

A VLAN Ethernet Backplane for Distributed Network Systems

Lei Shi *

Institute for Informatics
Göttingen University, Germany
shi@informatik.uni-goettingen.de

Peter Sjödin

School of Electrical Engineering
KTH—Royal Institute of Technology
SE-100 44, Stockholm, Sweden
psj@kth.se

Abstract—In a network system, such as a router or a switch, it is difficult to achieve flexibility and performance at the same time. We propose an architecture that consists of network processors for packet processing and a VLAN-based Ethernet backplane for switching. This allows us to use flexible network processors for packet processing functions, and still exploit the cost-effectiveness of Ethernet to achieve switching capacity. We propose an architecture where we use VLAN tagging for internal traffic management, and also for distributed packet forwarding decisions between ingress and egress units. We describe our implementation of this system and report performance analysis, where we find that we can achieve near line rate performance in a system with Gigabit Ethernet ports, and that internal memory management is important for network processor performance.

Index Terms—network processor, parallel processing, distributed router, performance evaluation

I. INTRODUCTION

NETWORKS are becoming more and more important in our daily lives, and our requirements on the systems in the network are increasing. We invent new ways of using networks, and at the same time we wish to integrate telephony, television and data into one network. The implication for network system architectures (routers, switches, servers, etc) is that they need to be more dynamic and flexible to satisfy the demands for new services and functionality, and at the same time fulfill the ever-increasing demands for capacity.

Capacity and flexibility are to some extent each other's opposites. Flexibility comes from programmability, but programmable units such as general purpose CPUs are slow. Capacity is achieved by using ASIC (Application Specific Integrated Circuits) and similar technologies—devices which are inherently static and inflexible.

In this work we investigate how the problem of achieving flexibility and capacity can be addressed in a diversified network system architecture that combines different types of technologies. Flexibility and programmability come from network processors (Intel IXP2400), which take care of packet processing (IPv4 in this case). The network processor

technology offers the advantage of being software-programmable and sufficient high-speed to accommodate interfaces even running at 40 Gbps today. Each network processor deals with only a fraction of the total traffic, so the processing burden, and thereby the capacity requirements, of each network processor are kept down. With an architecture that can support a large number of network processors, the system can scale by adding more and more network processors.

Network processors are interconnected by standard Ethernet switches, which provide switching capacity. Ethernet is still the most-cost effective way of providing switching capacity in a network, so by using Ethernet as interconnect it is possible to take advantage of Ethernet switching price-performance also for the purposes of higher-layer switching. In addition, Ethernet is attracting more and more interest as an alternative technology for intra-system interconnects [10][11].

The resulting architecture is illustrated in Fig. 1. This is based on the terminology from ForCES—the IETF working group for separation of forwarding and control [3]. Ordinary PCs are used as Control Elements (CEs), running standard software for routing, network management, etc. The network processors are Forwarding Elements (FEs). FEs are interconnected by an *internal data network*, and CEs and FEs communicate with each other over an *internal control network*. In our case, both internal networks are based on Ethernet.

The requirements of the internal network are somewhat different from those of regular networks. For instance, control and data need to be separated, so that heavy traffic load does not starve out internal control traffic and thereby prevent the correct operation of the system. In addition, in order for the network system to support different types of traffic, the internal network should be designed to provide appropriate services for this.

A further requirement is related to the internal operation of the system. In our case the system is a router, so this means that IPv4 classification and forwarding decisions are made at the incoming FEs in order to determine outgoing ports. If the outgoing FE (the FE with the outgoing port) is a different FE, the packet is forwarded over the internal data network to the outgoing FE. However, the information about the next hop comes from IPv4 classification at the incoming FE, so there has to be a way to convey the result of the classification from

* This work was performed while the author was at the Laboratory for Communication Networks, KTH, Sweden. Now he is with Institute for Informatics, University of Göttingen, Germany.

incoming to outgoing FE. This is something that incurs overhead, since next hop information needs to be sent with packets over the internal data network.

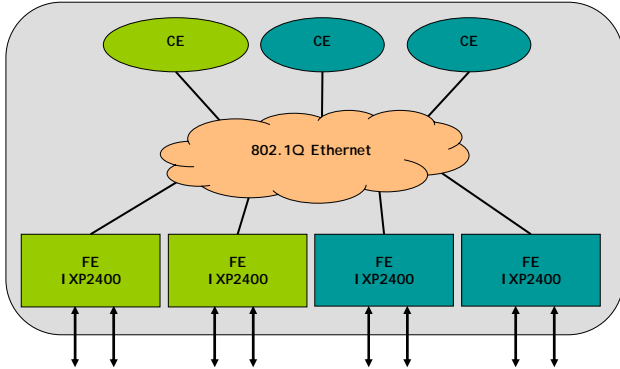


Fig. 1. Distributed router with Ethernet backplane and IXP2400 forwarding elements.

Our approach for the internal network is to base it on an IEEE 802.1Q VLAN Ethernet. This allows each packet to be tagged at the Ethernet level with a 12-bit VLAN tag. A VLAN tag represents a virtual LAN, where all frames with the same VLAN tag are switched as if they were on the same LAN, while frames on different LANs are separated. Hence, the result is that there are multiple virtual LANs over the same, shared infrastructure.

With such a VLAN-based Ethernet backplane we can achieve separation of control and data, by allocating different VLAN tags for control and data. It is also a way of supporting traffic separation on the internal data network, since different VLANs can be allocated for different types of traffic. Finally, we can use the VLAN tag to encode the next-hop information, which gives an efficient way of carrying control information from incoming to outgoing FE.

The purpose of this paper is to explore a VLAN-based backplane for a distributed network system in the form of a router, and to study the performance of network processor FEs for such a system. The outline is as follows: Section 2 deals with the VLAN-based backplane architecture and discusses the architectural requirements of the parts in the system. In Section 3, the network processor implementation of the FEs is described. The performance is analyzed in Section 4, and Section 5 discusses the results and concludes the paper.

II. VLAN BACKPLANE

Our network system is a distributed router, based on the principles of separation of control and forwarding, and with the design goal to support a heterogeneity of hardware and software modules within a single system [4][6]. The backplane consists of VLAN-capable Gigabit Ethernet switches. Resources are allocated in the backplane by setting up VLANs in a way that matches the anticipated traffic and its

requirements. The VLAN information is stored in the forwarding tables in incoming FEs, as part of the next-hop information. Forwarding tables are derived from the RIB (Routing Information Base), which is generated by the routing protocols.

There are two types of forwarding tables, ingress and egress. Ingress forwarding table are used for packets that arrive on external ports, that is, ports connected to other routers. An ingress forwarding table lookup is made through a longest prefix match operation, which will provide next hop information. Next hop information includes a VLAN label and a MAC address on the internal network, and this information is used to forward the packet in an Ethernet frame over the internal network. If the packet is destined to the router itself (the IP destination address is one of the router's addresses), the VLAN tag is for a VLAN used for internal control traffic, and the destination MAC address will be the MAC address of one of the CE's. If the packet is to be forwarded to another router, the VLAN identifies the next hop router, and the MAC address is for the FE to which the next hop router is connected.

On the egress side, the forwarding table is indexed by VLAN tags. So when a packet arrives at an FE over the internal data network, the FE uses the VLAN tag to look up the next hop information. The next hop information consists mainly of link layer information for the next hop, such as port number, MAC address, and so on.

III. NETWORK PROCESSOR FORWARDING ELEMENT

The forwarding element in our system is based on Intel's IXP network processor family. We use the IXP2400 processor [8] on Radisys ENP-2611 boards [9]. The ENP-2611 board is a PCI board with three optical Gigabit Ethernet ports and one 10/100 Ethernet port. An IXP2400 is a multiprocessor chip with one 32-bit XScale microprocessor core and eight multithreaded 32-bit RISC microengines. The XScale microprocessor is a general-purpose CPU based on the ARM architecture. It runs Linux (Montavista Linux Preview Kit for Pro 3.1), and is used for communication with CEs and for controlling the microengines. The microengines are special-purpose processors for packet processing. Each microengine has eight hardware threads, so a microengine can maintain eight contexts at the same time.

The IXP2400 has two 4 MB QDR SRAM memories and one 512 MB DDR SDRAM memory. The SRAM is a fast memory mainly used for lookups, buffer descriptors, and so on, while the SDRAM is slower, used for packet buffers and XScale instructions. In addition, there is a dedicated control memory for microengine instructions, and a small, fast scratchpad memory for general purpose storage.

One of the most delicate tasks when it comes to microengine programming is the separation of the program into a set of modules, and allocation of those modules onto microengines and threads to create a well-balanced system with good performance. The basic principle is that the software is divided into modules, which are then assigned to microengines. Within

each microengine, a number of threads are allocated to that module. The basic principle in our design, which is based on the reference design from Intel [7], is that within each module, a packet is assigned to one thread. This thread will do all processing for that packet within the module. The design consists of the following modules:

- **Packet Rx**, which receives packets from the Gigabit Ethernet MACs. The main task of Packet Rx is to reassembly packets from the smaller units (mpackets) used internally on the ENP-2611. The Packet Rx module runs on one microengine, with two threads for each Gigabit Ethernet port.
- **Packet Processing**, which handles Ethernet decapsulation and classification, and IPv4 forwarding. This is the most computation-intensive module, distributed over three microengines for load-sharing.
- **Queue Manager** takes care of packet queues to the output ports, and is responsible for enqueue/dequeue operations. It runs as eight threads on a single microengine.
- **Scheduler** is responsible for scheduling packets to output ports, so it schedules dequeue requests to Queue Manager based on a scheduling policy (in our case Round Robin). It runs as two threads on one microengine.
- **Packet Tx** takes packet from the queues, provides them with link layer framing and sends them to the Gigabit Ethernet MAC. This module runs as twelve threads; eight on one microengine and four on another microengine.

IV. PERFORMANCE EVALUATION

The first step in our performance analysis is to study the delay of the different modules, in order to study the relative performance of the different processing modules. This analysis is done through simulation. By running the microengine programs in a simulator, using packet traces as input data, we can collect information about the number of instructions per packet, and the delay per packet. Table 1 shows in the first two columns the average number of instructions performed per packet and the average packet delay (in cycles). All microengine instructions take one cycle to execute—the difference between cycles and instructions represents the time during which a thread is idle. The idle time is mainly related to waiting for SRAM and SDRAM memory operation completion, and to contention for microengines. The third column shows how many threads that have been allocated to a given module, and the last column shows the corresponding packet service interval (expressed as cycles per packet, i.e., number of cycles divided by the number of threads).

TABLE 1
INSTRUCTION AND CYCLE COUNT, THREAD NUMBER AND CYCLES PER PACKET

	Instructions	Cycles	Threads	Cycles per packet
Packet Rx	74	614	6	102
Packet Processing	298	2574	23	112
Queue Manager (Two misses)	72	1172	8	146
Queue Manager (Two hits)	66	423	8	53
Scheduler	62	84	2	42
Packet Tx	121	639	12	53

For most modules the table shows the average number of instructions and cycles it takes to process a packet. The Queue Manager, however, uses an internal CAM (content addressable memory) as a cache and does two accesses per packet to this cache. There is a large difference between the best case (two hits) and the worst case (two misses), so we chose to show the numbers for two cases. The average depends on the mix of misses and hits, which in turn depends on the locality of the traffic with respect to queuing policies.

In order to assess the results, we can compare them to the minimum packet inter-arrival time, which is 224 nanoseconds (corresponding to 134 cycles). If we consider this as the upper limit for the service interval, we can see that Packet Rx, Packet Processing, Scheduler and Packet Tx should be capable of processing packets at line rate, with a fair margin. Queue Manager, on the other hand, strongly depends on the memory behavior and hence appears to be a potential limiting factor in the system.

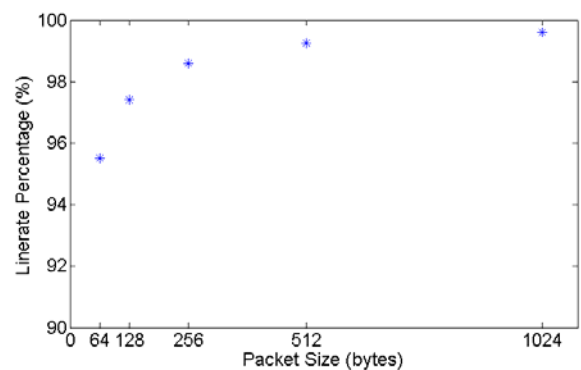


Fig. 2. Highest load at which no packet losses occur (as percentage of line rate)

The next step in our analysis is to measure the forwarding performance of our IXP2400-based FEs using a traffic generator. We measure a system with two FEs, with traffic flowing through the two FEs in both directions. The performance measurements are conducted for different packet sizes (64, 128, 256, 512, 1024 bytes). For each packet size, the load is increased to the point where packet loss occurs. The

highest load that could be achieved without packet loss is shown as a function of packet size in Fig. 2. The diagram indicates that the capacity of the system as a whole matches the speed of the links, but that there is a small amount of packet loss. The packet loss can be explained mainly by the difference in packet sizes: the VLAN tag adds four bytes to packets on the internal links, which means that the maximum theoretical throughput is less than 100%.

TABLE 2
PACKET DELAY IN MICROSECONDS FOR DIFFERENT PACKET SIZES

	64 byte	128 byte	256 byte	512 byte	1024 byte
IPv4 to VLAN	13.4	15.0	16.7	20.5	27.1
VLAN to IPv4	9.3	10.1	13.8	15.2	25.3
Total	22.8	25.5	30.4	35.9	52.6

In order to study the relationship between ingress and egress processing, we measure the delay through the system for different packet sizes, as shown in Table 2. The delay for ingress processing is shown (IPv4 to VLAN), egress processing (VLAN to IPv4) as well as the total delay through the system.

A. Discussion

The instruction and cycle count analysis show that most modules are capable of keeping up with the line rate, but that the Queue Manager may be a limiting factor. The large difference between instruction count and cycle count for the Queue Manager when there are CAM misses is an indication that it is not the processing that takes time, since the microengine is spending a significant amount of time being idle. Hence, assigning more microengines to the Queue Manager would not be a remedy. Instead one could consider modifications to the memory system that holds the queue descriptors.

Previous work has shown that the design of the forwarding decision-making can have significant impact on the performance of distributed routers [5]. From the performance measurements on our system, it can be seen that ingress processing takes slightly longer than egress. This follows from the design decision to put the lookup and classification burden entirely on the ingress FE. However, the difference is not large, and our instruction count analysis indicates that packet processing is not the main bottleneck. Hence, in our case there would be no substantial performance gain in dividing packet processing more evenly between ingress and egress.

V. SUMMARY AND CONCLUSIONS

We have described an architecture for an Ethernet-based backplane in a distributed network system. The architecture is based on IEEE 802.1Q VLAN, which is used for internal traffic engineering and for tagging packets with next hop information.

Performance analysis results have been reported from an IXP2400 network processor implementation of the ingress and egress nodes. The performance results indicate that the design and architecture are feasible for this setup, and we have shown high performance can be achieved even for small packets. The network processor being used is in the medium range, so by upgrading to another model within the same processor family it should be possible to achieve higher line rate for all packet sizes.

The VLAN architecture uses VLAN tags for two purposes: for internal traffic engineering and for tagging frames with next hop forwarding information. Since VLAN tags have limited size (12 bits), there is a concern that this overloading of tags could potentially exhaust the tag space. Even though we believe that 12 bits would be enough for many practical purposes, it would be interesting to study how IEEE 802.1ad tagging (“Q-in-Q”) [1] can be used to separate traffic engineering from next-hop tagging. With Q-in-Q, there are two levels of VLAN tags. The intended use is to allow operators to build VLAN-based access and metro networks, where the outer level of tags is used by the operator (metro tag, or “PE-VLAN”) and the inner tag is reserved for customer usage. For our design, it seems that the most straight-forward application would be to use the inner tag for next-hop encoding, and the outer tag for traffic engineering. This would have the additional advantage of separating the two functions from each other, and thereby simplifying the control plane.

REFERENCES

- [1] 802.1AD-2005 IEEE Standards for Local and metropolitan area networks—Virtual Bridged Local Area Networks—Revision—Amendment 4: Provider Bridges. IEEE Standard No 802.1AD-2005. <http://www.ieee802.org/1/pages/802.1ad.html>.
- [2] 802.1Q-2005 IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks. IEEE Standard No 802.1Q-2005. <http://www.ieee802.org/1/pages/802.1Q.html>
- [3] ForCES (Forwarding and Control Element Separation) IETF Working group, URL=<http://www.ietf.org/html.charters/forces-charter.html>.
- [4] O. Hagsand, M. Hidell, P. Sjödin, “Design and Implementation of a Distributed Router”, IEEE Symposium on Signal Processing and Information Technology (ISSPIT), Athens, Greece, December, 2005.
- [5] T. Hamano et. al., “Forwarding Model of Backplane Ethernet for Open Architecture Router, “ 2006 Workshop on High Performance Switching and Routing, Poznan, Poland, June 7 – 9, 2006.
- [6] M. Hidell, P. Sjödin, and O. Hagsand, “Control and Forwarding Plane Interaction in Distributed Routers”, in Proceedings of Networking 2005, Waterloo, Canada, May 2005.
- [7] “Intel Internet Exchange Architecture (Intel® IXA) Software Building Blocks Developers Manual,” Intel Corp.
- [8] “Intel IXP2400 Network Processor Product Brief,” Intel Corp.
- [9] ENP-2611 Data Sheet, Radisys Inc. <http://www.radisys.com>.
- [10] S. Reinemo, T. Skeie, T. Sødning, O. Lysne, and O. Tøruðbakken, “An Overview of QoS Capabilities in InfiniBand, Advanced Switching Interconnect, and Ethernet,” IEEE Communications Magazine, Vol. 44, No. 7. July 2006.
- [11] S. Vedantham, S.-H. Kim, and D. Kataria, “Carrier-Grade Ethernet Challenges for IPTV Deployment,” IEEE Communications Magazine, Vol. 44, No. 7. July 2006.