# Performance Evaluation of a Speculative Network Processor

Jürgen Foag, Maresa Praxenthaler, Thomas Wild [a]

[a] Institute for Integrated Circuits, Technical University of Munich, 80290 Munich, Bavaria, Germany
{Juergen.Foag, Maresa.Praxenthaler, Thomas.Wild}@ei.tum.de

Beside high bandwidth, real-time networking applications mainly require short end-to-end latencies. Up to now, network processors are focused in particular on application flexibility and throughput, but generally neglect fundamental new approaches concerning processing delay reduction. This paper describes the implementation and the evaluation of the parallel speculative packet processing architecture and of a reference architecture We show that a latency reduction of up to 26.3 percent can be obtained by the speculative approach in comparison to traditional implementations.

*Au delà d'une forte bande passante, les applications réseau en temps réel demandent de basses latences. Jusqu'à maintenant, les processeurs réseau sont dédiés en particulier à la flexibilité des applications et au débit, mais négligent généralement la réduction du temps de traitement. Cet article expose l'implémentation et l'évaluation d'une architecture parallele de traitement spéculatif de paquets, par rapport à une architecture de référence. Nous montrons que cette approche spéculative peut mener jusqu'à 26.3% de réduction de la latence par rapport à des implémentations traditionnelles.*

## 1 Introduction

Distributed high performance computing systems possess a strong dependency on the performance of the underlying network infrastructure. By increasing network throughput, the performance bottleneck is shifting from the bandwidth of transmission media to the processing capacity of hosts and transmit systems like routers [1]. The performance of these devices is significantly affected by their architecture [2]. High-speed networking consists not only of the quest for high bandwidth. In the same way, low latency as well as the ability to cope with high bandwidth-x-delay product paths is required [3].

Recently, *network processors (NP)* have been introduced to cope with throughput requirements of OC-48 (2.5 Gbit/s) and higher. They normally comprise multiple micro-programmable RISC processor cores, each realizing a simultaneous multithreading architecture [4]. By means of an efficient software implementation of message-parallelized protocol processing, high performance on NPs can be achieved. In addition, some NP provide dedicated hardware blocks to accelerate calculation intensive networking functionalities.

While these properties made a throughput increase possible, optimization of packet processing delay on a NP was rather uncared. To make real-time applications possible, the delay problems are bypassed: For instance, *content delivery networking (CDN)* shifts centralized real-time application content through the network closer to the user to distributed content engines [5]. Consequently, essential drawback is a high administration effort for the distributed system components.

A new concept called *speculative packet processing* was presented in [6]. Therein, the benefits were demonstrated by an abstract concept verification. Now in this paper, the ongoing work is presented, which describes the system implementation and presents the results of the performance evaluation. The principal contribution is the demonstration of the benefits of the new concept for packet processing based on a real networking simulation environment.

The remainder of this paper is organized as follows. Section 2 gives a short overview to previous work done in packet processing optimization and parallelization. The concept already presented and the architecture model are briefly summarized in section 3. Section 4 describes the performance evaluation of the speculative system and discusses the results. Finally, the conclusion and future work is discussed in section 5.

## 2 Related Work: Protocol Processing Methodologies

To cope with increasing transmission rates for high-speed networking, extensive research of protocol processing methodologies has been done or is still done [7] [8] et al. Principally, the results can be classified into suitable *protocol specifications* and appropriate *protocol implementations* [9]. The most common methodologies are:

*Parallel Protocol Processing:*

To allow parallel protocol processing, parallelization uses replications to perform the work of a single processing unit. In [10], various process architecture models are summarized. They fall into three general categories: *horizontal* (layer parallelism, functional parallelism), *vertical* (connection parallelism, message parallelism) and *hybrid*. Although each process architecture has different structural characteristics, it is generally possible to implement the same protocol functionality [11].

*Pipelining:*

Pipelining is an implementation technique whereby multiple instructions are overlapped in execution [12]. To achieve a performance gain by processing packets in

parallel, protocol layer tasks can be assigned to different pipeline stages. Packet data enter into the pipeline, progress through the connected stages and exit the pipeline. Thus, processing latency corresponds to the sum of individual pipeline stage delays.

*Multithreading:*

The majority of network processors use multithreading (MT) as execution concept. Originally, MT was introduced in the domain of microprocessors with the target of an increased number of instructions per cycle. It is enabled through the capability of context switches between threads to obtain an efficient load of CPU resources. Examples for the different kinds of MT are coarse-grained MT [13], fine-grained MT [14] and simultaneous MT [15].

*Jacobsons' header prediction and Woodsides' protocol bypass:*

As in [16], Jacobson header prediction supports unidirectional data transfer (TCP) by handling two common cases:

1. If TCP is sending data, the next expected segment for this connection is an ACK for outstanding data.

2. If TCP is receiving data, the next expected segment for this connection is the next in-sequence data segment.

In [17], Woodside presents a generalization of Jacobson's Header Prediction using a protocol bypass on sender and on receiver side. For frequent receipts of packets with identical protocol control information, the usual implementation of the protocol stack is bypassed via a fast path.

*Summary:*

Dominating factors on packet processing time for simple data manipulation functions, such as checksumming, and field value incrementation are the primary memory access times, i.e. time to write and read. Consequently, performance will most likely be limited by the capacity of the shared bus and the memory system [1]. This is intensified by the fact that access latency of memory systems using DRAMs is decreasing at a slower speed than CPU clock rates are increasing [18]. Parallel processing is focused on increasing protocol processing performance in terms of system bandwidth [19]. However, processing latency of individual packets stays unaffected. While pipelining increases the packet throughput, the complete latency which is the sum of the individual stage processing latencies keeps unchanged [20]. Multithreading is an effective technique when multiple threads share a system. But, single-thread performance is not improved [21]. Fig. 1 shows exemplarily two threads $T_1$ and $T_2$ which are processed multithreaded or speculatively, respectively. Each comprises the packet processing tasks of one data packet.

The latency of a packet in case of multithreading support is the sum of processing delay of the context in the processor units (PU), memory (MEM) access times and arbitration times of shared resources. It is obvious that processing latency of the second packet in case of multithreading support $L_{M_2}$ is less than in case of starting it not until termination of the previous one.

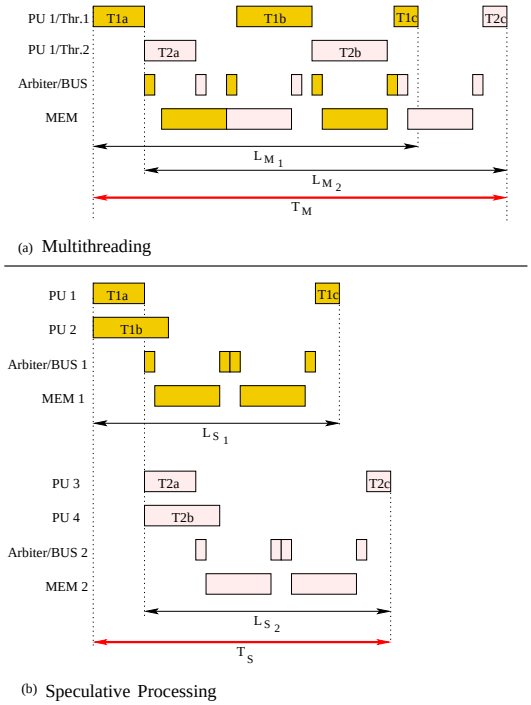


**Figure 1.** Multithreading and Parallel Processing

However, a decrease of packet processing delay, i.e. the time interval between thread processing start $T_{1_a}$ and end $T_{1_c}$ in conjunction with memory requests, is only achieved if shared resources between multiple packet contexts are avoided.

In case of speculative processing (b), the complete thread that refers to one packet is subdivided into several. Arbitration for the memory bus as well as MEM is shared between them. By duplication of shared resources, i.e. MEM1 and MEM2, the access bottleneck between contexts of different packets can be eliminated. Thus, latencies $T_{S_1}$ and $T_{S_2}$ are significantly reduced in comparison to their multithreading counterparts.

Weaknesses of protocol bypass and TCP header prediction are constraints which are inherent to the methodology. The applied prediction is based upon control and state transition information of connection-oriented TCP traffic, e.g. connection establishment and termination. The state information of a packet is merely kept in network end systems, but not in transmit nodes [22]. Furthermore, prediction of connectionless traffic, which is used commonly used for real-time applications, i.e. UDP, is not supported [23].

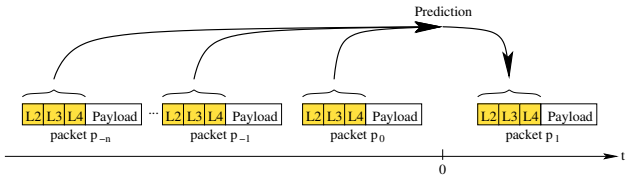


**Figure 2.** Protocol Stack Prediction Scheme

## 3 Protocol Processing Methodology & Architecture

The concept is based on two components: *Protocol Stack Prediction* and *Speculative Packet Processing*. Using both, this packet processing concept serves to overcome processing latency limitations which result from network layer hierarchies.

*Protocol Stack Prediction:*

The decision how to handle and forward a packet is done upon the information which is contained in the packet header. The header of the data-link layer, the network layer and the transport layer is hierarchically arranged. I.e. the type of a higher layer of the ISO/OSI reference model of communication is derived from the header of the lower layer [24]. To circumvent a serial extraction, *protocol stack prediction* predicts the protocol stack of the next packet. The scheme is depicted in Fig. 2.

The prediction is based on the history of packets received earlier. It should be noted, that the prediction *not* only delivers protocol layer types of a packet. In a store-and-forward architecture, this information could be gathered quickly from the packet but would not influence significantly processing delay. Furthermore, additional information which is required for control point processing is implicitly provided by the prediction. For instance, costly processing for checksum verification of layer 2 has to be done in front of processing of layer 3 as in Fig. 3.

*Speculative Data Packet Processing:*

Based on the prediction results, execution of protocol stack processes is done as described above. In this context, a process means a functional task of packet header field analysis or manipulation. Examples are value verification, checksum calculation or address compare. Their execution order depends on associated protocols, supported applications (e.g. Differentiated Services or Multiprotocol Label Switching) as well as on requirements of the networking environment (access, core or edge). In Fig. 4, the flow of speculative data packet processing is illustrated.

Tasks $T_{Li_j}$ of one protocol layer $i$ are grouped to a *thread*. The order of processing is indicated by an increasing value of $j$. Threads for protocol layer 2, 3 and 4 are started simultaneously. The determination of the protocol layer type of layer 3 is done by execution of a so-called

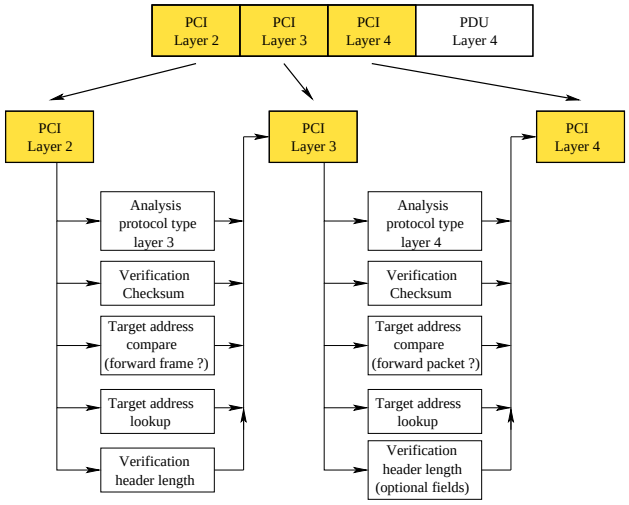


**Figure 3.** Protocol Stack Prediction

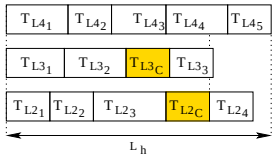


**Figure 4.** Speculative Packet Processing

checktask $T_{L2_C}$ which is highlighted in grey. For layer 4, the checktask is $T_{L3_C}$. By finishing checktask processing, the data dependency for the next higher protocol layer is resolved (see dashed line). Two cases can occur: a prediction hit or a misprediction. The first case, which is illustrated in the figure above, delivers a correct assumption of the higher layer type. Consequently, execution of tasks for the upper layer has been made on correct input data and the results can be kept. Tasks which are simultaneously processed can be proceeded. Remaining tasks can be executed in the following. The complete processing time is called hit latency $L_h$. Else, in case of a misprediction for layer 3, which results from analysis of $T_{L2_C}$, a second prediction is requested. It provides the layer type of layer 4 based on knowledge of layer 2 and 3. Dependent on its output, either a prediction hit for layer 4 follows the latency $L_{mh}$, Figure 5, or a misprediction $L_{mm}$ (not shown). Both latencies possess a larger value than $L_h$.

Summarized, the mean processing latency for a sample of N packets is

$$\overline{L} = \frac{1}{N}\left[N_h L_h + N_{mh} L_{mh} + N_{mm} L_{mm}\right] \qquad (1)$$

with the number of packets $N_h$ for case hit, $N_{mh}$ for case layer three miss / layer four hit and $N_{mm}$ for case layer three miss / layer four miss. Goal of the concept is to

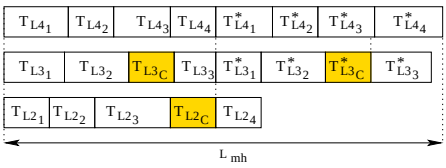


**Figure 5.** Speculative Packet Processing: Case Layer 2 miss/ Layer 3 hit

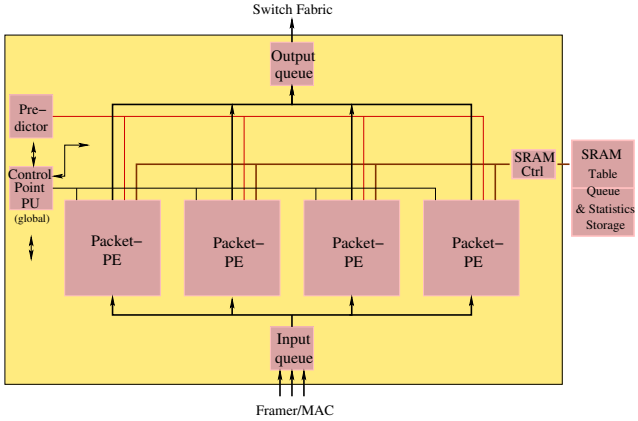


**Figure 6.** System architecture

achieve a value for $\overline{L_N}$ which is less than in case of traditional processing.

*Architecture*

In Fig. 6, the hybrid system architecture is depicted. It implements per-packet parallelism with one received packet per packet processing element (PPE). The predictor unit delivers the value that stands for the protocol-stack characteristic. The global control-point processing unit (PU) maintains the system procedure.

In detail, one of the four PPE from above is illustrated in Fig. 7. The received packet data is stored in DRAM. Based on predicted header boundaries, the headers of layer 2, 3 and 4 are stored in the general purpose registers of the assigned network-processing units (NW-PU). One NW-PU completely processes the layer packet header. The local prediction memory disposes of a small table which contains the prediction values.

## 4 Performance Evaluation

To evaluate the proposed processing concept within a real networking environment, the system implementation is done in SystemC [25]. As reference model, an additional implementation is done which follows the pseudo-parallel processing methodology. This method disposes of identical parallel processing resources. Only difference is the lack of

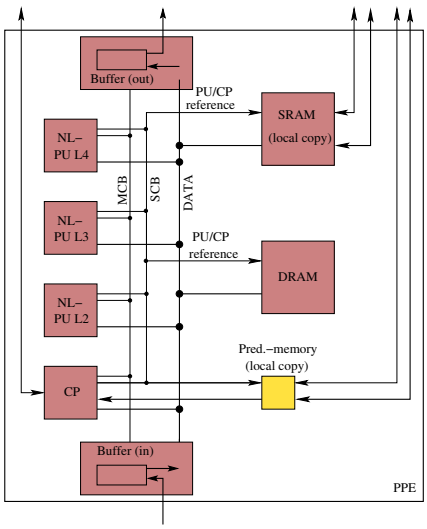


**Figure 7.** PPE architecture

a prediction unit and the fact, that processing of upper protocol layers is started immediately after resolution of their types. Thus, the protocol stack differs from another only in the predictive process steps. The system is stimulated at its ingress by actual network traffic which is composed as listed below (T: *Traffic*; GE: *Gigabit E.*):

| T | Protocol Stack |
|---|---|
| 1 | 95% TCP/IP4/GE 5% UDP/IP4/GE |
| 2 | 50% TCP/IP4/GE 50% UDP/IP4/GE |
| 3 | 50% TCP/IP4/GE 50% UDP/IP6/GE |
| 4 | 50% TCP/IP4/GE 50% UDP/IP6/PoS |
| 5 | 80% TCP/IP4/GE 20% UDP/IP4/GE |
| 6 | 100% TCP/IP4/GE |

While $T_1$ corresponds to real protocol distributions [26], remaining traffic serve only for performance studies under worst case conditions.

Fig. 8 shows the mean latency per PPE and per packet. For traffic 1, the number of processing cycles for the speculative is 858 versus 1107 in the pseudo parallel case. The least difference occurs for traffic 4 with 1160 versus 1239 cycles. For actual network processor cycles of 400 MHz, the difference of 249 cycles concludes to a latency reduction of 0.6225 $\mu$s. Along the networking path, at each Internet service provider ingress, time consuming diffserv functionalities have to be done, e.g. packet classification and policing. Thus, multiple times the reduction can be achieved. Considering real-time applications and additional queuing delays along the network path, the putative small latency reduction of one network processor cannot be neglected.

Thus, the latency reduction r within a PPE for traffic 1 is 26.3 % as in Fig. 9. In case of a worse prediction hit rate, which is achieved for traffic 4, r is 6.3 %. However, it is nevertheless positive.

Considering a network processor cycle of 400 MHz and

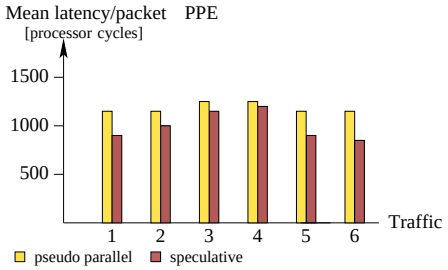


**Figure 8.** PPE Latency

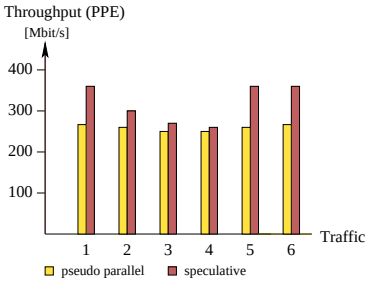


**Figure 10.** Throughput

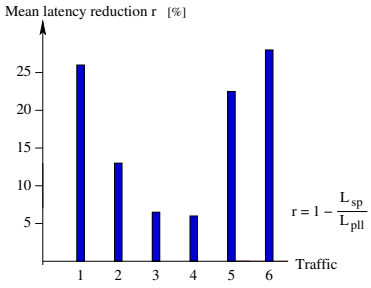


**Figure 9.** Latency Reduction

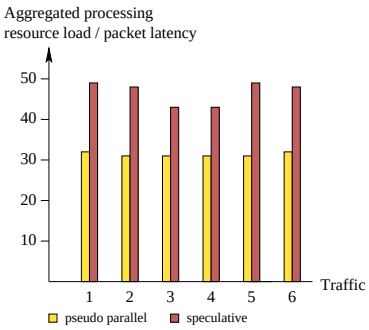


**Figure 11.** Aggregated Processing Resource Load

simulation packets of 96 bytes length, the throughput for traffic 1 of one speculative PPE is 358 Mbit/s. In case of pseudo-parallel processing, 278 Mbit/s is obtained.

Fig. 11 shows the aggregated processing resource load. It is the sum of busy times of the layer-PU 2, 3 and 4 compared to three times the simulation duration. The load of the speculative system is about 12 to 18 percent higher than the reference model load. This results from blocked shared memory requests due to concurrent access trials. Furthermore, for traffic 1 to 5, occurring mispredictions require additional layer processing. Despite of a worse prediction hit rate in case of traffic 3 and 4, the value of the relative system load is less due to a larger amount of processing cycles.

Fig. 12 shows the layer-processor load separated into layer-PU 2, 3 and 4. In particular, costly diffserv functionalities comprise classification, policing and accounting. These tasks are processed by the layer-4 processor. In case of mispredictions, the load is 77 percent, while 44 percent load is used in case of pseudo-parallel processing.

Reasons for the significant increase of system load can be derived from Fig. 13. Due to concurrent trials to accesses shared resources, the shared bus arbiter of the speculative system is busy between 74 and 82 percents of the packet processing time. In the reference model, only 4 percents of concurrent arbitration is present. This results from the fact, that request of shared resources is more relaxed. Utilization of the shared bus is about 19 to 27 percents which results from occurring mispredictions. For pseudo-parallel processing, it is only 7 percent.

## 5 Conclusions

We have presented simulation results, stimulated by real network traffic, which demonstrate the benefit of the proposed speculative processing concept. Processing delay can be reduced up to 26.3 percent without drawbacks on system throughput. Further work will concentrate on other prediction algorithms as well as on optimized egress scheduling algorithms.

## References


1. M. Bjoerkman, P. Gunningberg, *Locking Effects in Multiprocessor Implementations of Protocols* ACM Sigcomm 93, Sep. 1993.
2. D.D. Clark, *Modularity and Efficiency in Protocol Implementation* NIC RFC 817, 1982.
3. J.P.G. Sterbenz, *Protocols for High-Speed Networks: A Brief Retrospective Survey of High-Speed Networking Research* IFIP/IEEE Workshop on Protocols for High Speed Networks, May 2002.
4. S. Lakshmanamurthy, K. Liu, Y. Pun, L. Huston, U. Naik, *Network Processor Performance Analysis Methodology* Intel Technical Journal, Vol. 6, Aug. 2002.
5. T. Vaseeharan, M. Maheswaran, *Towards a Novel Architecture for Wide-Area Data Caching and Replication* Interna-

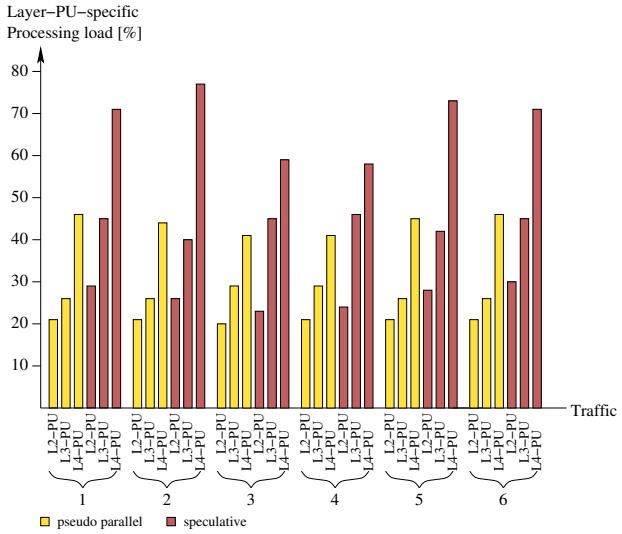


**Figure 12.** Individual Processing Resource Load

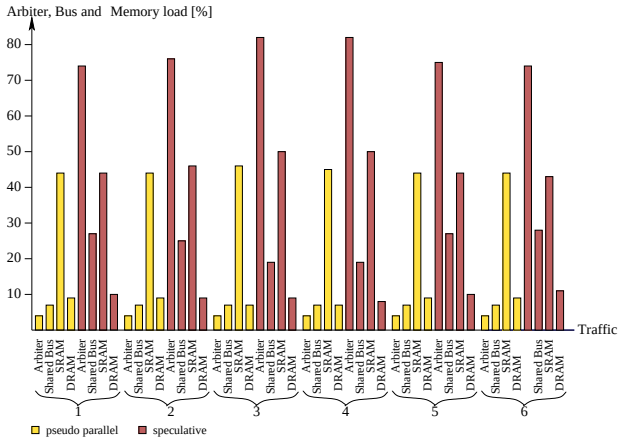


**Figure 13.** Arbitration- / Bus- and Memory-Load


tional Conference on Internet Computing, Las Vegas, Jun. 2000.

6. J. Foag, N. Pazos, T. Wild, W. Brunnbauer, *Self-adaptive Parallel Processing Architecture for High-Speed Networking* Proc. 16th Annual International Symposium on High Performance Computing Systems and Applications, Jun., 2002.
7. D. Clark, *Protocol Design and Performance* Proc. IEEE INFOCOM'95, Boston, Apr. 1995.
8. M. Zitterbart, B. Stiller, A. Tantawy, *A Model for Flexible High-Performance Communication Subsystems* IEEE Journal on Selected Areas in Communications 11, May 1993.
9. M. Heddes, E. Ruetsche, *A Survey of Parallelism in Communication Subsystems* IBM Zurich Research Laboratory, Res. Rep. RZ 2570, 1994.
10. D.C. Schmidt, T. Suda, *The Performance of Alternative Threading Architectures for Parallel Communication Subsystems* Submitted to Journal of Parallel and Distributed Processing, 1996.
11. D.C. Schmidt, T. Suda, *Adaptive: A Framework for Experimenting with High-Performance Transport System Process Architectures* International Conference on Computer Communication Networks, San Diego, Jun. 1993.
12. J. Hennessy, D. Patterson, *Computer Architecture, A Quantitative Approach* 2nd Edition, Morgan Kaufmann Publishers, Inc., San Francisco, 1996.
13. S. Storino, A. Aipperspach, J. Borkenhagen, S. Levenstein, *A commercial multithreaded RISC processor* International Solid State Circuits Conference, Feb. 1998.
14. R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Poerterfield, B. Smith, *The Tera computer system* International Conference on Supercomputing, Jun. 1990.
15. D. Tullsen, S. Eggers, H. Levy, *Simultaneous Multithreading: Maximizing On-Chip Parallelism* Proc. of 22nd Annual International Symposium on Computer Architecture, Santa Kargherita Ligure, Jun. 1995.
16. V. Jacobson, *Congestion Avoidance and Control* ACM Computer Communication Review, Oct. 1988.
17. C.M. Woodside, K. Ravinadran, R.G. Franks, *The Protocol Bypass Concept for High Speed OSI Data Transfer* IFIP Workshop on Protocols for High Speed Networks, Nov. 1990.
18. D. Patterson, J. Hennessy, *Computer Organization and Design: The Hardware/Software Interface* Morgan Kaufmann Publishers, 1994.
19. D. Schmidt, T. Suda, *Measuring the Perfomance of Parallel Message-based Process Architectures* IEEE INFOCOM, Boston, Apr. 1995.
20. M. Kaiserswerth, *The Parallel Protocol Engine* IEEE/ACM Transactions on Networking, 1993.
21. S. Wallace, B. Clader, D. Tullsen, *Threaded Multiple Path Execution* 25th International Symposium on Computer Architecture, Jun. 1998.
22. K. Ravindran, G. Singh, C. Woodside, *Architectural Concepts in Implementation of End-system Protocols for High Performance Communications* IEEE International Conference on Network Protocols, Oct. 1996.
23. W. R. Stevens, *TCP/IP Illustrated, Volume 1* Addison-Wesley, Logman Inc., 1999.
24. *ISO 7498. Information Processing Systems - Open Systems Interconnection - Basic Reference Model* International Standard, ISO, 1984.
25. *http://www.systemc.org*
26. K. Claffy, G. Miller, K. Thompson, *The nature of the beast: recent traffic measurements from an Internet backbone* Proc. of INET 98.