Performance of a Parallel Transport Subsystem Implementation

Torsten Braun
Institute of Telematics, University of Karlsruhe, P.O. Box 6980, Zirkel 2, 76128 Karlsruhe, Germany
Phone: +49 721 608-3982, Fax: +49 721 388097, E-mail: braun@telematik.informatik.uni-karlsruhe.de

Introduction

The concept of parallelism may significantly increase the performance of *transport subsystems* [1, 2, 3, etc.]. A transport subsystem covers a subset of a complete transport system, which combines the functions of the four lower layers according to the OSI Reference Model. Because parallelising standard protocols like OSI TP4/CLNP or TCP/IP suffers from their inherent sequential structure, the transport subsystem PATROCLOS (parallel transport subsystem for cell based high speed networks) realises a parallel protocol architecture with fine granularity adequate for parallel implementations on hybrid multiprocessor platforms. PATROCLOS overcomes 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 service interface (functionality of the OSI layers 2b-4) in a single protocol component. The approach followed in the PATROCLOS design combines a very fine granularity with a protocol function-oriented decomposition into basic modular building blocks. In addition, several characteristics of emerging cell based networks such as ATM and DQDB influence the PATROCLOS protocol design [4]. PATROCLOS provides a number of protocol mechanisms to support different application and network requirements. Especially, mechanisms appropriate for high-speed network environments with large a bandwidth-delay-product have been integrated.

Parallel Protocol Architecture of PATROCLOS

PATROCLOS combines a modular design and the avoidance of layering to achieve a high degree of parallelism. Redundant placement of protocol functions in several layers can be avoided, e.g., segmentation and reassembly functions in layer 3 and 4 of the OSI Reference Model. Furthermore, the modular protocol function oriented design simplifies the configuration of the protocol. Configuration facilities will play an important role in future protocols to allow adaptation of protocols to a wide variety of application requirements and different network environments [5, 6]. Moreover, the modular protocol design supports an easy mapping on multiprocessor architectures, if the protocol components can be used as processes that can run on separate processors. Based on the analysis of XTP [7] specification, version 3.4 [8], which is based on a set of parallel finite state machines (FSMs), and the performance results of the corresponding multi-processor implementation, several design guidelines for protocol architectures appropriate for parallel processing have been derived [9]: reduced data dependences, decoupled control and user data processing, modular function-oriented decomposition, reducing event signalling, minimising common resources. These design guidelines resulted in the PATROCLOS specification. 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 messages for co-operation. All FSMs are designed as autonomous as possible to minimise interactions among other FSMs of the same entity. 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. PATROCLOS consists of two types of FSMs: interface and protocol FSMs (cf. Figure 1).

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, SC) controls and schedules transmission of *all* protocol data units (PDUs, P-frames) to the underlying network unit. Note that this FSM is missing in Figure 1. The transport service interface FSM maps service primitives to signals for the protocol FSMs. The type of service offered by PATROCLOS is almost hidden by this FSM. This allows that the connection management FSM operates always in a connection-oriented mode, although offering connection-oriented, connectionless and transaction services according to the OSI 95 service definition [10]. Furthermore, the transport interface FSM exchanges TSDUs with the transport service user in both directions and is involved in performing secure connection termination.

The key feature of ***protocol FSMs*** is that they communicate directly by separate protocols with the corresponding FSMs at the peer entity. They 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 travelling between communicating end systems, state exchange (e.g., acknowledgements and connection state information exchange) after dedicated requests and errors (e.g., XTP [7]) or periodic exchanges (e.g., SNR [1]) seem to be more suitable than state exchange for each data packet. All communication relations among FSMs in peer entities have own protocols with separate P-frames, individual error recovery mechanisms, and timers. Each partial protocol is performed by communicating peer FSMs, using protocol data units containing only the necessary information for their function. Multiplexing of P-frames by different FSMs is avoided to support parallelism. Together, the autonomous partial protocols build the 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 [5]. Currently, there are the following protocol FSMs:

The ***connection management*** FSM is responsible for establishment, management, and correct release of a full duplex connection. Therefore, it 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. The peer FSMs exchange P-frames for connection establishment. They initialise the connection and forward quality-of-service requirements to other FSMs. The connection management FSMs also guarantee the unique use of connection identifiers and perform signalling tasks, e.g. in B-ISDN networks. The transport service FSM selects handshake-

based or implicit establishment procedures dependent on the required service. Additionally, explicit/timer-based and graceful/normal connection release may be selected. 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. Because the connection state information (context) is distributed among the FSMs the graceful release requires co-operation with other FSMs. Connection management is not further divided, because establishment, maintenance, and termination of connections occur sequentially.

User data transfer can be divided into a send path and a receive path. User data transfer FSMs send and receive user data. Therefore, the main tasks of the two FSMs are formatting and analysing P-frames with user data. Additional functions are segmentation and reassembly of TSDUs and P-frames. The user data send FSM uses the header fields of the P-frames to describe the user data (e.g., sequence numbers), while the user data receive FSM operates on these informations to detect duplicates and errors or it checks the lifetime of the packet. Because segmentation/reassembly is separated from formatting and analysing functions, each data transfer FSM consists of two subordinate FSMs. In intermediate systems there is another FSM in addition to the data transfer FSMs performing functions such as ***routeing & relaying***, route management, and monitoring.

Because data transfer often performs the critical path in communication protocols, ***retransmission and acknowledgement*** are handled by two special FSMs. The FSM on the send side (retransmission FSM) requests and receives acknowledgements from the peer's FSM on the receive side (acknowledgement FSM). The retransmission FSM analyses the acknowledgement packets, initiates retransmissions depending on the acknowledgement information and the selected strategy (selective or Go-back-N) and releases the buffers of acknowledged data packets. Retransmissions are sent to the peer's user data receive FSM.

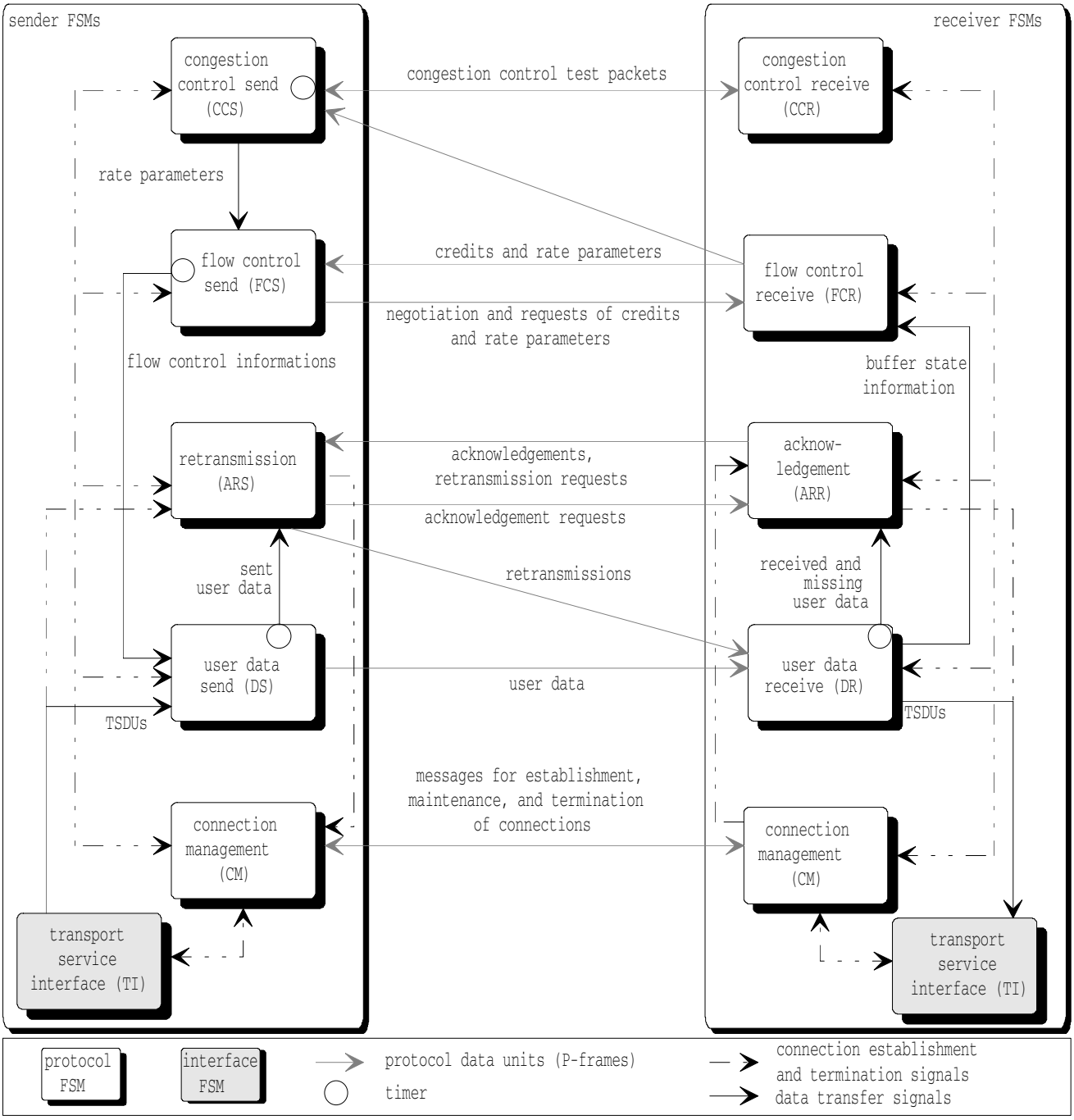

Figure 1: Communication among FSMs

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. The acknowledgement FSM sends negative selective and positive cumulative acknowledgements to the peer's retransmission FSM. The user data receive FSM signals informations about received data to the acknowledgement FSM periodically, which requests retransmissions based on the received results. 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.

Flow control in PATROCLOS is a combination of window-based and rate-based mechanisms. Additionally, a stop-and-go mechanism is provided. Two flow control FSMs exist in each entity, one FSM for each data flow direction. These FSMs negotiate flow control parameters with the remote peer and use these parameters to control the data flow. 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. The flow control receive FSM transmits rate parameters or credit values according to the current traffic and available buffer space. The flow control send FSM signals periodically the maximum sequence number to be sent (limited by window-based flow control) to the data send FSM. Furthermore, this FSM starts a rate timer for rate-based flow control, calculates the maximum sequence number to be sent for the next time period, and signals this value to the network interface 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 packets. The FSMs exchange test P-frames to identify congestions. Based on test packet delays and inter-arrival times, congestion situations 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 [11] and the packet pair algorithm (2P) [12]. The FSM may also react to the feedback signals of intermediate systems.

Several multi media applications require well defined data packet inter-arrival times. The ***jitter control*** FSMs support the timing relation between sent and received data by exchanging 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.

Implementation

PATROCLOS is appropriate but not limited to be implemented on a hybrid multiprocessor architecture. The implementation architecture will be connected via a DMA unit to a host system. The FSMs of Figure 1 have been implemented as processes, one process for each FSM. 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. Protocol and timer processes co-operate by message passing, too. Processes on the same processor are interconnected by software channels, processes on different processors are interconnected by hardware channels. Inter-processor communication is performed in parallel to other computations on transputers. A parallel C language with additional elements for message passing and process management has been used for implementation. Figure 2 shows the data send process as an example for the process implementation structure. Different processing steps have to be performed dependent on the incoming message.

```
process data_send (){
   while (TRUE){
      read_message_from_queue();
      switch (message_type){
         case FRAME_FROM_SEGMENTATION:
            format_frame ();
            send_frame ();
            break;
         case TIME_OUT:
            send_info_to_retransmission ();
            break;
         ...
}}}
```

Figure 2: Process Implementation

The multiprocessor-based implementation architecture uses transputers as universal processors and several specialised hardware units. The combination of universal processors and dedicated hardware results in a hybrid multiprocessor architecture. Each process derived from the interface and protocol FSMs of the PATROCLOS protocol architecture may be mapped to a separate processor. Access control units to control the access of several processors to shared memory buses, and simple protocol functions on cell level (SAR processing of AAL4) have been implemented by dedicated hardware units to support the high network bandwidth. Moreover, on the receive side, the demultiplexing hardware performs demultiplexing below the cell reassembly level and distributes incoming cells to the reassembly hardware units of the user data send processor and the processor modules of the control functions. Memory management is distributed among several processors and requires special data structures, which have been designed in order to minimise copying of data and to permit an efficient memory management and access by several processors without any inconsistencies and conflicts.

Performance

In the following analysis [13], we study the achievable performance of the receive part, which is the bottleneck of the PATROCLOS implementation. Figure 3 shows the processor configuration used for the performance measurements of the receive part. Compared to Figure 1, we added a test environment, which consists of processes for the transport user (US) and the network (NET). All processes of the send part are mapped to a single transputer. They are inactive during receive processing. In this configuration we selected rate-based and window-based flow control, selective retransmission, and inactive congestion control. Furthermore we do not perform segmentation and reassembly. However, the lack of congestion control and segmentation/reassembly does not influence the performance as we show below.

The measurement scenario is as follows: After establishing a connection, we generate user data, which are buffered by the network process. All buffered packets are delivered as fast as possible to the DR process, so that the receiver will be overloaded. Because P-control-frames are sent and received periodically and therefore usually less frequently than P-data-frames, processing of received user data will be the critical path.

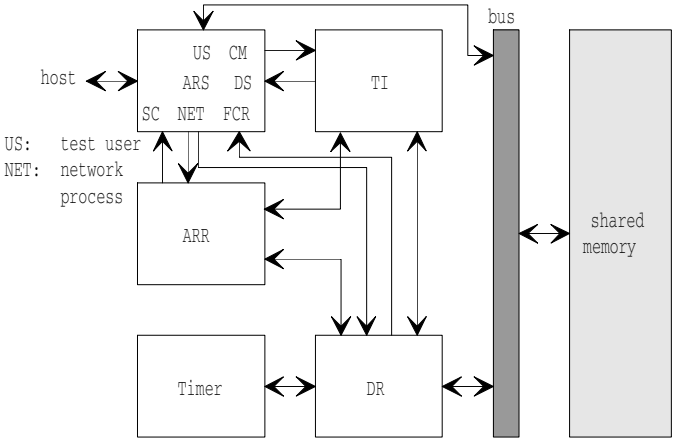

Figure 3: Transputer configuration for performance evaluation of PATROCLOS

The *bottleneck process* on the receive part is the data receive process. In addition to analyse the received data packets, the data receive process has to inform periodically the acknowledgement process about P-data-frames received correctly and the flow control receive process about buffer state information. We assume that both signals are sent q times per data packet. Decreasing the value of q results in a higher performance. The throughput is higher than 5000 packets/s for q < 1/5, higher than 6000 packets/s for q < 1/10, but is limited to 6300 packets/s by the time of the data receive process to analyse a packet (159 µs). The data receive process will also be the bottleneck, if packets have to be reassembled, because the reassembly process reassembles a packet faster than the data receive process analyses it. The throughput of the receive path have been calculated based on the processing times of the receive processes and have been verified by real measurements performed with the configuration showed in Figure 4. The measured and the calculated results differ only by 2 %. Asynchronous inter-process communications allow very exact performance predictions.

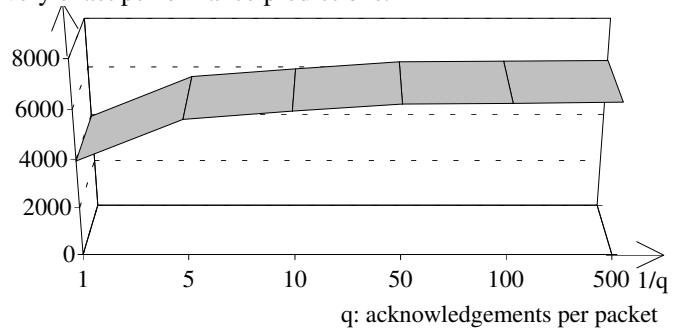

Figure 4: Throughput of receive path

Similar to the analysis of the receive path we can evaluate 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. The time of this period must be lower than the round trip time (rtt), which includes propagation delay as well as processing of user data and acknowledgements at the sender and the receiver. Therefore, rtt will be higher than 1 ms. Furthermore, the data send process receives q credit informations per packet from the flow control receive process. The throughput depends on the parameters q and rtt as shown in Figure 5. For q < 1/5 we achieve more than 6000 packets/s. The speed-up of the discussed implementation will be 2-2.5 (receive path) and 2.5-3.5 (send path) dependent on the parameters q and rtt [13].

A TCP/IP implementation [3] running on the same platform as used for performance evaluation of PATROCLOS achieves less than 3000 packets/s and lower speed-up values. The very high performance results of the PATROCLOS implementation indicate that a protocol design appropriate for efficient parallel implementation is extremely useful.

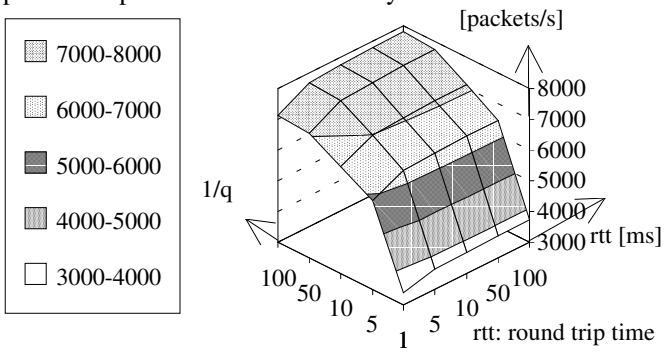

Figure 5: Throughput of send path

References

[1] 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

[2] 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

[3] 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

[4] Braun, T.: *Parallel High Performance Transport System for MANs*, Computer Communications, Vol. 16, No. 2, February 1993, pp. 108-115

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

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

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

[10] Danthine, A.; Baguette, Y.; Leduc, G; Leonard, L.: *The OSI 95 Connection-mode Transport Service: The Enhanced QoS*, 4th IFIP Conference on High Performance Networking, HPN92, December 14-18, 1992, Liège, Belgium

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

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