High Performance Internetworking Protocol

Torsten Braun Martina Zitterbart

Insitute of Telematics
University of Karlsruhe
Kaiserstr. 12
7500 Karlsruhe, FRG;

Abstract

To interconnect high speed networks with data rates of 100 Mbit/s and more high performance internetworking units are required. Due to the protocol processing bottleneck in communication nodes being the most significant limiting factor of high speed networks, solutions for efficient implementations have to be developed. One approach is based on the use of parallelism for protocol processing.

This paper presents parallel concepts and their application in a parallel architecture of an internetworking unit. This architecture consists of pipelines and arrays of processors and supports multiple memory concepts (local and global memory). A high performance parallel implementation of the internetworking protocol in a gateway is discussed and selected performance results are presented.

1. Introduction

The development of communication networks from local area networks with data rates of a few Mbit/s to high speed local area networks (HSLANs) with data rates of some hundred Mbit/s (e.g. FDDI [2]) or even more impose new requirements on the performance of communication nodes and especially internetworking units. Limited bandwidth on the physical transmission media is no longer the performance bottleneck in communication systems. Processing of communication protocols inside the network nodes is the most significant limiting factor of high speed networks.

Interconnections between HSLANs must be very powerful to fulfil their requirements. Today´s network interconnection only achieve data rates of some Mbit/s, which will not be sufficient in HSLANs. Thus high performance interconnections are necessary to interconnect high speed networks.

Faster processors and storage medium and most of all new architectural concepts are necessary to overcome the processing bottlenecks in internetworking units. The use of multiprocessor architectures for the design of future internetworking units will be a powerful decision. This paper presents such an architecture, which is well suited for communication nodes.

Section 2 focuses on the use of parallelism to increase the performance of communication nodes. Based on general concepts to parallelized processing in communication nodes a parallel architecture for high performance gateways is presented. Section 3 describes a parallel implementation of the OSI internetworking protocol providing connectionless service following the parallel concepts discussed in section 2. The implementation environment is based on Transputer networks. In Section 4 performance results of this implementation are discussed. Finally conclusions in section 5 summarize the paper.

2. Parallelism in Communication Systems

Communication architectures are typically structured and described in form of hierarchical protocol layers independent of being standardized as e.g. the OSI Reference Model. Such architectures support parallelism between protocol layers and inside the protocol state machine of a protocol layer.
This section describes parallel concepts, that can be used to realize efficient communication systems. First parallelism based on the structure and the functions of communication systems is considered, followed by a discussion of memory concepts, that should be regarded.

2.1 Parallel Concepts

Communication systems are typically structured in hierarchical protocol layers, as suggested by the OSI Reference Model. This section describes parallelism within those systems. Thereby it is distinguished between structural and functional parallelism. ***Structural parallelism*** can be derived directly from the layered model and ***functional parallelism*** is based on the configuration of protocol functions inside the protocol entities.
The protocol layers operate in a pipelined principle: each packet moves step by step through the protocol layers. Thereby several packets may resist concurrently in different protocol layers. Some of the layers even can be subdivided into sublayers, e.g. the layer 3 into the sublayers 3a, 3b and 3c. Such a layer pipeline performs overlapped computation of sublayers to exploit temporal parallelism. Additionally to hierarchical structuring of sublayers the protocol entities of those layers can be subdivided internally into a send and a receive part resulting in a send and a receive pipeline for each network node (cf. figure 1).
Usually these two pipelines are not completely independent from each other. More in detail, two types of events in a network node can be defined: internal and external events. ***External events*** represent data exchanges between adjacent layers or sublayers. ***Internal events*** appear between the send and receive part of a protocol entity and are used for passing complete packets or control information from the receive to the send part. Internal events, as for example timeouts can initiate external events to other protocol layers (e.g. retransmissions).

As a summary, the ***structural parallelism*** results in two pipelines, which may increase the performance of a network node, because several data units can be processed concurrently in different protocol layers. But there are some problems with those pipelines, because their performance is limited by the processing time of the slowest stage. The pipeline is triggered by the slowest stage and thus the delay within the different pipeline stages must be minimized to achieve optimal performance results. The so called layer cycle is responsible for the performance of the pipeline. It is determined by the maximum processing time within a sublayer [7]:

$$t\ (\text{layer cycle}) = \max \{t(\text{layer } 2) \ldots t(\text{layer } 4)\}$$

An acceptable speed up will only be achievable, if all pipeline stages need nearly the same processing time. Especially due to the different complexities of the protocol layers (e.g. connection oriented and connection less protocols), the utilization of several pipeline stages will not be very high because they are idle for certain time intervals and thus do not improve the performance of the overall system. On the other hand, throughput can increase, if processing time of the pipeline stages decreases. Thereby it must be considered, that the resulting processing time of a stage is higher than the communication time between the stages.

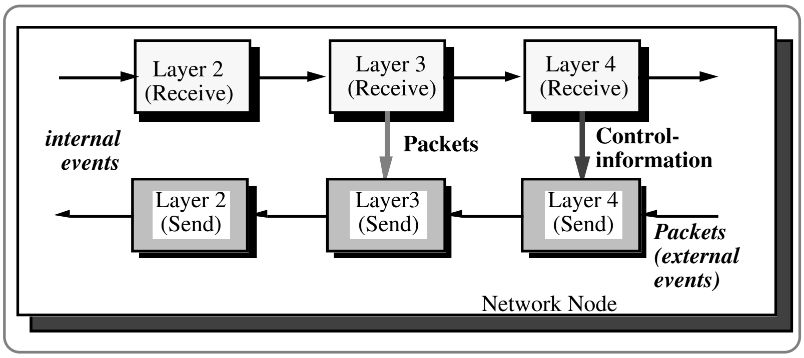

Figure 1 Send and receive pipeline of a network node

The ***functional parallelism*** mentioned above tries to minimize the processing time inside the pipeline stages. Based on the data dependencies between protocol functions, they can be associated with different so called building blocks (cf. figure 2). ***Pipeline blocks*** must be used for data dependent protocol functions, e.g. *routing* and *segmenting* in the network layer. *Segmenting* can only be performed if the *routing* decision is finished, because it needs the maximum packet length of the network connected to the next hop which is determined by the routing function.
Array blocks can be subdivided in those, which consist of data independent protocol functions and in those, which e.g. are used for dividing large databases into smaller ones. In figure 2 an example of subdividing a routing table in three smaller tables is shown. An example for independent functions building an array block can be found in the OSI internetworking protocol: The *routing* function can be computed in parallel with the *header format analysis* and the *PDU lifetime control*. In the case of an error in the PDU the *routing* function has been executed uselessly. But in the case of fiber based local area networks with low error rates these situations are negligible.
Additionally communication protocols include functions which cannot be included into pipeline and array blocks, e.g. the timer function. They are configured in so called ***irregular blocks***.

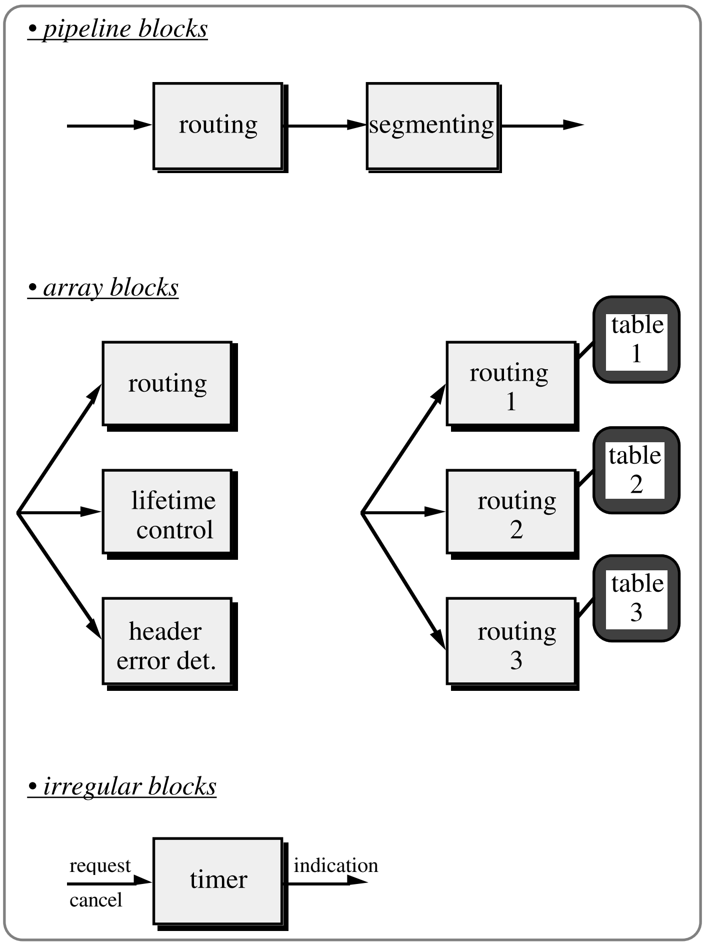

Figure 2 Building blocks

2.2 Memory Concepts

Similar to the building blocks above, the passive components of a communication protocol, the memory can be structured into several memory components, which should be integrated into a multiprocessor architecture (cf. figure 3).

To reduce overhead caused by data movements from one stage of the pipeline to the other a ***global data memory*** is necessary. The user and the network have direct access to that memory and write the data units into the memory. The protocol functions receive an identification of the protocol data unit in the global memory and additional control information.

They add the packet header to the data unit, but don´t move them inside the memory. Subdividing the global memory in receive and send memories further reduces the number of processes that have access to the memory and thus helps avoiding memory contentions.

Furthermore a ***global layer memory*** will be necessary in connection oriented protocol layers. All relevant connection data are stored in this memory, which can be accessed from the receive part as well as from the send part. Other processes will only have a ***local memory***, e.g. the routing process needs a routing table to which other processes need not to have access.

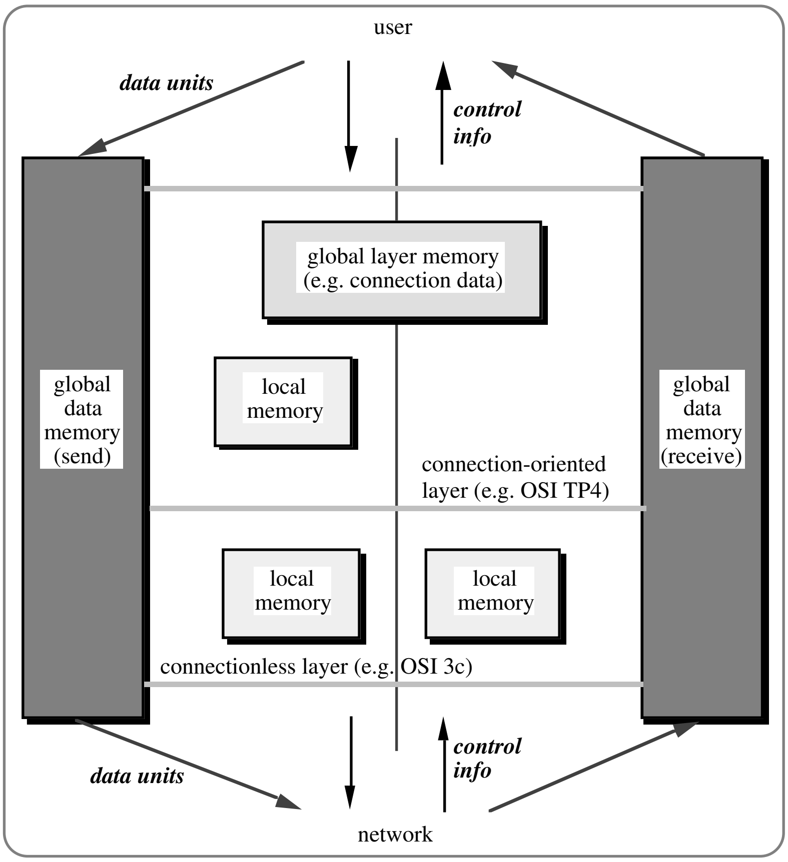

Figure 3 Memory concepts

2.3 Parallel Gateway Architecture

Based on the concepts described above a parallel gateway architecture can be derived. Throughout this paper we focus on the design of a router being a layer-3-gateway.

Generally gateways must solve problems, which arise with incompatibility of the protocol layers in nodes associated with different networks. One of their main functions is forwarding of protocol data units from a receiving network to a sending network, which is connected to the next hop on the route. The International Organization for Standardization (ISO) has defined an internetworking architecture, which subdivides the network layer in three sublayers 3a, 3b and 3c as shown in figure 4. Layer 3a (subnetwork access function, SNACF) is responsible for access to the underlying network. Layer 3b (subnetwork dependent convergence function, SNDCF) must adapt the service provided from the subnetwork to the service required from layer 3c (subnetwork independent convergence function, SNICF). A high performance parallel implementation of the layer 3c, being a central protocol layer in a router, is described in the next chapter.

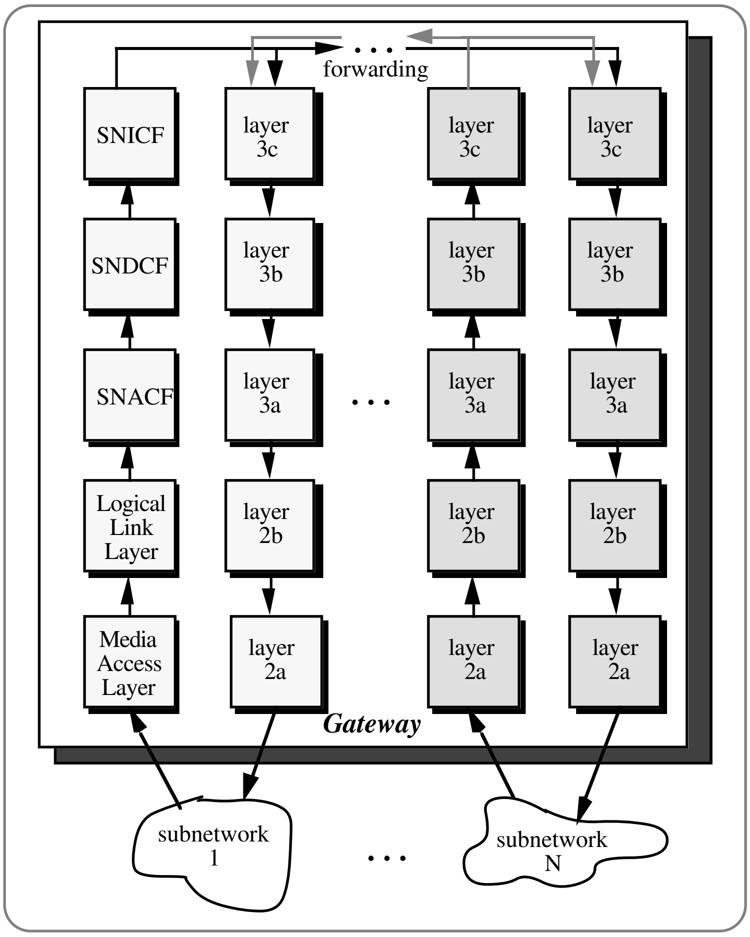

Figure 4 Parallel Internetworking Unit

Considering the concept of structural parallelism as described above, a gateway consists of several receive and send pipelines (cf. figure 4) each connected to a dedicated subnetwork. Thus temporal and spatial parallelism can be achieved: the receive pipeline of a subnetwork processes packets from this subnetwork in temporal parallelism, whereas receive pipelines of different subnetworks process packets in spatial parallelism. In figure 4 a sublayer is depicted by a grey rectangle, which itself represents a multiprocessor architecture based on pipelines and arrays of processors. An example thereof is described in the next section [6].

Each send and receive pipeline has its own global data memory as described above to avoid memory contentions. To avoid data movement between the receive and the send pipeline an interconnection network is necessary, that allows the send pipeline to have access to the global data memory of the receive pipeline of the incoming network.

In the following section a high performance implementation of the OSI internetworking protocol, the layer 3c in a gateway, is described.

3. Parallel Implementation of the OSI Internetwork Protocol

This section deals with a high performance implementation of the OSI internetwork protocol on Transputer networks according to the parallel architecture proposed in the previous section. The parallel implementation has been realized with Transputer networks, because they provide an efficient and flexible environment.

3.1 Implementation environment

The Transputer is a family of high performance microcomputers with a processor, a timer, on-chip memory, an external memory interface and four communication links on a single chip (cf. figure 5). One of its most significant features is the ability to perform multi-tasking in hardware. The Transputer uses processes as the fundamental standard software building block. Processes are able to communicate using the synchronized and unbuffered rendezvous mechanism [5]. Transputer networks are built by connecting the links of different Transputers together. Due to the rendezvous-based communication Transputer networks are loosely coupled multiprocessor systems with distributed memory.

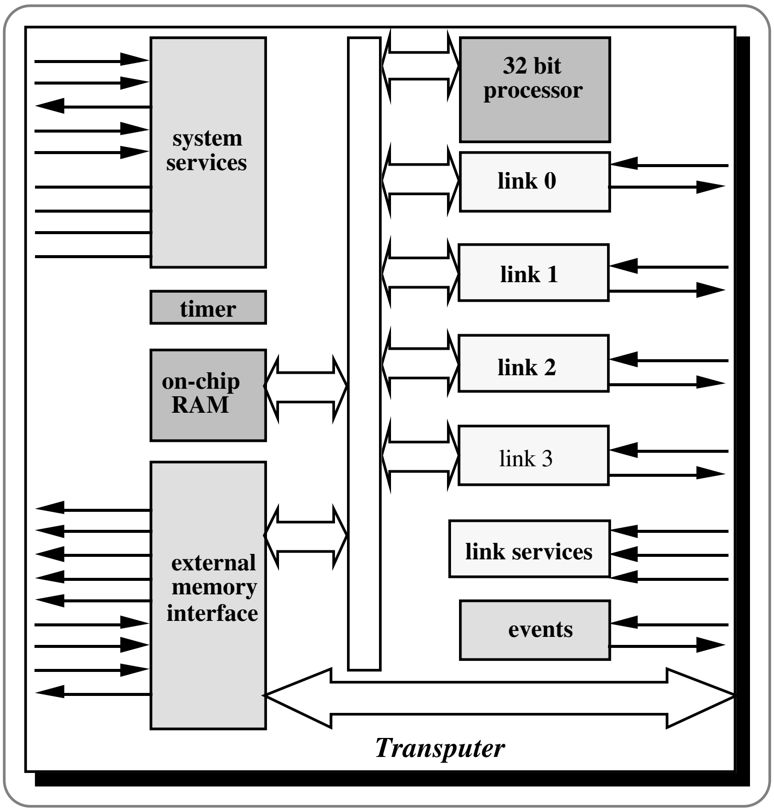

Figure 5 Transputer architecture

The Transputer was developed concurrently with the programming language OCCAM, but is also well suited to other languages as e.g. the C language. During the project described in this paper a parallel TCC (Transputer C Compiler) [3] is being used, which provides the most significant features of the OCCAM language including the datatype "channel", the "alt"-construct and the timer variable.

The "alt"-construct allows an alternative waiting for inputs on different channels and for timer events. TCC supports an easy distribution of processes to processors by defining the network structures with precompiler instructions. Adaptions to various Transputer networks only require changes within these instructions, but no changes in the internal code of a process.
Throughout the project Transputer boards with up to four T414 Transputers on a single board have been used. These boards are plugged into PC/AT systems.

3.2 Implementation Model

The parallel concepts described in section 2 comprise structural and functional parallelism. Therefore the protocol is subdivided into its protocol functions, which can be configured in pipeline or array blocks. In the parallel implementation of a part of a layer-3-gateway described in this paper every protocol function is reflected as a single process. The general structure of a typical process is shown in figure 6. Processes are waiting for an input message on a communication channel, execute some code dependent on the input channel, send results to other processes and wait again for new incoming messages.

```
void process(channel1, channel2, channel3)
channel *channel1, channel2, channel3;
{
   for(;;)
      alt
      {
         guard channel1:
            read_packet(channel1, length, address);
            …
            send_packet(channel3, length3, address3);
            break;
         guard channel2:
            …
            break;
      }
}
```

Figure 6 Process structure

Moreover in section 2 a global memory concept was discussed, which should avoid a performance decrease caused by unnecessary movements of protocol data units during protocol processing. This concept can not be realized directly with Transputer networks, because they do not provide any global memory. Therefore so called *Memory Managers* had to be implemented which emulate the global

receive and send memories. Both, the Memory Manager for the receive memory (*MMR*) and the Memory Manager for the send memory (*MMS*) are implemented on dedicated Transputers, whose local memories are logically used as global memory. A special developed protocol between the protocol processes and the memory managers handles memory access.
Figure 7 depicts the architecture of the parallel implementation of the OSI internetwork protocol being an essential part of an OSI router. The Memory Managers are placed on the right and left side of the figure. Furthermore a load generator (*Testuser*) and a network simulator (*Network*) to connect sending and receiveng entities had to be implemented for test purposes.

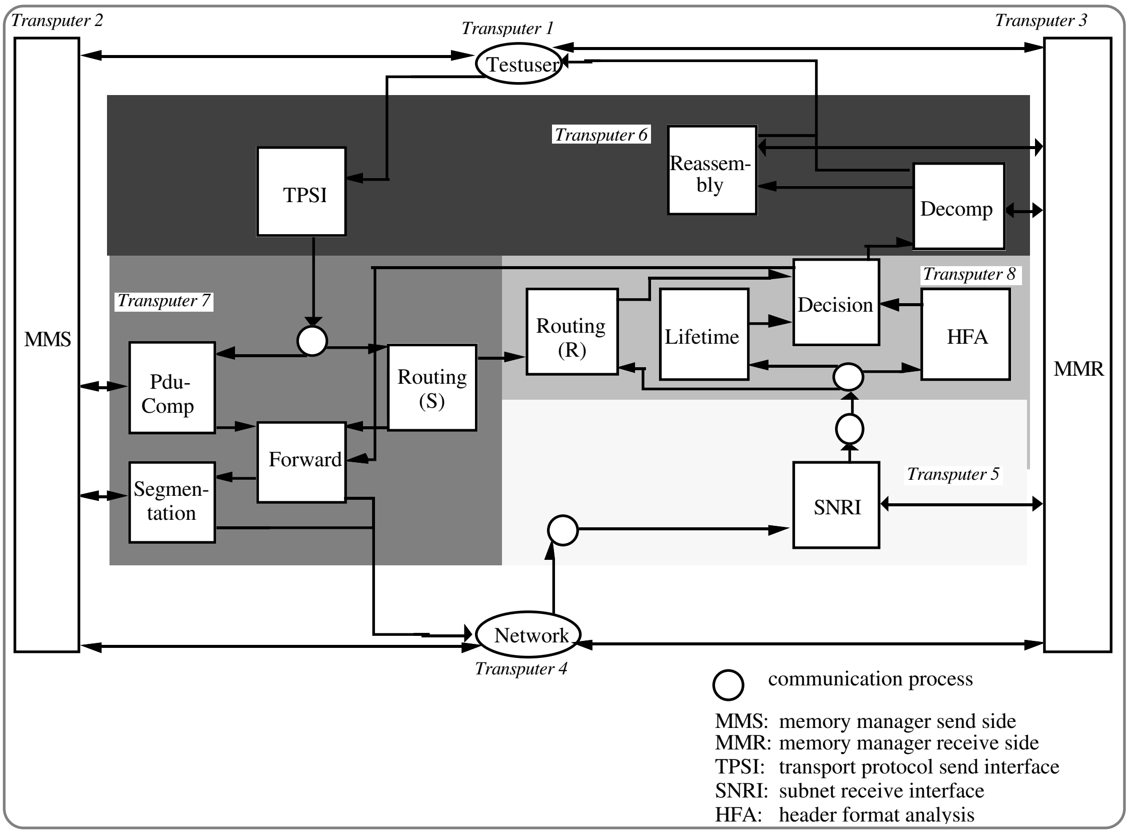

Figure 7: Parallel implementation of the internetwork protocol

The main functions of the OSI network protocol itself are composition and decomposition of protocol data units (PDUs), routing between the networks, segmentation and reassembly of PDUs, lifetime control, error detection and reporting. In figure 7 the structure of the implementation is shown, which reflects the functions of the OSI internetwork protocol being implemented as processes. Thereby the routing process was splitted in two independent routing processes to decouple the send and receive part of the protocol implementation. The concepts of pipeline and array blocks as discussed above are also reflected in this implementation. For example in the send pipeline an incoming packet is first processed by *TPSI* and then concurrently by *Routing(S)* and *PduComp* and thereafter by the *Forward* process being the next pipeline stage. In the receive pipeline these concepts can also be seen. There *Routing(R)*, *Lifetime* and *HFA* (Header Format Analysis) can process the same packet at the same time and thus build an array block.

As explained above the implementation can easily be adapted to various Transputer networks. Actually there mainly exist four configurations of the implementation, which have been used for performance evaluation purposes. The basic configuration uses 1 Transputer for all processes, independent of being protocol processes, which realize protocol functions or memory managers and test processes. A configuration with 5 Transputers separates the protocol functions from the test and memory management processes, which each are placed on a dedicated Transputer. Thereby the protocol processes are still all configured on the same processor. Additionally there exists a configuration with 8 Transputers (more Transputers have not been available), where the protocol processes have been distributed over four Transputers. The implementation of these configurations do not include the segmentation and reassembly functions, therefore another 8 Transputer configuration exists. The Transputer network for this configuration is depicted in figure 7. In the following these configurations will be referred to as C1, C5, C8 and C8+, where the number indicates how many Transputers have been used for the configuration. The "+" indicates that segmentation and reassembly are realized.
Only the configurations C8 and C8+ contain the so called communication processes, which enable an improved decoupling of protocol processing and interprocessor communication. Thus they will increase the achievable performance, because the protocol process needs not to wait for the communication with processes on another processor and thus can continue working. The communication between the Transputes will then be performed by the communication processes.
In the following section performance results of the OSI internetwork protocol on the different configurations are discussed.

4. Performance Evaluation

Several measurement experiments have been performed with the various implementation configurations to evaluate the achievable performance, e.g. throughput and delay /Brau90/. Especially in multiprocessor systems it also is interesting how efficient additional processors can be used to increase performance. Therefore the traditional measures in multiprocessor architectures ***speed up*** S,

$$S = \frac{T_1}{T_n},$$

T_i: processing time for an algorithm for i processors performed on an architecture with i processors)

and ***efficiency*** E,

$$E = \frac{S}{n},$$

n: number of the processors)

have been investigated. As a rule efficiency is less than 1 (S < n). For this so called ***efficiency loss*** there exist different reasons:

- A processor can be idle.
- A procesor executes redundant computations, which are not necessary for the solution of the considered problem.
- A processor executes additional computations, which are not necessary on a monoprocessor architecture, but which occur on a multiprocessor archtitecture.

The efficiency loss can be caused by the system (hardware or operating system) or by the application software. In Transputer networks mainly idle times and communication times are responsible for the efficiency loss. During the ***idle times*** neither the processor nor the link managers are busy and during the ***communication times*** the processor is idle, but the link managers perform some interprocessor communications. Other reasons for efficiency loss like e.g. process switching can be neglected on Transputers, because process switching is hardware based and takes less than 1 µs.

To investigate the achievable performance of the OSI internetwork protocol three types of measurement experiments have been performed:

- type ***S*** (send test, simplex traffic),
- type ***R*** (receive test, simplex traffic) and,
- type ***D*** (combined send and receive test, duplex traffic).

On type S the send part of the internetwork protocol is loaded by the upper layer with packets of constant length and minimal interarrival time and on type R the receive side is loaded by the lower layer with the appropriate load. On type D the internetwork layer is loaded concurrently by the two neighbour layers to simulate duplex traffic.
As monitor tools three event-driven monitors have been used to investigate the behaviour of the implementation: a software monitor, a logic analysis system and the distributed monitor system NETMON-II [4]. Selected results of measurements with these monitors are comprised in the following subsections.

4.1. Throughput and Delay

In figure 8 throughput and delay results for the Transputer network configurations mentioned above are presented. As described above the implementation is designed according to the concepts presented in section 2. This especially means, that no packets have to be moved during protocol processing and thus throughput results are independent of packet length. A data packet first is copied by the user into the memory managed by the *Memory Managers*. To pass packets from one layer to another only references to the packets are delivered.

Figure 8 shows, that sending is generally faster than receiving. The throughput differencies between the considered configurations on measurement type S are not as high as on type R. That means, that parallelization on send part is not as successful as on receive part, because interprocessor communication especially on the send part of the configurations C5 and C8 causes a very high overhead.

In contrast to that in the case of duplex traffic even in the send part an impressive performance increase is achievable by the use of eight Transputers. The reason therefore is the placement of the send and receive pipeline on different processors. So the two pipelines have less influence on one another. Thus differencies between simplex and duplex traffic are very small on the configuration C8 in contrast to C1 and C5.

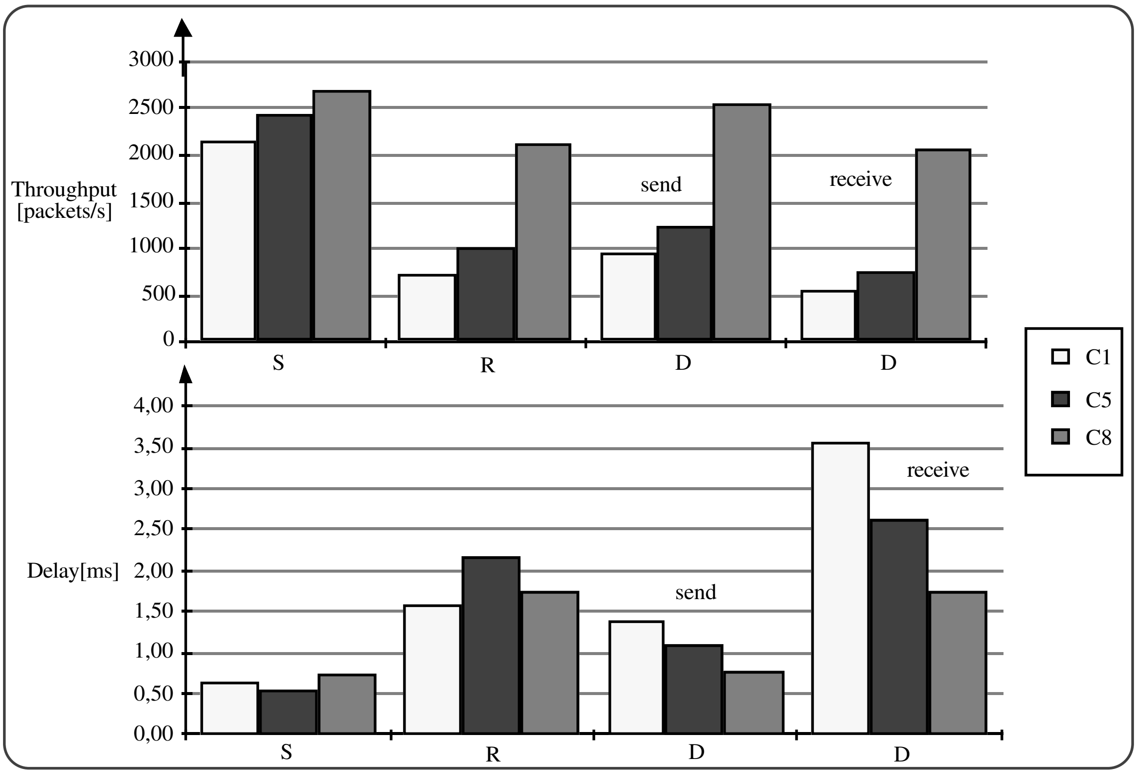

Figure 8

Throughput and delay results

It is remarkable that C5 results in throughput measures, which are up to 37 % higher as C1, where the internetwork protocol and the simulation processes of the other layers are placed on the same processor. This result emphasizes that processing of different protocol layers on different processors leads to better performance results.

The throughput on configuration C8 on type D reaches up to 2550 packets/s (i.e. 170 Mbit/s with 8 kByte packet length) on send side and 2100 packets/s (140 Mbit/s with 8 kByte packet length) on receive side. On C5 the troughput results only come to 1240 and 760 packets/s (80 and 50 Mbit/s) [1].

Additionally in figure 8 measurement results concerning the delay of packets in the protocol layer are depicted. It can be seen that delay increase is not as high as could be expected in pipeline structures.

In a pipeline a packet often must wait before it can be processed by the next pipeline step, thus delay increases.

4.2 Speed up

The performance measure speed up usually is used to compare the performance of solutions on multiprocessor architectures with those on monoprocessor systems. In our case we also compared the configurations C5 and C8 with each other, because there the protocol processes were distributed from a single Transputer to four Tranputers.
Table 1 shows the speed up results, which are based on the comparison of the throughput measures. Especially receiving on C8 with type D is very powerful compared with C5. C8 uses four processors for processing of the internetwork protocol and achieves a speed up of nearly three in the case of receiving on duplex traffic, which is very impressive. The speed up of the configuration C8 to C5 is only slightly better than the result of C8 to C1, because the protocol processes being the most processing intensive processes still do not run in parallel on C5. The other processes *MMR*, *MMS*, *Network* and *Testuser* are very short, because mainly they receive or send messages to the protocol processes.

measurement type	S	R	D (send)	D (receive)	
speed up (C8 / C5)	1.11	2.10	2.05	2.74	
speed up (C8 / C1)	1.27	2.89	2.65	3.74	

Table 1: speed up

4.3 Process Execution Times

To find a reasonable well configuration of processes on multiprocessor architectures the execution times as well as the communication times must be considered. Therefore the execution times of the protocol processes have been measured. Figure 9 shows that they differ widely from each other. These results must be considered for the design of new configurations, so that the procesor utilization is distributed uniformly.
It can be seen that the processing time of *SNRI* is very high compared with other processes and thus is the bottleneck on the receive part. For further performance improvements this process must be optimized or divided into subprocesses.

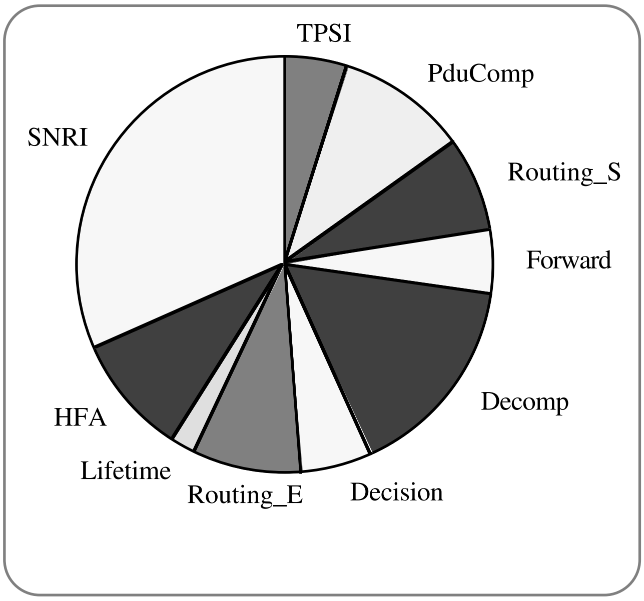

Figure 9 Process execution times

4.4 Segmentation and Reassembly

In internetworking units segmentation and reassmbly of protocol data units are of high significance, because of the different characteristics of underlying networks. Thus some results of C8+, where the functions segmentation and reassembly are incorporated, will be discussed in this subsection. In figure 10 the influence of these functions on throughput results is presented. For segmentation measurement type S was taken, for reassembly measurement type R.

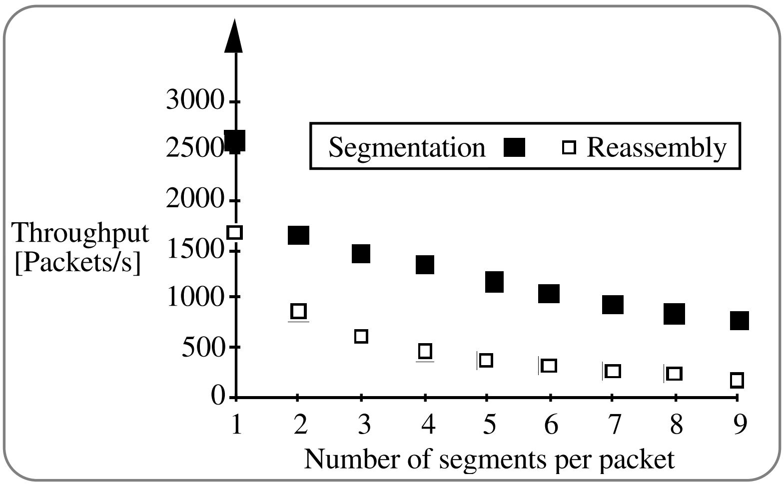

Figure 10 Results of segmentation and reassembly

It can be seen that the *Segmentation* process could be implemented very powerful. Copying of data segments was avoided, only the headers of internetwork protocol data units are moved inside the global data memory. The influence of *Reassembly* process is not as important as the *Segmentation* process. Reassembling n segments is as powerful as receiving n packets, because *Reassembly* is faster than *SNRI* being the slowest process in the receive pipeline. Thus throughput measured in packets/s is inversely proportional to the number of segments per packet.

4.5 Additional Results and Remarks

Beside the performance measures discussed above there are some additional results which are worth to be presented here. As mentioned above on configuration C8 there are additional so called communication processes implemented, which have the task to minimize communication times of a process. By these processes additional computations are performed. The efficiency loss of C8 in comparison with C5 caused by these computations lies between 4 and 10 % dependent on the measurement type. The efficiency loss caused by idle times comes to 8 - 16 %, the loss by communication times to 17 %.
Moreover especially the link rate is an important limiting factor for performance of Transputer networks. For example some measurement experiments have shown results, which are up to 25 % worse, if 10 Mbit/s link rate was used instead of 20 Mbit/s. Thus it can be expected that performance results presented in this paper will further increase with higher link rates.
Mainly communications between internetwork protocol processes and memory management processes are responsible for the communication times, because they claim about 80 % of the total communication times. This reflects our experiences with other configurations, where the processes have not been distributed as carefully as in the configuration depicted in figure 7 and thus the achievable performance was decreased by high communication times.
The costs for process switching in the presented implementation are only 1-2% of processing time, because in Transputers it is hardware supported.

5. Conclusions

In this paper an approach for the design high performance internetworking units, which is based on the use of parallel concepts, is presented. A multiprocessor gateway architecture according to OSI proposals and a parallel implementation of the respective internetwork pr otocol is described. Selected performance results considering throughput and delay are discussed. The results are very impressive and show, that requirements of high speed networks with throughputs of more than 100 Mbit/s can be fulfilled with the proposed parallel architecture and implementation.
As a summary it can be stated, that the use of parallelism in communication systems, endsystems as well as gateways, is a promising solution for future systems.

References

/Brau90/ Braun, T.
Monitoring von Transputer-Netzwerken am Beispiel des OSI-Internetzwerkprotokolls
Diploma thesis , Institute of Telematics, University of Karlsruhe, January 1990

/Burr88/ Burr, W.E.; Zuqui, L.
An Overview of FDDI
Proceedings EFOC/LAN ´88, Amsterdam, 1988

/Defi88/ Definicon Systems Inc.
The Parallel Transputer System: Operation and Installation Manual
Release 2.0, 1988

/ESZi90/ Endriss, O.; Steinbrunn, M.; Zitterbart, M.
NETMON-II a monitoring tool for distributed and multiprocessor systems
Fourth International Conference on Data Communication Systems and their Performance, Barcelona, June 1990

/Mitc90/ Mitchell, D.A.P.; Thompson, J.A.; Manson, G.A., Brookes, G.R.
Inside the Transputer
Blackwell Scientific Publications, 1990

/Zit89a/ Zitterbart, M.
A Multiprocessor Architecture for High Speed Network Interconnections
IEEE INFOCOM ´89, Ottawa, Kanada, 1989

/Zit89b/ Zitterbart, M.
High Speed Protocol Implementations based on a multiprocessor architecture
in: Rudin, H.; Williamson, R.: Protocols for High Speed Networks
North Holland, 1989