A Proposal for a High-Performance Active Hardware Architecture

Tilman Wolf

WUCS-99-08

February 15, 1999

A Proposal for a High-Performance Active Hardware Architecture

Tilman Wolf

Department of Computer Science
Washington University
One Brookings Drive
St. Louis, MO 63130
wolf@ccrc.wustl.edu

February 15, 1999

Current research in Active Networking is focused on developing software architectures and defining functionality of Execution Environments. While active network systems show superior functionality compared to traditional networks, they only operate at substantially lower link speeds. To increase the acceptance of Active Networks in environments where link speeds of several Gb/s are common, we propose a hardware architecture that performs high-speed packet handling while providing the same flexibility as a common software system. The design exploits the independence between data streams for parallel processing. To measure the impact of different design decisions on the performance of the system, we also propose a benchmark for Active Network components. This benchmark can be used for many Active Network architectures and can help to standardize performance results.

Introduction

Active networking is a new approach to generalize the capability of networks. By executing end user specified programs within the network, data packets can be handled in a customized way. This allows the easy introduction of new network layer protocols and applications without changing components of the network infrastructure. [1]

To implement an active network, traditional network routers have to be augmented by the capability to execute customized code, which goes beyond the common header processing and signaling. Currently, this is achieved by using workstations as routers in the experimental ABONE [3]. The processing power of the general-purpose workstation CPU is shared among the operating system, the Execution Environment, and the processing of active packets. The limited computational power of the workstation restricts the data traffic to a few Mb/s [6]. This data rate is several orders of magnitude below the bandwidth of new data networks, which operate in the Gb/s range. For the evolving field of active networks it is crucial to be competitive with respect to the end-to-end performance of current networks. This can be achieved partially by providing superior functionality. However, the throughput of an active network system also has to be increased to the range of Gb/s, otherwise the acceptance of active networks for real-world applications might be low.

There is an approach that makes use of specialized hardware in form of an FPGA [4]. However, the complex programming and the incompatibility with software that is developed for Active Execution Environments (EE) limit the widespread use.

Here we propose a hardware architecture for a high-performance Active Network Processing Element (ANPE) that can operate at link speeds in the range of Gb/s. This ANPE can be used on line cards of active routers in combination with a traditional high-performance switching fabric. Also, the ANPE can be used on network interface cards of workstations to reduce the load on the workstation CPU caused by processing active network traffic. The proposed hardware is designed to fit on a single ASIC. Such system-on-a-chip technology allows for the necessary high performance of all components to support high-bandwidth data traffic. The design is general enough to be used for different Execution Environment to make use of the most recent developments in Active Network software systems.

In order to measure the performance of this system and compare it to other active network architectures, we propose a benchmark for active network components. This benchmark defines several typical Active Applications (AA) which cover a wide range of computational complexity and bandwidth, a suggested test environment, and measurement procedures.

The following section describes a hardware architecture that makes use of the parallel nature of data streams to achieve high performance. Its functionality is illustrated, and concurrency and performance issues are described. The Active Processing Unit (APU) which provides the Execution Environment for the processing of the active packet is explained.

The last section describes the proposed benchmark. The characteristics of Active Applications are described and used to determine a set of benchmark applications and a test environment.

Hardware Architecture

Our proposal for an Active Network Processing Element in hardware is based on current developments in ASIC technology. It is possible to combine flow classification, routing lookup, active processing, and scheduling in a single design. We show that parallel processing can be used to achieve high performance while keeping concurrency issues minimal.

System-on-a-chip technology

The levels of integration and chip sizes have increased over the last years to the point where it is possible to combine a processor and memory on a single ASIC. This system-on-a-chip technology provides an easy way to build an ANPE with high computational power that is specialized on processing active network traffic.

The conventional measure for the size of an ASIC is the number of logic gates or transistors on a chip. Figure 1 shows some real world examples obtained from press releases of several ASIC manufacturers. It shows that the number of logic gates approximately doubles each year. This estimate is a little bit more optimistic than predicted my Moore's Law, which expects the size to double every 18 months.

With an increasing level of integration, not only the size of ASICs increase, but also the clock speeds at which they are driven. Current clock rates can range up to 500 MHz.

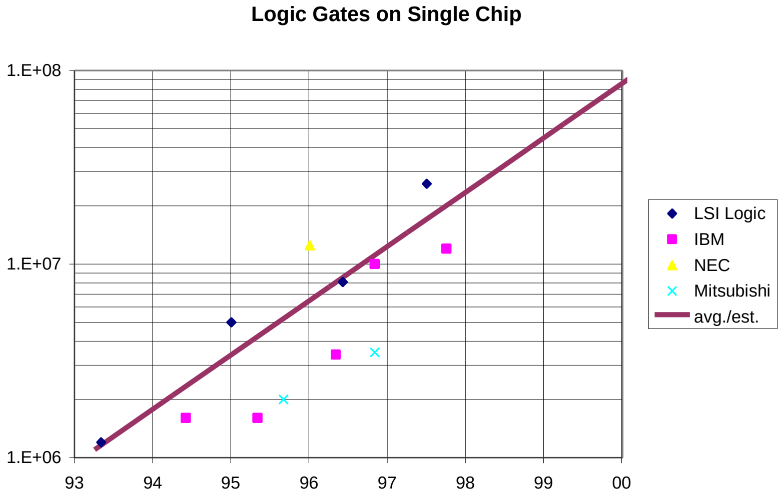

Fig. 1. Development in ASIC technology. The graph shows the number of logic gates that can be implemented on an ASIC. The increase is caused by larger die sizes as well as smaller feature sizes.

As a direct result of improvements in hardware technology, the complexity of state-of-the-art processors increases, too. Figure 2 shows the transistor count of several lines of CPUs. The transistor count relates to the gate count of the ASIC 10:1. It shows that the CPU size doubles every two years, which is a smaller rate than the growth of ASICs.

While today it is only possible to combine one or two processors with memory and control logic, one can expect that in a few years up to a dozen CPUs and memory can be put on a single ASIC. This assumption is the basis for our hardware design.

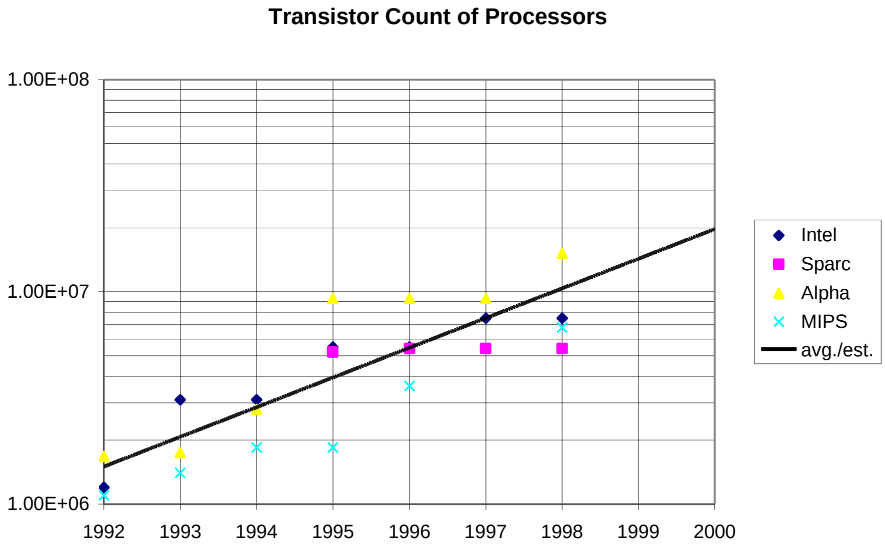

Fig. 2. Development in CPU size. The graph shows the size of state-of-the art CPUs. The transistor count relates roughly 10:1 with the logic gate count from Figure 1.

System Overview

We propose to build an ASIC that contains all functionality required for the processing of active data traffic. The ASIC is designed to be easily used on line cards of common routers and on network interface cards of workstations to upgrade traditional network equipment to handle active data traffic. Figure 3 shows a system outline.

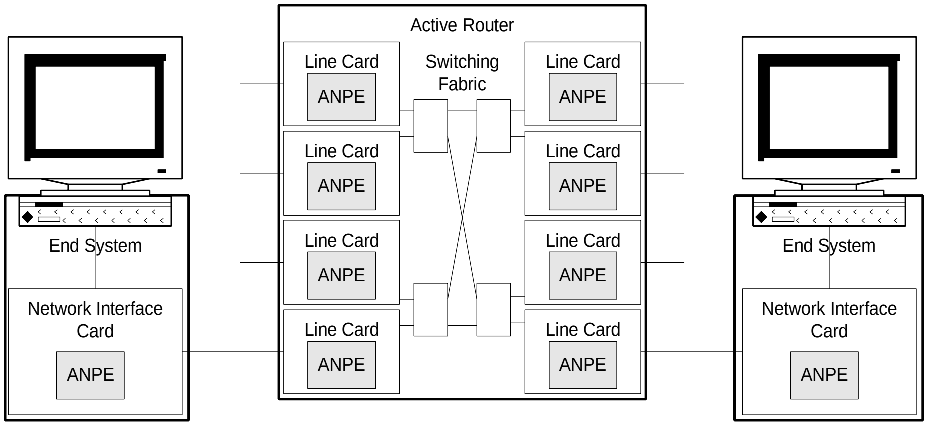

Fig. 3. System outline for an Active Network using Active Network Processing Elements (ANPE). The main application for the ANPE is on line cards of routers where it processes and forwards active data packets. However, the ANPE can also be used on network interface cards of workstations where it decreases the load on the workstation CPU by handling the processing of active data traffic.

Functional Design

One goal of the ANPE design is to exploit the parallel nature of independent active data streams. In most cases, data of one stream is independent from data of other streams. This leads to the observation that, for most traffic, there is only a dependency between the packets of a particular stream. Therefore, packets need to be processed in order to assure that state information is accessible and consistent for all packets of a stream. The lack of inter-stream dependencies makes it possible to process several data streams in parallel without the need of concurrency control.

Another observation is that a router handles active data traffic in three basic steps:

- *Routing and flow classification*, which identifies the source and destination of a packet. This is required to make the state information of the flow accessible to that packet. Routing is not necessary when the ANPE is used on end systems.
- *Active processing*, which executes the custom program code for that data packet.
- *Scheduling*, which buffers the outgoing packet until it can be sent on the outgoing link.

These three stages can be pipelined to increase performance since there is no interdependence between later stages and earlier ones. The functional design is shown in Figure 4.

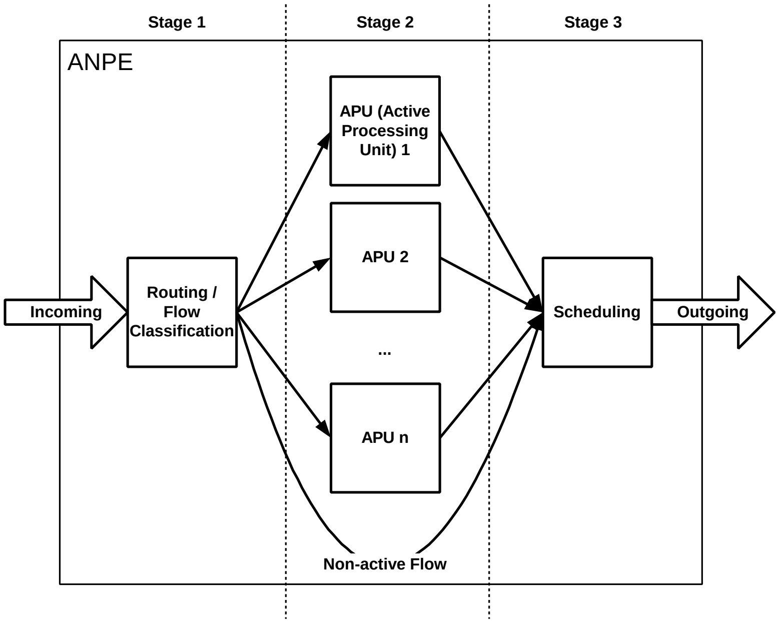

Fig. 4. Data flow in an Active Network Processing Element. In stage 1 a routing lookup is done and the packet is forwarded to the appropriate Active Processing Unit (APU). All packets of an active flow are handled by the same APU. The execution of the custom code is done in stage 2 by the APU. All outgoing packets go through the scheduler in stage 3. Traditional non-active traffic is not processed by an APU but goes directly to the scheduler.

For the execution of custom code on the active packet an Active Processing Unit (APU) is used. This APU executes one or more Execution Environments to provide the functionality necessary to handle the active packet. In practice the APU can either be a general-purpose processor or a processor specialized for the needs of active processing. There is a by-pass around the APU stage, which is taken by non-active network traffic.

Ideally, packets that belong to the same data stream are handled by the same APU. This way it can be guaranteed that packets are processed in order and the state information associated with the stream is kept consistent.

Having several APUs available for active processing makes it easy to support several Execution Environments (EE). Each APU can possibly run a different EE. Packets that require a specific EE are then forwarded to the appropriate APU.

ANPE Design

To implement the above scheme we plan to use the layout shown in Figure 5.

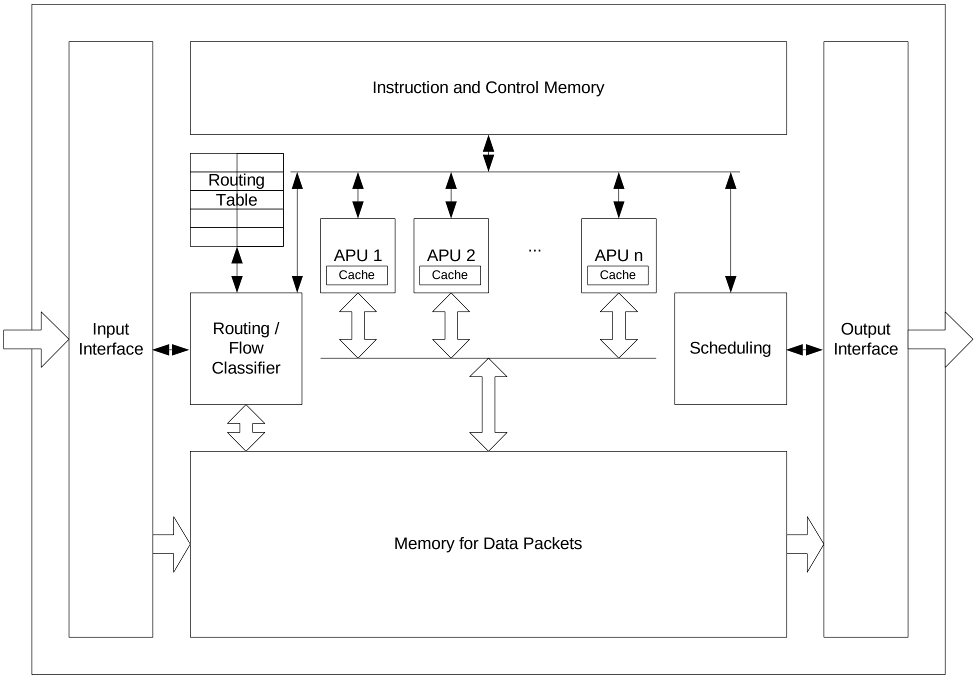

Fig. 5. Layout of the Active Network Processing Element. The Flow Classifier, the Active Processing Unit, and the Scheduler process data packets, which are stored in shared memory. The data path is shown in broad arrows, the control path in small arrows.

Data Flow

To illustrate the interactions between the different components of the ANPE this section describes the processing of an active data packet.

The general data flow is from left to right. Incoming data is received by the input interface, which stores the packet in the memory for data packets. This memory is a shared memory to which the input interface, the flow classifier, the APUs, and the scheduler have access. Once a packet is stored in the data memory it is not moved anymore. This way only a pointer to the packet has to be passed between the different processing stages instead of the complete packet, which greatly reduces the internal bandwidth of the ANPE.

The Flow Classifier reads the header of an incoming packet, determines the stream to which the packet belongs, and does a routing lookup to decide the outgoing interface. Then it passes the packet to the appropriate APU for processing.

The APU requires the custom code that has to be executed on the packet. If the packet is a capsule, it carries the code within it and can be executed easily. If the packet belongs to a programmable switch stream, it contains only a reference to the code. In this case the code will be loaded from the instruction and control memory. To load the code onto the ANPE a code server can be used as described in [5].

Modifications to the data packet are done directly in the shared data memory. Adding a data cache to the APU makes repeated access to a packet more efficient.

After the active processing, the scheduler is informed that the packet can be sent out. The output interface is responsible for the actual transmission of the packet. Once the packet has left the ANPE its data memory is reused for the next packet.

Concurrency Issues

The usage of shared memory raises issues of how shared access and concurrency are handled. Our design includes two shared memories: the instruction memory and the data memory.

The instruction memory is shared among the APUs. Each APU accesses it to read program code. The only writing access to the instruction memory occurs when additional code for packet processing is added. This does not happen very frequently, and a simple scheme to avoid possible conflicts is to block read access for that moment. We are investigating to find out if a more sophisticated scheme is required.

Conflicts on the data memory are avoided by assuring that no two components access the same memory location at the same time. Since a memory location corresponds to a data packet, only one processing component can have access to a packet at any time. This condition can be satisfied because the packet processing involves three steps: classification, processing, and scheduling. However, none of these steps occur in parallel for one packet. Additionally, the processing of a packet is done by only one APU, so there is no parallel processing on a single packet and no conflicts occur in the data memory.

Performance Issues

There are several components in the ANPE that have to be carefully designed to avoid bottlenecks that limit the throughput. In this section we give a brief reasoning of why the ANPE could handle high-speed data traffic. As an example we assume a link speed of 1.2 Gb/s (OC-12):

- *I/O performance*: Every incoming data packet is written once into the data memory, read by the APU (and locally cached), possibly written back, and read by the output interface. Therefore every data byte crosses the memory bus at most four times. This requires 4.8 Gb/s of I/O-bandwidth, which results in a bus clock rate of 75 MHz for a 64 bit wide bus. This can be implemented with current technology.
- *Computational power*: Assuming a small configuration with 5 parallel APUs with a clock rate of 400 MHz and 1 cycle per instruction, the ANPE can execute 2000 million instructions per second for active processing. With the given data rate this gives about 16 instructions per byte of data.
- *Data memory size*: The size of the data memory determines how long a packet can stay in the ANPE. A memory size of 1 MB can hold 6 milliseconds of link speed data traffic. During this time, a routing lookup, the active processing, and the scheduling have to take place.

Active Processing Unit

The Active Processing Unit (APU) is the heart of the ANPE, as it executes the customized code that is used to handle each packet. The functionality of the APU also determines the capabilities of the ANPE. Additionally, the overall performance depends on the performance of the APU. Thus, The APU has to provide enough computational power to execute the code for every packet in the time allotted for that packet.

There are two basic approaches for the implementation of an APU. In the first, the APU can be a standard workstation CPU. To provide the functionality for the EE it runs a standard OS. This approach is simple to implement, because current CPU designs and common OS software can be used.

Another approach is to use a simpler, specialized processing unit that provides just enough functionality for an EE. Complex or rare instructions, like floating point operations, are not really necessary for basic active processing and can be left out or implemented in software. Since the hardware is specialized, only a small OS is required for the EE to run on. This approach can achieve better performance since the processing unit and the operating systems are tailored for active processing. However, it is also more complicated since it requires the design of a new processing unit and a corresponding OS.

Design Issues to Achieve Good Overall Performance

There are several aspects that are important to build an Active Network Processing Element with a good overall performance:

- *functionality*
- *performance*
- *scalability*

The following sections explain the reason behind each aspect and how good overall performance can be achieved.

Functionality

The NodeOS[2] and the Execution Environment determine the functionality of an ANPE that is exposed to active packets. Typically, there is a tradeoff between additional features and performance. There are some basic features that are characteristics of an active network that have to be supported:

- *Execution of custom code* that comes with the data packet either as the code directly (capsule approach) or as a reference (programmable switch approach).
- Supporting the *concept of streams* for packets that go from the same source address / port number to the same destination address / port number (or from one source to several destinations in case of data-splitting, like multicast, or vice versa for data merging). This is important in order to be able to leave stream dependent state information for subsequent packets.
- Providing *basic functions* that can be requested by the executed code, like access to routing information, queue lengths and load of the processing unit. These are

important in order to make smart decisions by the code that comes with the packet.

- *Resource Control* for memory and processor usage.
- *Load Balancing* for systems with several processing units or systems with several ANPEs.

Performance

In order to show good performance, an ANPE has to provide sufficient computational power for the active packets that it processes. Certainly it is possible to design custom code that overloads any active network system. The idea in defining 'sufficient' is to find among all typical and reasonable application the one that requires the most computational power. If a system can accommodate data traffic of this application at full link speed, one can safely assume that there is enough computational power to accommodate any other reasonable active applications.

A System with lower computational power is still very useful for most applications, though the amount of traffic of the most complex application it can handle without getting overloaded must be specified.

Another question is how a system handles the case when it runs short on computational power. Possible solutions are not to process all packets, to do admission control and deny access for flows that would overload the system, or to perform load balancing between several systems.

Scalability

One major feature that assures the usefulness of an Active Network System is scalability. Currently, most aspects of active networking are still developing, and implemented systems consist of only a few workstations. Ideally, an ANPE has to be useful for large routers, too. Putting ANPEs on line cards makes it possible to use it on any size router without problems caused by poor scalability.

The ANPE must also scale well with respect to the number of Execution Environments that might be required to handle active packets. Having multiple APUs that can handle several Execution Environments simultaneously should give flexibility with the number of supported EEs.

Benchmark

Currently it is very difficult to compare the performance of different active network systems. A major reason for this is the wide spectrum of functionality that these systems provide. Some systems put a higher emphasis on performance than others. Starting with the choice between the programmable switch approach and the capsule approach, one can implement Execution Environments with different levels of sophistication. Another hurdle for direct comparison is that most performance results are based on a single application that was used for a particular system.

In order to allow an objective comparison between active network systems we propose a benchmark for active networks. This was also suggested by the DARPA

ITO Active Networks Working Group. With a standardized benchmark it is possible to compare the system performance of network components as well as end-to-end system performance. It also allows measuring the tradeoff between extended functionality and performance.

It is very difficult to program one benchmark suite and compile it for all existing systems, since current active network systems are not standardized to one common Execution Environment. Therefore we propose a set of benchmark applications with clearly defined functionality. The implementation of these applications is left to the developers of the systems. That way it is also possible that developers can make use of special features of their system and show its performance.

The proposed benchmark consists of

- a *set of active applications* that covers a wide range of computational complexity, bandwidth and functionality,
- a *test environment* that can show the system performance under a variety of loads and its scalability, and
- a *set of methods* according to which the test are performed.

The following sections specify these three components in detail.

Benchmark Applications

Two criteria are important for an application to be useful as a benchmark application. First, it has to be an Active Application and make use of the functionality provided by Active Networks. This way, one can see how a system performs when it does custom processing on a per packet basis. Applications that do not make use of processing on network nodes can be used for measuring how a system handles traditional network traffic, but this is not the main goal of the benchmark.

Second, the computational complexity application should be clearly defined and ideally constant. This can be achieved by using stream applications. Active stream applications are characterized by continuous data flow, usually low dependence between packets, and no interactions with other streams. Examples for such stream applications are data encryption, media transcoding, and automatic protocol deployment. Non-stream applications are typically characterized by event-driven data transmission, like sending acks on a packet arrival, data merging and data splitting, and no constant bandwidth. Examples of these applications are network management applications, reliable multicast, sensor data fusion, and web caching. The variations in bandwidth due to event-driven transmissions make this category difficult to use for a benchmark. Also, these applications often require very complex functionality, like packet merging. This functionality might not be supported by all active network systems. Therefore, we only use reliable multicast as one application from this category and concentrate the benchmark on stream applications.

The stream applications we use for the benchmark should cover the full range of possible computational complexity, from simple packet forwarding to highly complex data manipulations. In addition, it is helpful to have stream applications, which are variable in their bandwidth. Figure 6 shows a diagram illustrating the computational complexity and bandwidth of active applications.

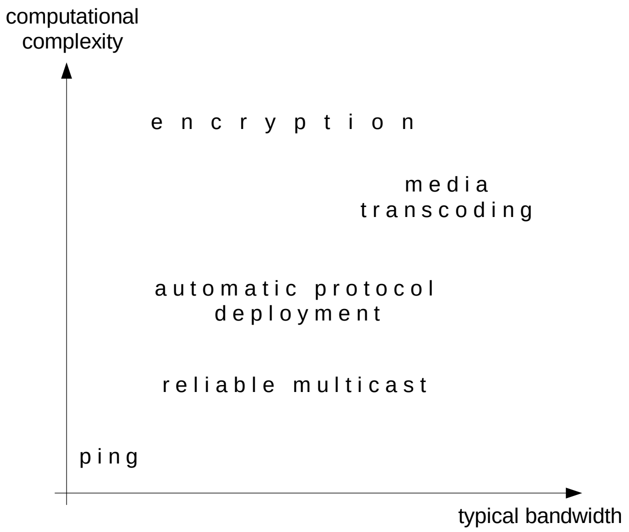

Fig. 6. Bandwidth and computational complexity of typical Active Applications. This graph shows the applications used for the benchmark and their relation to each other.

We selected the following Active Applications for the benchmark for the reasons given below:

- *Ping* (transmission of a simple data packet back and forth between two nodes): It is a very simple application that can be executed on any active network system. It shows the minimum delay for a system, because the packet has to go through the complete data path, but it does not require any complex computation. The ping application can be adjusted to a variety of packet sizes and stream bandwidths.
- *Encryption* (transmission of a data stream that is being encoded on the fly at the active network node): This application requires great computational power, because most encryption algorithms are fairly complex. The encryption application is expected to show the limits of most systems with respect to the computational power that they can provide. Encryption can be used for streams of all bandwidths.
- *Media Transcoding* (transforming of a media stream from one format to another format, typically combined with change of compression ratio and quality): This application requires not only computational power, but also might need floating point operations (e.g. JPEG decoding / encoding). Depending on whether floating-point operations are implemented in software or hardware, there can be significant differences between systems. Media data streams are typically of higher bandwidth.
- *Reliable Multicast* (transmission of a point to multi-point data stream in a reliable fashion): This application makes use of state information that can be stored on an active network node. Furthermore, it is an application that is event driven and requires the storage of a packet over a possibly longer period of time.

- *Automatic Protocol Deployment* (on demand processing of data packets that belong to a previously uninstalled protocol): This application shows how well an active system can add new functionality into the data path processing. If there is a general concept of dynamically adding protocols to a system, the performance can be significantly better than treating a new protocol as a special case.
- Regular non-active *IP flow* (transmission of common IP data packets): This application shows how a system can handle traditional IP packets. This is important to consider, because the active traffic and IP traffic will coexist in future systems.

Test Environment

The goal of the benchmark is to test the performance of an active network component. Therefore the basic configuration is to test the component by itself. Figure 7 shows a system outline for such a test. Additional tests can include testing the component on a line card in a router, in a real network environment, or on a testbed with several other active network nodes. For all those configurations this basic testing scheme can be used.

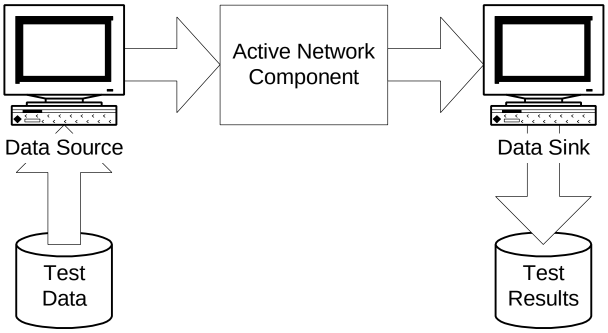

Fig. 7. Test environment for benchmark measurements. The data packets are stored locally on the workstation that acts as data source. The data sink stores all packets that are sent out by the ANPE. These results can be compared offline to the expected results to measure throughput, packet loss, delay and correctness.

The data traffic is generated by a workstation that sends packets from local storage to the active network component. The data is generated offline to assure that complex data generation programs do not delay the data transmission, and that reproducible results are being generated. This is especially important when high bandwidth data traffic has to be simulated.

The output of the active network component is stored on a workstation that acts as a data sink. The results can then be compared offline to the expected results. This way the throughput, packet loss, and correctness can be determined.

Testing Methods

Using the above described test environment, it is possible to determine the following characteristics of the ANPE:

- *correctness* of active processing
- *throughput and packet loss*
- *packet delay*

By varying the set of Active Applications, the stream bandwidth, the packet size, and the combination of streams, the following relations can be established. They help determine the performance of an ANPE. Possible bottlenecks can be found and eliminated.

- *Throughput and packet loss for different Active Applications*. If throughput decreases and packet loss increases with the complexity of the Application, then the computational power of the ANPE is insufficient. If the packet loss is independent from the Active Application, then there might be a bottleneck in the general network components, like routing and flow classification, or in the switching fabric.
- *Throughput and packet loss for streams with different bandwidth*. A system might show a different behavior if it handles few high bandwidth streams compared to many low bandwidth streams. If throughput decreases and packet loss increases with many low bandwidth streams, then the flow classification or the context switching on an APU might be a bottleneck. It is also possible that the packet loss increases with fewer high bandwidth streams. Fewer streams decrease the amount of parallelism, which the ANPE usually exploits. In an extreme case there might be only one stream at link rate, which would be handled by one APU with no parallel processing at all.
- *Delay and packet loss for combinations of Active Applications*. If Execution Environments without resource management are used, this might show that computationally complex applications delay the processing of other streams.

Although these questions are focused on the ANPE architecture, many similar observations can be made for most active networking components.

Conclusions and Future Work

We have proposed a hardware architecture for an Active Network Processing Element which will allow us to build network components for high bandwidth active data traffic. Using several parallel processing units on a single ASIC is an approach to increase the processing power of an ANPE. It deviates from the common direction of computer architecture that aims at building more and more complex single processor systems.

The use of specialized hardware for active processing opens the interesting question if different aspects of active functionality should be implemented in the hardware, the Node OS, or the Execution Environment of a system.

The implementation of a benchmark for active networks is an immediate goal. With the use of such a benchmark it will be possible to quantitative comparisons and relate different active architectures to each other.

Our future research is focused on defining the functionality of a simple APU in order to simulate it and get a performance estimate. The results from current work on defining a NodeOS will help to provide basic functions for efficient executions of EEs.

Another direction of our future research is to investigate in resource management scheme for memory and CPU. Additionally, finding a flow classification scheme which can easily adapted for new protocols would improve the ability to use the ANPE for rapid protocol deployment applications.

Bibliography

[1] Tennenhouse, D. L., Wetherall, D. J. (1996): "Towards an Active Network Architecture". Computer and Communications Review, Vol. 26, No. 2, April 1996.

[2] Peterson, L. (ed.): "NodeOS Interface Specification". AN Node OS Working Group, October 1998.

[3] Braden, B., Ricciulli, L.: "A Plan for a Scalable ABone – A Modest proposal". ftp://ftp.isi.edu/pub/braden/ActiveNets/ABone.whpaper.ps, January 1999.

[4] Hadzic, I., Smith, J.: "On-the-fly Programmable Hardware for Networks". Distributed Systems Laboratory, University of Pennsylvania, 1998.

[5] Decasper, D., Plattner, B.: "DAN - Distributed Code Caching for Active Networks". Proceedings of IEEE INFOCOM'98, San Francisco, April 1998.

[6] Wetherall, D., Guttag, J., Tennenhouse, D.: "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols". Proceedings of IEEE OPENARCH'98, San Francisco, April 1998.