data transfer, retransmission, and acknowledgement

Data transfer can be divided into a sending path and a receiving path. The sending path includes most of the data manipulation functions, the receiving path consists of most of the data analysis functions. Because data transfer often performs the critical path in communication protocols, retransmission and acknowledgement are handled by special control functions, i.e., the user 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. Retransmissions depend on incoming acknowledgements from the peer's acknowledgement FSM.

Because processing the user data transfer functions usually performs the critical path in communication protocols, communication between the participating FSMs 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. E.g., the user data send FSM gives the control over the sent packets to the retransmission FSM periodically. To avoid delays in error cases, the information is sent as quick as possible.

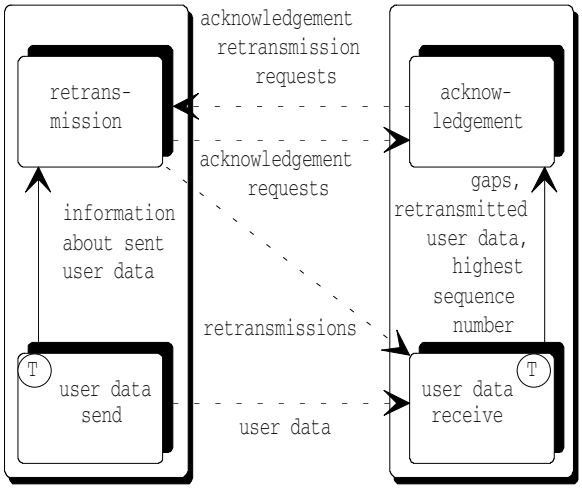

Figure 2: Co-operation among FSMs for data transfer and retransmission

Flow control

Flow control in PATROCLOS is a combination of window-based and rate-based mechanisms. Two flow control FSMs exist in each entity, one FSM for each data flow direction (cf. Figure 3). These FSMs negotiate flow control parameters with the remote peer and use these parameters to control the data flow. On the receiving 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 new flow control parameters and sends them periodically to the remote peer's flow control send FSM. This one controls the FSMs of the user data sending functions by the negotiated flow control parameters. The user data send FSM is responsible for the window-based part of the flow control and, therefore, receives the new window bounds from the flow control send FSM. If required, the user data send FSM can request new credit values. 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. Simultaneously, the congestion control send FSM informs it about the actual traffic patterns. To minimise 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 exception cases, e.g., the user data send FSM requests a new credit, if it is blocked on the sliding window.

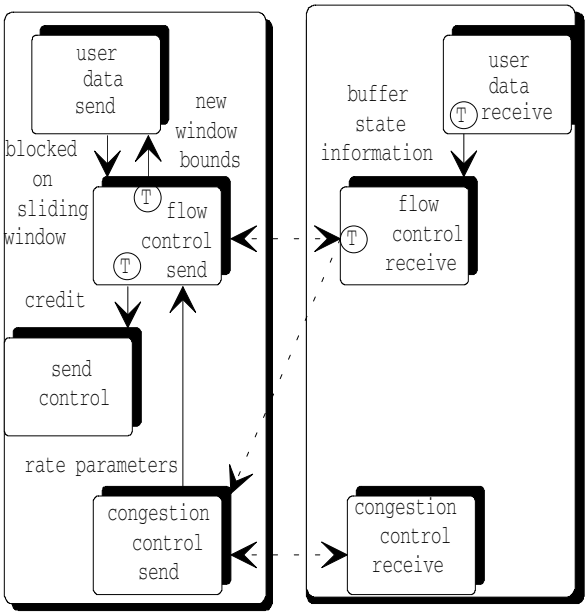

Figure 3: Co-operation of FSMs involved in flow control

4. Prototype Implementation

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 [1, 3, 4]. This section describes the mapping of the protocol specification based on a set of parallel FSMs to appropriate implementation building blocks. Based on these implementation concepts, a hybrid multiprocessor architecture has been developed. Finally, the achievable performance of the parallel implementation is analysed.

4.1. Implementation of FSMs by Processes

The FSMs have been implemented as processes (main 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. Processes on the same processor are interconnected by software channels, processes on different processors are interconnected by hardware channels.

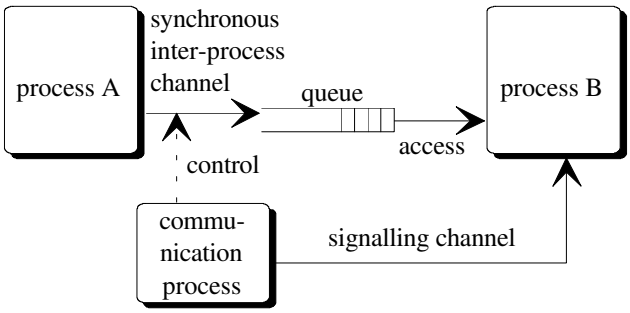

Figure 4: Asynchronous communication between processes

Each main process starts a subprocess, the so-called communication process, which supports asynchronous inter-process communication across synchronous channels (cf. Figure 4) and avoids deadlocks. A communication process receives incoming messages from another main process over a synchronous inter-process channel and stores the received messages in a queue. The communication process co-operates with the according main process by this queue and wakes up the main process whenever a message arrives by sending a signal over the signalling channel. Then, the main process performs computations dependent on the contents of the received message. Transputers provide special capabilities to perform inter-processor communication concurrently with computations of any process. A parallel C language [15] with additional elements for message passing and process management has been used for implementation.

```
process data_send () {
  while (TRUE) {
    read_message_from_queue();
    switch (message_type) {
      case FRAME_FROM_SEGMENTATION:
        format_frame ();
        deliver_frame_to_send_control ();
        break;
      ...
      case OPEN_MSG_FROM_CONN_MANAGER:
        initialise_new_context ();
        break;
  }}}
```

Figure 5: Process implementation

The program in Figure 5 shows the structure of the data send process as example for the implementation of protocol processes. Different processing 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.

4.2. Multiprocessor Based Implementation Architecture

The multiprocessor-based implementation architecture uses transputers as universal processors. Each process derived from the PATROCLOS protocol architecture can be mapped to a separate processor. A demultiplexing unit to demultiplex incoming P-frames to the appropriate processor, access control units to control the access of several processors to the shared memory buses, and protocol functions on cell level (SAR processing according to AAL4) have been implemented by dedicated hardware units to support the high network bandwidth. The combination of universal processors and specialised processors results in a hybrid multiprocessor architecture.
The implementation architecture will be connected via a DMA unit to a host system. On the send side, TSDUs are segmented by the segmentation processor and delivered to the data send processor. The use of shared memories avoids copying of data between different processors. The send control processor receives P-frames from the data send processor and from the transfer control processors, determines the next P-frame to be sent, and delivers corresponding P-frame informations to the segmentation hardware, which performs cell segmentation for a cell-based network.
On the receive side, the demultiplexing hardware distributes incoming cells to the reassembly hardware units of the data manipulation processor and the transfer control processor modules. The reassembly hardware units 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 receive control processor to the reassembly processor, which generates TSDUs. The TSDUs are delivered via the transport service interface to the transport service user.
Memory management tasks are distributed among the processors of the send and receive pipeline. Special data structures are required, which have been 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.

4.3. Performance Evaluation

Figure 6 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 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 in Figure 5 by an arrow. Time-out messages from timer processes, are indicated by a circle near the beginning of an arrow.
The time periods needed for 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 processing steps triggered by timeouts. 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 101 µs, the retransmission process needs 86 µs for processing the incoming message.
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 6. Because both paths may be processed by pipelines, the throughput values depend on the processes with the largest processing time of the pipeline. These so-called *bottleneck processes* are the data send process on the send pipeline with more than 159 µs, and the data receive process on the receive pipeline with more than 179 µs.
To compute the average processing time per frame of the data send process, we have to take into account that the data send process is being triggered by three events during the regular data flow periods:

- 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. This 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 needs 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 acknowledgements. The time period between two messages must be lower than the minimum round trip time (*rtt*), i.e., the minimum time between sending a P-data-frame and receiving its acknowledgement. With *p* as the packet rate [packets/s], the processing time per P-data-frame for sending information about sent P-frames is $\frac{t_{ret}}{p*rtt}$.

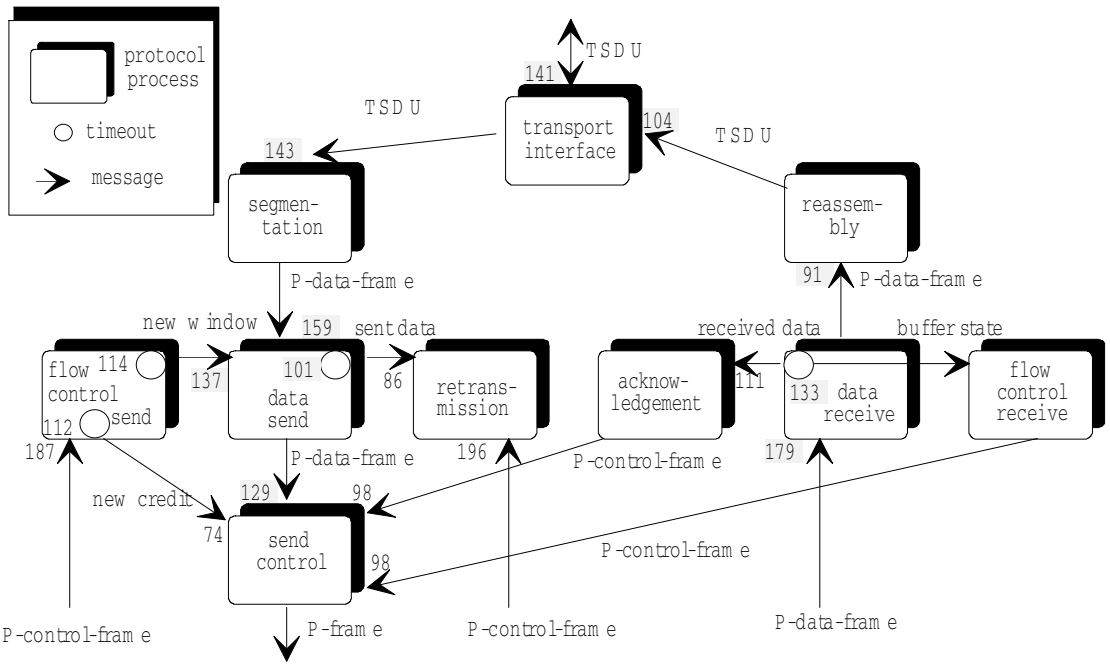

Figure 6: Performance evaluation of the PATROCLOS processes on transputers

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

$$p = \frac{1}{t + \frac{t_{fcs}}{w} + \frac{t_{ret}}{p*rtt}} \Rightarrow p = \frac{1 - \frac{t_{ret}}{rtt}}{t + \frac{t_{fcs}}{w}}$$

Figure 7 shows the throughput in packets (P-data-frames) per second dependent on the parameters w and rtt (t_{seg} = 159 µs, t_{fcs} = 137 µs, t_{ret} = 101 µs). Increasing the values of rtt and w results in a higher performance. The achievable throughput is higher than 5000 packets/s for rtt > 5 ms and w > 5, but is limited to 6300 packets/s by the time of 159 µs of the data send process to format a packet.

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

- The data receive process gets P-data-frames from the network, analyses 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 acknowledgement 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. These messages should be exchanged at least once per rtt. If the generation and distribution of these messages is performed n times per rtt, it costs $\frac{n * t_{ack}}{p * rtt}$ per P-data-frame.

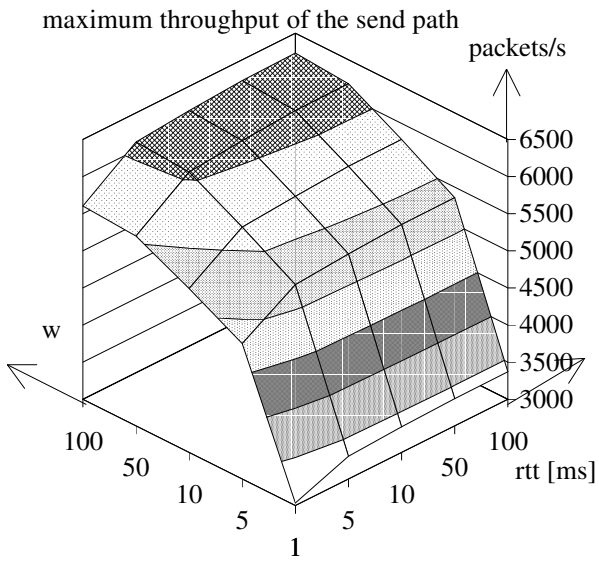

Figure 7: Throughput of the send path

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

$$p = \frac{1}{t_{net} + \frac{n * t_{ack}}{p * rtt}} \Rightarrow p = \frac{1 - \frac{n * t_{ack}}{rtt}}{t_{net}}$$

Figure 8 shows the throughput dependent on the parameters n and rtt (t_{net} = 179 µs, t_{ack} = 133 µs). The throughput grows with an increasing value of the ratio $\frac{rtt}{n}$, but is limited by t_{net} to 5600 packet/s and is higher than 5000 packets/s for rtt > 5 ms.

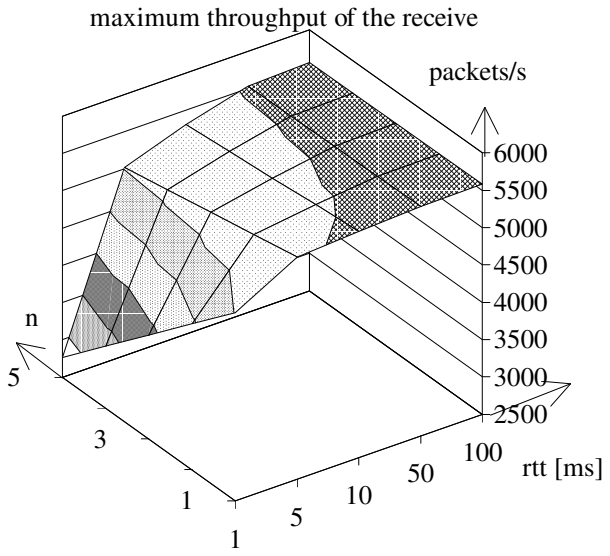

Figure 8: Throughput of the receive path

Compared with other implementations, the PATROCLOS implementation achieves very high performance. The XTP implementation [3], which, however, uses an other memory management scheme, accomplishes a maximum throughput of less than 1330 packets/s. A TCP/IP implementation [4] running on a faster transputer architecture (parallel protocol engine, PPE) than the one used for PATROCLOS and XTP achieves up to 2800 packets/s. The throughput values of PATROCLOS will at least be 15 % higher on the PPE.

5. Conclusions

In this paper, PATROCLOS, a parallel transport subsystem appropriate for high performance communication subsystems, has been presented. The general goal was an integrated design of protocol and implementation architecture issues. Multiprocessor systems may be applicable candidates for future gigabit rate networks and, thus, a protocol design appropriate for efficient implementation architectures is required.

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 almost independently from each other. The design comprises separate FSMs for each control function and for user data processing. With respect to concurrent processing, synchronisation among FSMs of a single protocol entity is reduced by associating each FSM with an individual protocol for message exchange between FSMs residing in different 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.

References

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

[2] 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

[3] 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

[4] Rütsche, E.; Kaiserswerth, M.
TCP/IP on the Parallel Protocol Engine
4th IFIP Conference on High Performance Networking, HPN92, December 14-18, 1992, Liège, Belgium

[5] Braun, T.; Zitterbart, M.
Parallel Transport System Design
4th IFIP Conference on High Performance Networking, HPN92, December 14-18, 1992, Liège, Belgium

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

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

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

[9] Krumm, H.
Functional Analysis of Communication Protocols
Informatik-Fachberichte 247 (in German), Springer, 1990

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

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

[12] 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

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

[14] Schmidt, C.
Design of a Transport System Based on Parallel Finite State Machines
Diploma thesis (in German), University of Karlsruhe, July 1992

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