Towards the design of an high performance active node

Jean-Patrick Gelas, Saad El Hadri, Laurent Lefèvre
INRIA RESO / LIP
Ecole Normale Supérieure de Lyon
46, allée d'Italie 69364 LYON Cedex 07 - France
Jean-Patrick.Gelas@ens-lyon.fr, Saad.El.Hadri@ens-lyon.fr, Laurent.Lefevre@inria.fr

Abstract

Achieving high performance in active networks is one of the most challenging task. In this paper, we propose an architecture for the design of next generation gigabit active routers[1]*. This original architecture allows service deployment of 4 levels : inside network cards, in kernel space, in user space and on distributed computing resources. We deploy and validate this architecture within the Tamanoir execution environment. First experiments on gigabit network platforms are described.*

Keywords : high performance, active networking, execution environment, Tamanoir

1 Introduction

The integration of new and dynamic technologies into the shared network infrastructure is a challenging task, and the growing interest in the active networking field[18] might be seen as a natural consequence.

In active networking vision, routers can perform computations on user data in transit and users can modify the behavior of the network by supplying programs, called *services*, that perform these computations. These routers are called *active nodes* (or *active routers*) and propose a greater flexibility towards the deployment of new functionalities, more adapted to architecture, users and service providers requirements. Other network equipments (gateways,bridges, proxies) can benefit from active network technology

Currently active network designers must face two major problems : security of services deployment inside equipments and performance of on-the-fly processing. This paper proposes solutions for the design of an high performance active node.

For most of researchers in active networks, providing active services with high level languages (Java) and inside user space is a too costly approach due to the latency added for processing packets. This is mainly due to the fact that numerous experiments in active networks relies on ANTS toolkit [19] (based on Java) with peak performance of around 5 Mbits.

Our main goal is to focus on the providing of performance inside active network equipments. We define an architecture targeted for the design of a Gigabit active node. This active node should be able to process and route active streams coming from Gigabit networks (Fig. 2).

This layered architecture proposes solutions for the dynamic embedding of active services optimally deployed on suitable levels:

- ultra lightweight services in network programmable cards (packets marking, dropping and filtering services);
- lightweight services in kernel space level (packets counting, QoS, management services, intelligent dropping and state-based services);

[1]This work is supported by the French RNRT VTHD++ Project.

- middle services in user space level (reliable multicast, packets aggregating, packets monitoring and data caching);
- high level services in distributed architecture (compression and multimedia transcoding on-the-fly).

In order to validate the proposed architecture, we design the Tamanoir Active node software suite based on widely used equipments and tools : Myrinet for NIC level, Netfilter/Linux[16] for kernel space support, Java for user space level and Linux Virtual Server[22] for clustering approach. This Tamanoir software is deployed and experimented on various local and long distance platforms.

The paper is organized as follows. In section 2, we propose an architecture of an high performance active router. Section 3 describes the Tamanoir architecture developed in our laboratory. Section 4 presents performances obtained with Tamanoir. We briefly describe other solutions proposed for high performance active networking (section 5). We finish by some conclusions and future directions for the improvement of software active routers.

2 An High Performance Active Node architecture

We want to design an architecture of an active router able to be deployed around high performance backbone. Our approach concerns both a strategic deployment of active network functionalities around backbone in access layer networks and by providing an high performance dedicated architecture.

Our active network model is focused on active edge routers, located around the core network between backbone and access networks (Fig. 1). Core networks are mainly optical and must remain fast (40 Gb/s) and simple. Access networks must face heterogeneous equipments and protocols and could benefit from the deployment of dynamic network services.

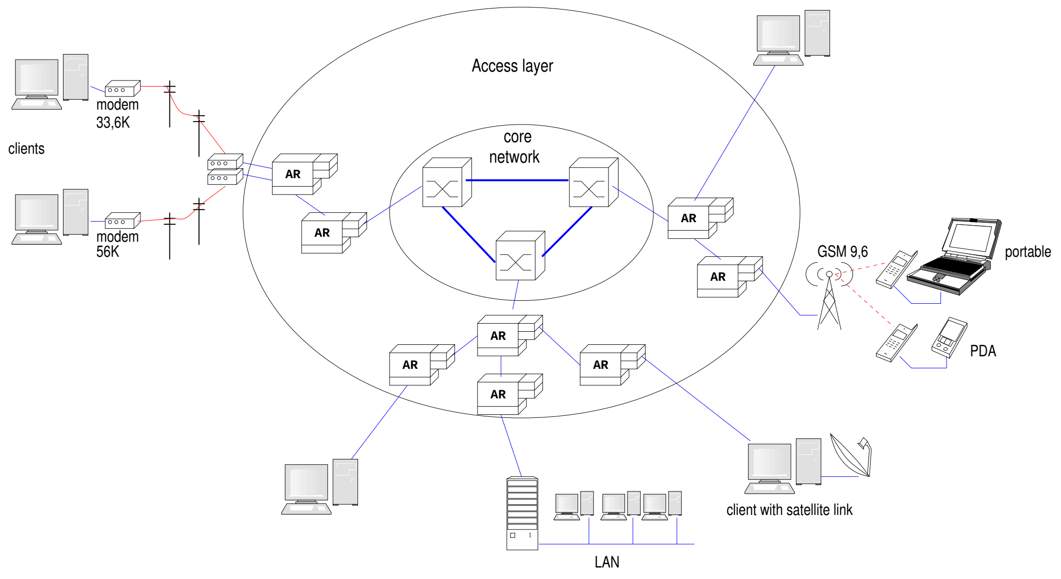

Figure 1. Active Routers (AR) deployed in access networks

We define an Active Network Execution Environment (EE) as an environment able to *load* and deploy a service in memory's execution system. It must be also able to *direct* packets towards the required service thanks to appropriate headers filtering. Ideally, an EE direct packets to the service as transparently as possible without adding overheads.

Most common protocols like TCP, UDP, ICMP, RTP, RTCP,... should be supported by an active node. While UDP or RTP are used for multimedia or real-time applications, supporting TCP streams is required for applications requiring reliable communications like file transfer, web traffic and Grid application.

By taking into account high performance challenges, active services must be deployed at various levels depending on resources (processing capabilities, memory consumption, storage capacity...) and intelligence (flexibility of the execution environment) they need. In order to provide an adapted EE for each kind of services and to limit packet overhead, we design an active node architecture on 4 levels : Network Interface card (NIC), Kernel Space, User Space and Distributed Resources (see Fig. 3).

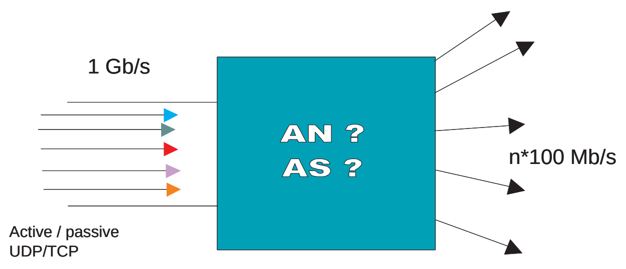

Figure 2. Active node between backbone and access network

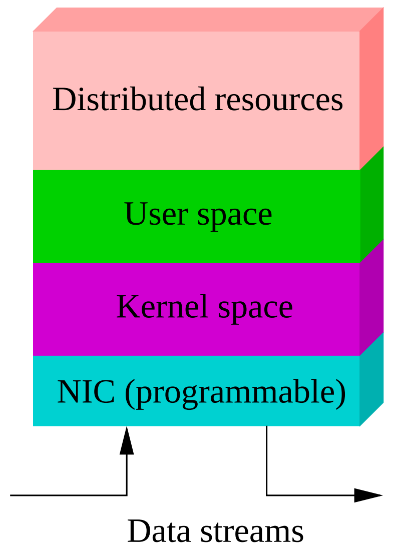

Figure 3. Execution environment of an active node architecture

2.1 Network Interface Card level

Programmable network interface cards (NIC) like Myrinet[3] embed CPU, RAM and DMA engines. In these cards, software network protocols can be executed to optimize communications between host and network. We take advantage of this flexibility to deploy low level network services on these NIC. Running services directly on programmable network interface cards gives the advantage to run services as close as possible to the wire. The idea is not so far from Network Processor. Class of services must be restricted to ultra-lightweight one : packets marking, packets dropping, packets counting. . . in order to not impact processing time per packet and NIC memory space allocation.

While this paper does not focus on services deployment inside NIC, this topic is currently under investigation in our team.

2.2 Kernel space level

In kernel space, OS runs time-sensitive operations : scheduler, protocols stacks, drivers. . . An active node can deploy, in this level, efficient lightweight services requiring memory and processing capabilities from the host. This deployment is specially useful when NIC are not programmable.

The kernel space level is perfectly suited for lightweight level services like QoS services or intelligent packets dropping. Moreover, services running at kernel space level can benefit to the the routing functionalities of the kernel and also use zero-copy or OS bypass technics to communicate with the user space. This approach requires an open kernel and easy access to the network protocol stack.

Running a service in kernel space allows a very fast execution and takes advantages of resources (fast CPU, system memory) of the host system. Services must be written in C or assembly code which limits portability and makes the writing less obvious. This approach requires obviously an open operating system (like Linux or BSD). This system must provide tools to direct active packets to kernel lightweight services (like *Netfilter* in Linux). Kernel space is a very sensible part of the system and doesn't tolerate any misbehavior. There is a risk to endanger the whole system. Services must be restricted to time-sensitive services in terms of processing time per packets.

2.3 User space level

A user space level can provide all the safety, flexibility and easiness for running a full-featured execution environment. Services executed in this level can access to all system resources (memory, disk, dedicated hardware...). It gives also the opportunity to use high level languages (like Java).

However, overhead introduced by the processing of packets on this user space level and the cost of copying data from kernel space to user space must be taken into account in order to reduce the impact on raw performances. The EE must be executed as fast as possible and then written with a constant high-performance objective in mind. It should not be interpreted during execution and then must be either compiled or use Just-In-Time compilation technics.

2.4 Distributed resources level

Active streams requiring heavy processing functions like compression, cryptography or conversion on-the-fly require heavy processing capabilities in of the active node. These services must be supported by a parallel architecture.

We explore the design of a parallel active node depending of Execution environment requirements and available architecture.

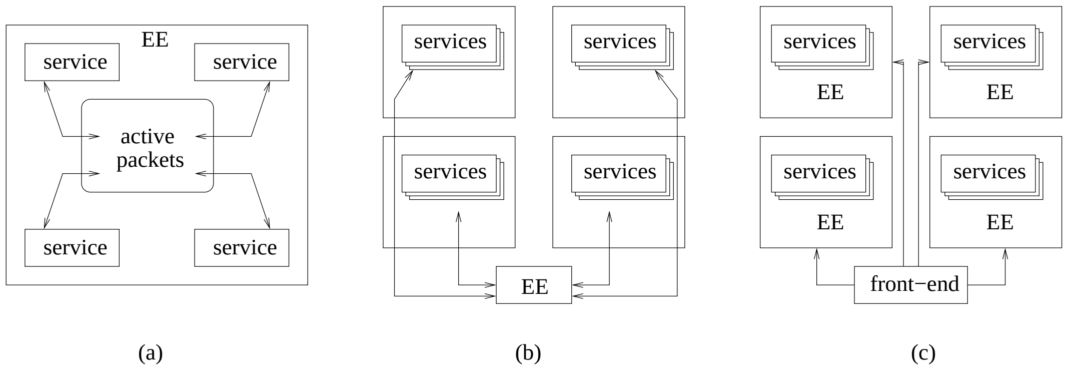

Figure 4. Approaches to design a parallel active node architecture: (a) shared memory, (b) message-passing and (c) replicated EEs

Shared memory approach : First approach consists of distributing services on various processing units (Fig. 4(a)). These services can be executed in parallel and can access to packets through a shared memory (on an SMP architecture) or a distributed shared memory (on a cluster of machines [12]). Packets reaching the active node are placed in queues located in shared memory with one queue for each available service, but a unique queue for services of same name. Services are considered as consumers of these queues. This approach easily allows services migration between processing units.

Message-passing approach : This approach is dedicated for distributed computing resources (clusters of machines) communicating through messages. Like first approach, services are distributed on various machines. The execution environment is mapped on a dedicated node. In figure 4(b) the EE receives packets and directs them towards the node holding the required service. Message passing techniques libraries (PVM, MPI...) could be used. Next, the node process the packet with the suitable service before its retransmission.

Replicated EEs : Last approach consists of replicating EE and services on distributed resources (Fig. 4(c)). We call these nodes back-ends. In order to provide to replicated EEs active packets, a front-end machine must be added to the architecture. This front-end is completely dedicated to distribute streams to the back-ends. This approach requires less modifications to Execution environments and services.

These 3 approaches (shared memory system, message passing, replicated EEs) provides parallelism in streams processing. Another advantage of using a distributed architecture like a cluster of PC concerns the fault-tolerance capability of the active node. It gives the possibility of stopping a back-end node for maintenance without stopping the whole active node and services. Moreover, it is possible to upgrade performances of an active node by adding more back-ends. In order to avoid single point of failure on EE (b) or on the front-end (c), these last ones must be replicated on various nodes.

All distributed solutions must also take into account of load balancing of services and streams (from round-robin (RR) algorithm to more sophisticated algorithm like weighted RR, least connection RR...)

3 The Tamanoir experience

The aims of the Tamanoir[2] project is to design an high performance active node validating the architecture described in section 2. The whole development of the high performance EE Tamanoir has been done in several steps. First we implemented a EE running in user space, next we investigate the kernel space and finally the distributed computing approach.

3.1 High level multi-threaded Execution Environment

The Tamanoir[7, 8] suite is a complete software environment dedicated to deploy active routers and services inside the network. Tamanoir Active Nodes (TAN) provide persistent active routers which are able to handle different applications and various data stream (audio, video,...) at the same time (multi-thread approach). The both main transport protocol TCP and UDP are supported by TAN. We rely on the ANEP (Active Network Encapsulated Protocol)[1] format to send data over the active network (Figure 5).

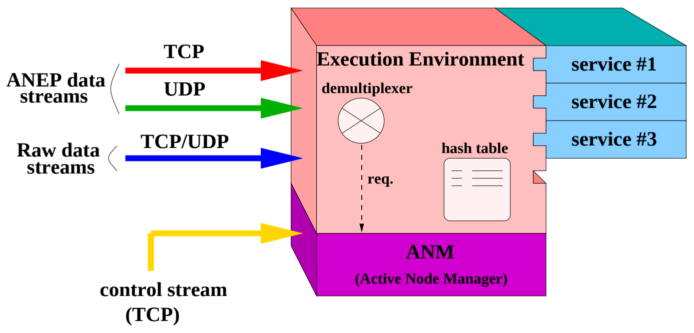

Figure 5. A Tamanoir Active Node (TAN)

The Execution Environment relies on a demultiplexer receiving active packets and redirecting these packets towards the adapted service in function of a hash key contained in packets header. New services are plugged in the TAN dynamically. The Active Node Manager (ANM) is dedicated to deployment of active services and to update routing tables.

3.1.1 User space and implementations issues

For the user space part of our EE, we choose to use a portable language for the active networks users be able to define and write their own services. Thus, the Tamanoir execution environment running in user space is entirely written in Java [10] which provides a great flexibility and is shipped with standard library. Unfortunately, the execution environment provided by the JVM *(Java Virtual Machine)* gives a very high level of abstraction, through which applications have some difficulties to obtain good performances. However, recent JVM releases ($\geq$ 1.3.x)

[2]Tamanoir (*great anteater*) is one of the strangest animal of south America : living in savanna, with an impressive tongue and a mouth of 2 centimeters diameter this animal only eats ants (up to 30000 daily). We choose this animal in reference to the ANTS [19] active network Java-based system.

give excellent performance for the mainstream hardware architecture (i.e x86), mainly due to the improvements of Just-In-Time (JIT) compilation techniques.

Each service is written in Java and inherited from a generic class called *Service*, itself inherited from the Java *Thread* class. Thus, each service is executed in a independent thread. For a given service, with TCP active streams, a thread service is dedicated for each stream while with UDP only one dedicated thread processes all streams. A given service can be applied on TCP or UDP active streams without change.

In order to improve safety and security, some EE runs each service in a separated sand-boxes (JVM). This approach does not improve resources sharing on a node (if one process consumes all CPU resources, others processes will not be able to work correctly). Moreover a standard JVM footprint takes more than 100 MB of memory for each instance. Using a multi-thread approach and running one service in each thread rather than running as many instance of JVM as services greatly improves the memory consumption.

3.1.2 Dynamic service deployment

The injection of new functionalities, called services, is independent from the data stream: services are deployed on demand when streams reach an active node which does not hold the required service. Two services deployment are available : by using a *service repository*, where TANs send all requests for downloading required services, by deploying service from TAN to TAN (TAN query the active node that sends the stream for the service). In order to avoid single point of failure service repository can be mirrored and replicated. When the service is available on a node, it is ready to process the stream. Of course, an active stream can cross equally a classical router, obviously, without any processing actions.

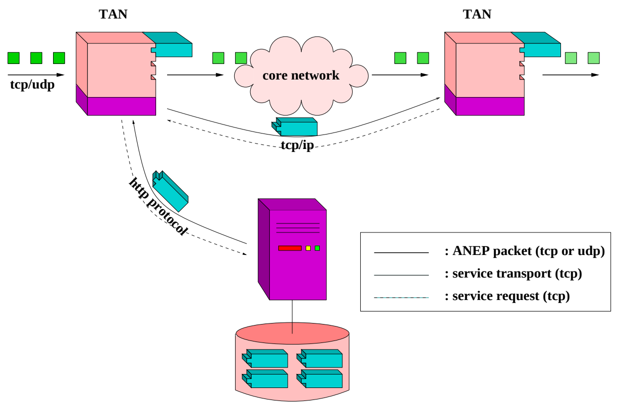

Figure 6. Two deployment scenario : from code repository or from a Tamanoir Active Node

3.2 Kernel space Execution Environment

After our EE user space investigation, we focus our approach on deploying lightweight services inside the kernel space of the operating system. Our main purpose, here, is to deport efficiently active functionalities from the high level execution environment (JVM) into the OS kernel.

Recent versions of the Linux kernel (2.4.x) are well furnished with networking functionalities and protocols : QoS, Firewall, routing and packet filtering. NetFilter is a framework for packet modification, outside the normal Berkeley socket interface [16]. With IPv4 communication protocol, NetFilter provides five *hooks*, which are defined points on the IP packet way. These hooks allow to develop and run modules, written in C, in the kernel level. The function *nf_register_hook* is used to attach a personalized function to a specific hook. When a packet reaches the hook, it is automatically transmitted to this personalized function.

The various modules which are set up into the OS kernel can be modified dynamically by active services. A Tamanoir active service, running inside the JVM, configures the NetFilter module by sending control messages (Fig. 7). This message is captured by the NetFilter module and used to parameterize lightweight services (forward, packet marking, drop…). This configuration on-the-fly allows to dynamically deport personalized functions inside the kernel.

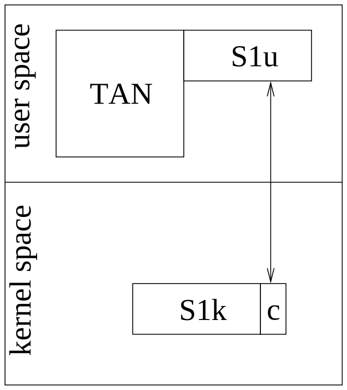

Figure 7. User space service ($S1u$) and kernel space service ($S1k$) communicates through the communication module (c)

3.3 Distributed service processing : Tamanoir on a cluster

High level and application oriented active services (compression, cryptography, transcoding on-the-fly…) require intensive computing resources. To support these services, Tamanoir use the Replicated EEs architecture shown in Fig.4(c). A Tamanoir Active Node embeds a dedicated cluster to support efficiently parallel services on streams.

The Linux Virtual Server (LVS)[22] software suite is dedicated to provide distributed servers (ftp, web, mail…) offering best performances in terms of throughput and availability. LVS is able to transmit packets in 3 different ways : LVS-NAT, based on address translation (NAT); LVS-DR (Direct Routing) where packets MAC address are changed and packets transmitted to a real server; LVS-TUN (tunneling) where packets are IPIP encapsulated and transmitted to a back-end machine.

We modify LVS usage for active networking and use it in Tamanoir EE. A Tamanoir-LVS is a collection of TAN execution environment running on a cluster of machines and linked together with an high performance network (Myrinet or GigaEthernet). A dedicated machine is configured as a front-end and is used to route packets from the Internet to back-ends machines. The front-end is seen by external client (on the Internet) as a single server dedicated to distribute connections on each node of the cluster in a round robin way or weighted round robin. Tamanoir Execution Environment is replicated on each back-end machine.

4 Experiments

An open problem in active networking is to compare and benchmark results between different execution environment. In order to show the efficiency of our approach, we first experiment raw performance obtained by a lightweight active service (packet monitoring : which includes packet marking, packets counting and forwarding) running in user space, in kernel space and, finally distributed on a cluster.

We deploy and use J2RE 1.3 a JVM from IBM on a GNU/Linux Debian distribution. We use three different platforms at different time to perform measures of throughput and latencies.

Our first experimental platform (P1) consists of dual-processor Pentium III 1Ghz for TANs and AMD Athlon 1 Ghz for client hosts. TAN and clients are connected through a dedicated Fast Ethernet (100Mb/s) network. This platform was used to measure latency in kernel and user space.

The second experimental platform (P2) is a cluster of 1U rack Compaq DL360 Proliant dual-PIII/1.4GHz with a PCI 66MHz bus, connected through Gigabit Ethernet network with a *Foundry* switch.

Third platform (P3) is a cluster of 1U rack SUN LX50 dual-PIII/1.4GHz with a 66MHz PCI bus too, connected through a Myrinet (Gigabits) network with a Myricom switch.

4.1 Raw performances on stand-alone base Tamanoir node

Throughput

First, we experiment the throughput achieved with packet monitoring service in Tamanoir user space EE (Figure 8). All experiments are based on the same Java client application which sends and receives ANEP packet streams.

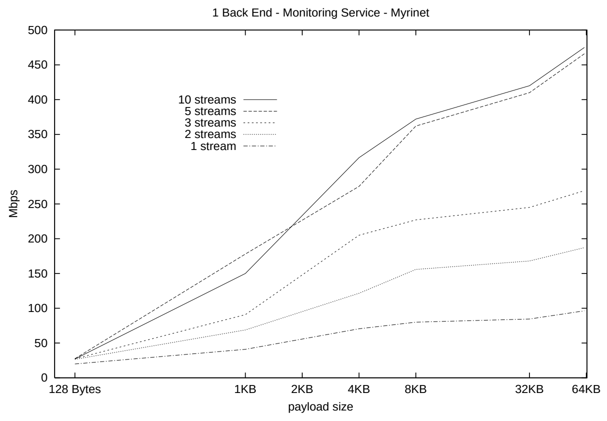

Figure 8. Throughput results on a stand-alone machine deploying active monitoring service

We perform experiments on a lightweight Monitoring service running in user space (in the JVM) on the P3 platform (with Myrinet networks). As shown in figure 8, in order to achieve the best throughput on only one active node we use large packets (between 32 and 64KB). We also send a large number of streams in order to benefit from aggregation. Figure 8 shows that we achieve more than 430Mb/s for 5 or 10 streams with packets size of 32KB. There is not a big difference between the 5 and 10 streams curves because we were limited by the number of sender and receiver machines to produce data streams.

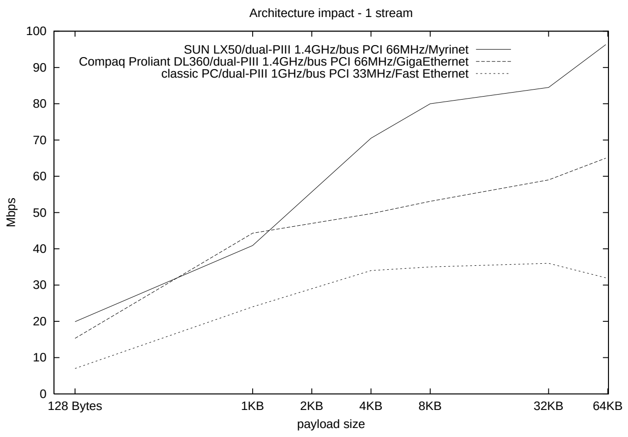

Figure 9. Architecture and network impact on P1, P2, P3 platforms

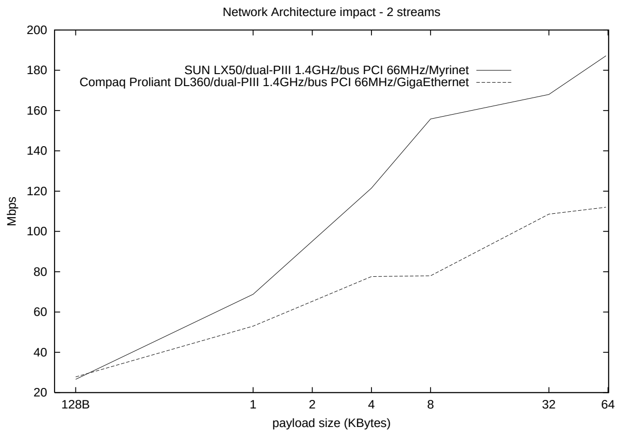

Figure 10. SMP impact on 1 stream transport

Figure 9 shows the network technology and architecture impact of our experimental platform on one stream transport. We consider that P2 and P3 platforms differ only by network capabilities (SUN LX50 and Compaq DL360 are comparable architectures). Myrinet technology is faster than GigaEthernet technology from 4KB packets size. It is more difficult to compare classic PCs shipped with a Fast-Ethernet cards with the SUN and Compaq machine, because these PC are older and use a slower PCI bus. For only one active stream there is finally a small difference with the 2 first expensive configurations.

As shown in figure 10 we experiment the benefit from SMP architecture by sending two parallel active streams. To process a TCP stream, Tamanoir instantiates a service inside a Java thread. For two streams, Tamanoir deploys two concurrent threads. On a dual architecture each thread is distributed on among CPU. So, the total throughput is in average the double of the throughput we have in figure 9.

Latency

Latency is the time for an ANEP packet to be processed and routed to its next destination by a Tamanoir active node. The measures were made on the first experimental platform thanks to Netfilter. When a packet reach the node we start a timer and stop it when the same packet now processed leave the node.

Packets crossing the Tamanoir active node remaining in the Linux kernel layer spend around 7 microseconds for basic forwarding operations with TCP or UDP (Fig. 11 and 12) on P1 platform. On the kernel level, the size of ANEP packet does not affect performances.

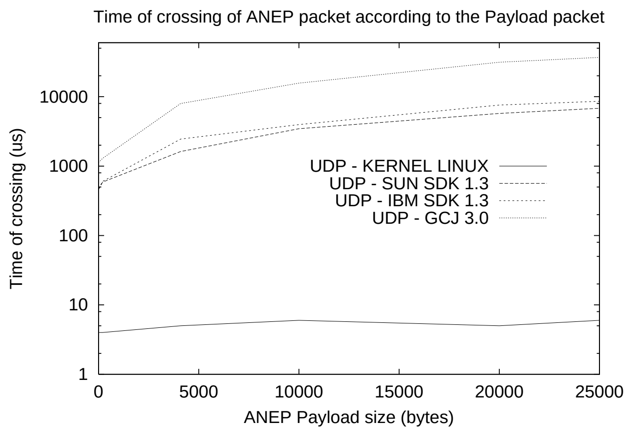

Figure 11. Crossing time for ANEP packets using UDP protocol

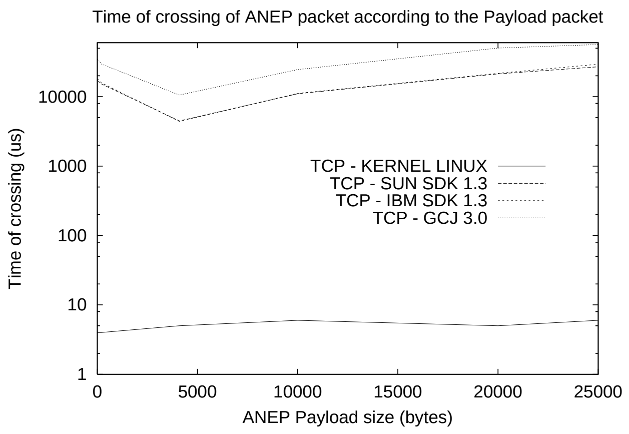

Figure 12. Crossing time for ANEP packets using TCP protocol

As we expected, when a packet is forwarded by the Tamanoir EE running in user space (in the JVM), performances are impacted. Results obtained with standard Java Virtual Machines (SUN [17] or IBM [9]) are quite similar. GCJ [6] is the GNU compiler for Java and provides native code from Java sources or bytecode (.class) files. Code is next linked with the library *libgcj*. Compiled execution environment obtained with GCJ, running in user space too, does not improve performances. With TCP transport (Fig 12), performances obtained with small packets remain around 16 ms (<4096 Bytes). Meanwhile we obtain better results with bigger packets. Around 4.4 ms for 4KB TCP ANEP packets with JVM and around 10.5 ms with GCJ version. This is due to the policy of small packets aggregation originally designed for improving data transmission. As shown in figure 11 we obtain better results, between 0.5 and 1.25 ms, on UDP with small packets size.

First experiments of deported services show how we can configure a Tamanoir Service running in kernel space. Figure 13 describes the case of an active service which needs to propagate half of data packets to the user space EE. By using standard messages, we can easily configure an active service running in a NetFilter module.

Figure 13 and 14 describes performances obtained thanks to a forwarding and packet marking service previously executed in the user space EE and next inside a NetFilter module. ANEP packets need 7 μs to be processed and routed by a service running in kernel space and need around 2 ms for a small packet (200 Bytes), then an order of magnitude of 1000.

By running some services in kernel space, we improve performance for active packets transport and low level services executed in kernel space unload the JVM (and user space) from superfluous work.

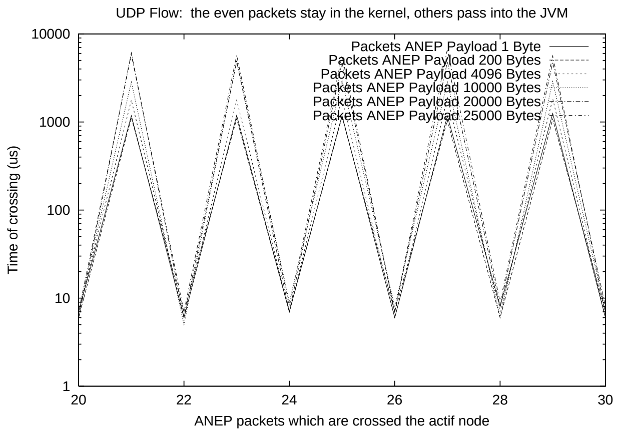

Figure 13. A packet on two goes up in the JVM, others are forwarded by the kernel

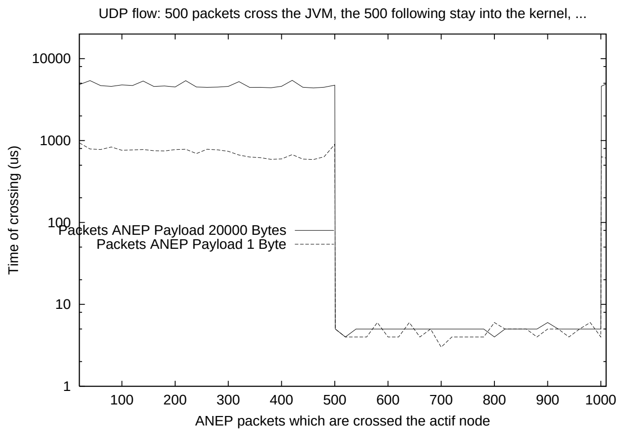

Figure 14. First 500 packets are processed inside the JVM, the following remain in the kernel

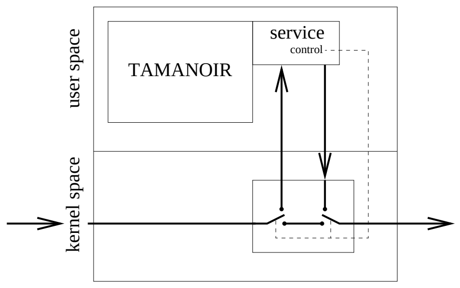

Figure 15. NetFilter module runs in kernel space. Tamanoir service runs in user user space and controls the kernel module to switch active packets

4.2 Performances on cluster-based Tamanoir node

Finally, we evaluate the benefit of distributing resources inside an active node by designing a cluster-based Tamanoir node. Giving first experiments results of LVS, we present here only performances achieved with Direct Routing [22].

The local experimental platform consists of 12 clients and a Tamanoir-LVS node (embedding one front-end and three back-ends) (Figure 16 show only 6 clients). *Sx* are active packets senders, *Rx* are receivers. Streams are routed by the front-end node acting as a director (streams dispatcher), three back-ends are attached to provide distributed resources. Results reported in this section have been measured on a Gigabit Myrinet network.

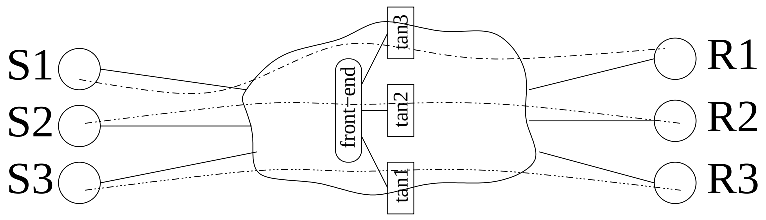

Figure 16. Platform topology with clients and a cluster-based TAN

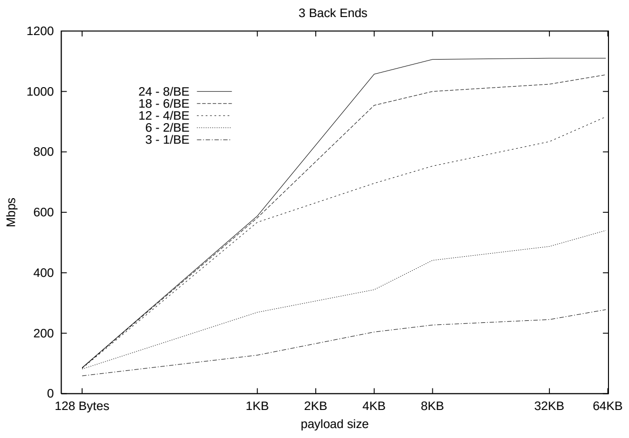

Figure 17. Throughput of a monitoring service in Mb/s a 3 nodes cluster-based TAN depending on number of streams

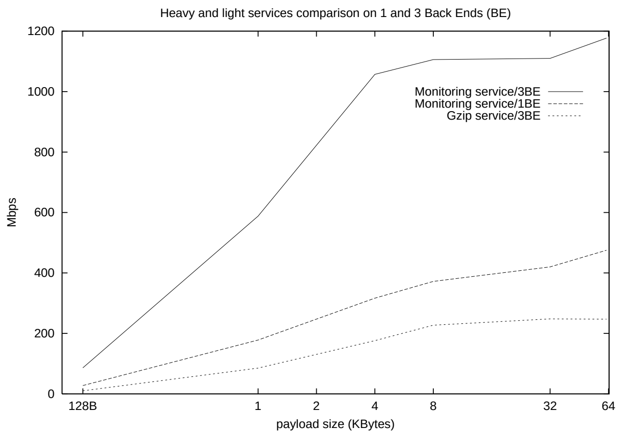

Figure 18. Lightweight (monitoring) and heavy (gzip) services comparisons

Figure 17 and 18 show performances results achieved experimentally on the P3 platform. Figure 17 presents performances obtained with a 3 node cluster based Tamanoir. With this configuration, Tamanoir supports Gbit performance (1.1 Gbit for 8KB packets) for a monitoring service applied on 24 active streams. We outpass 1 Gbits limit due to the high bandwith provided by Myrinet networks.

Figure 18 summarises the best results. All these results shows that to exploit all the potential of the processing resources, our active node needs to process lot of streams. But with an heavy or high-level service like the *Gzip* service (data compression on the fly of the tar file of the 2.4.19 Linux kernel sources), as shown in figure 18, active node resources are more used and throughput is reduced. With this heavy services a 3 back end based Tamanoir active node is still able to process up to about 240 Mb/s of active packets.

5 Related work

Since the proposition of active networks, numerous research projects deal with active networking technology. But on the topic of *high performance* active networking, only few are concerned. In this section, we attempt to give a brief overview of the mains works in the field of performance and active networks.

In the ANN project [4], each packet use a reference to an active module (call service) stored on a trusted code server. Modules are dynamically linked and executed like native code on the router. This technique is called Distributed code caching for Active Networks (DAN). Besides DAN, ANN claim that it "will" support ANTS [19] who is less focused on high performance but provide facility to design prototypes for experiments and refinement. From a hardware point of view, ANN people are aware of tightly coupling processing engine and network and also distribute computations over the CPUs available are important. These both last ideas gave birth, few years ago, to an Active Networking Node (ANN) which can be attached to an ATM switch backplane to meet the above requirements.

The PAN [13] project aims at developing a prototype called Practical Active Networking (PAN) that will eventually address safety, security, inter-operability and high performance. The current implementation focus only on high performance. This project was wrote in C and obtain very good raw performances. There is two implementations of PAN, one of these run in user-space and the other one in kernel-space as module. This last, allows to saturate a *Fast Ethernet* link with 1,500 bytes packets, with an overhead of only 13 % to process each packet. Performance are obtained thanks to limited copy, packets processing only when necessary and finally native code.

The TAGS [20] works focus on the packets demultiplexing bottleneck. In the Active Networking equipment each packets have to be demultiplexed not only to the network layer, but to the application level Execution Environment (EE). To speedup this demultiplexing stage TAGS implements a new active packet format called *Simple Active Packet Format (SAPF)*. Measurements show that SAPF packets can be processed 30% faster than regular IP packets that use the traditional ANEP header.

CANEs [14], which stands for Composable Active Network Elements, is a project which aims to design a coherent architectural framework for active networking including consistent terminology, minimum functional requirements, and interface specifications. The main goal is to provide network-based capabilities that enhance the communication service and/or performance seen by users of the network with mechanisms like reacting to a congestion, transparent caching of information in network nodes, and support for multicast video distribution to heterogeneous end-users.

CANEs is an execution environments running on *NodeOS* (but currently on an interim platform called *Bowman*[15] implementing just a subset of the NodeOS interface).

The AMP[2] project is developing a new software base that allows active code to be executed securely, safely and with high performance. AMP system should provide a fast and lightweight execution environment for Active Networks nodes. By enforcing resource usage limitations, active code cannot tamper with the rest of the active node. AMP take advantage of techniques and software developed by the DARPA-funded *exokernel* project that demonstrate physical resources may be managed by user-level applications in way that allows both efficiency and potential for protection.

The *Protocol Boosters*[5] project aims to improve the performance of heterogeneous distributed computing systems by improving the performance of the communication protocols that are used by the nodes of the distributed systems. They can dynamically avoiding any unnecessary protocol processing and dynamically optimizing the communication protocol. This will give an efficient programming model for active networks applications.

Clara is an architecture for a cluster based computing router used in the *Journey* network model providing computation as a scalable network service. A "media unit" crossing a *Clara* computing router will be processed in function of local conditions resources availability, making decision independently of other computing router. This model doesn't guarantee that each "media unit" will arrive processed. It's in the same spirit of best-effort routing in IP networks. A media unit processed or unprocessed is determined by the *IP Router Alert* option. If un-detected, packets are directly routed by IP, else packets are handed up to the *Clara* software for possible processing. The *Clara* architecture use one PC for routing, the others linked by a SAN, are only dedicated for processing, with a simple round-robin dispatching algorithm. A prototype has been evaluated in the context of real-time transcoding MPEG video.

While most projects are software environments (except ANN) for packet processing on programmable routers (which are workstations that act like routers). Some companies (like IBM, Intel, Motorola,...) make available commercially programmable packet processing engines for routers called "Network Processors". These Network Processors perform processing from the data link layer to the application layer. They come as system-on-a-chip designs that combine processors, memory and *IO* on a single ASIC. In [21] they study the design of an high performance active router with these brand new specifics processors.

6 Conclusion

In this paper, we present our first step towards the design of an high performance software active router. We propose a new architecture for active nodes targeted to provide high performance support for active services. We validate this architecture by designing the Tamanoir execution environment. Tamanoir supports deployment of services in user space level, kernel level and distributed services on a cluster. Our experiments have been deployed on Gigabit networks. A stand-alone SMP based Tamanoir node can support around 500 Mbit/s of bandwith for lightweight service with its multi-threaded design and services support in kernel. We demonstrate the need to deploy cluster based Tamanoir nodes to fully support a GBit network.

One of our next step consists of fully integrating and evaluating active services inside Network Programmable Interface Card (Myrinet). With this offload approach, Tamanoir should benefit from services located closed to the link and directly executed on the network card.

In our quest of performances we also want to reduce the impact of ANEP packets by supporting other active packets format (like SAPF or custom format) and simple IP packets requesting active services.

Providing performance in active routers is also a mandatory aspect for high performance long distance applications. One of our current research concerns the deployment of active networking technology to the requirements of Grid middlewares and applications ([11]).

References

[1] S. Alexander, B. Braden, C. Gunter, A. Jackson, A. Keromytis, G. Minden, and D. Wetherall. Active Network Encapsulation Protocol (ANEP). RFC Draft, Category : Experimental, July 1997.

[2] AMP Project. http://www.pgp.com/research/nailabs/distributed/amp.asp.

[3] Nanette Boden, Danny Cohen, Robert Felderman, Alan Kulawik, Charles Seitz, Jakov Seizovic, and Wen-King Su. Myrinet : a gigabit per second local area network. *IEEE-Micro*, 15(1):29–36, February 1995.

[4] D. Decasper, G. Parulkar, S. Choi, J. DeHart, T. Wolf, and B. Plattner. A scalable, high performance active network node. In *IEEE Network*, volume 13, January 1999.

[5] D. Feldmeier, A. McAuley, J. Smith, D. Bakin, W. Marcus, and T. Raleigh. Protocol boosters. *IEEE Journal On Selected Areas in Communications*, 16(3):437–444, April 1998.

[6] GCJ. The GNU Compiler for the Java Programming Language. http://sourceware.cygnus.com/java/.

[7] Jean-Patrick Gelas and Laurent Lefèvre. Tamanoir: A high performance active network framework. In C. S. Raghavendra S. Hariri, C. A. Lee, editor, *Active Middleware Services, Ninth IEEE International Symposium on High Performance Distributed Computing*, pages 105–114, Pittsburgh, Pennsylvania, USA, August 2000. Kluwer Academic Publishers. ISBN 0-7923-7973-X.

[8] Jean-Patrick Gelas and Laurent Lefèvre. Mixing high performance and portability for the design of active network framework with java. In *3rd International Workshop on Java for Parallel and Distributed Computing, International Parallel and Distributed Processing Symposium (IPDPS 2001)*, San Fransisco, USA, April 2001.

[9] IBM. IBM Java Developer Kit for Linux. http://www.alphaworks.ibm.com/tech/linuxjdk.

[10] Java programming language. http://java.sun.com/.

[11] L. Lefèvre, C. Pham, P. Primet, B. Tourancheau, B. Gaidioz, J.P. Gelas, and M. Maimour. Active networking support for the grid. In Noaki Wakamiya Ian W. Marshall, Scott Nettles, editor, *IFIP-TC6 Third International Working Conference on Active Networks, IWAN 2001*, volume 2207 of *Lecture Notes in Computer Science*, pages 16–33, oct 2001. ISBN: 3-540-42678-7.

[12] Laurent Lefèvre and Olivier Reymann. Combining low-latency communication protocols with multithreading for high performance dsm systems on clusters. In *8th Euromicro Workshop on Parallel and Distributed Processing*, pages 333–340, Rhodes, Greece, Jan 2000. IEEE Computer Society Press.

[13] Erik L.Nygren, Stephen J.Garland, and M.Frans Kaashoek. PAN: A High-Performance Active Network Node Supporting Multiple Mobile Code Systems. In *IEEE OPENARCH '99*, March 1999.

[14] S. Merugu, S. Bhattacharjee, Y. Chae, M. Sanders, K. Calvert, and E. Zegura. Bowman and canes: Implementation of an active network. In *37th Annual Allerton Conference, Monticello, IL*, September 1999.

[15] S. Merugu, S. Bhattacharjee, E. Zegura, and K. Calvert. Bowman: A node os for active networks. In *IEEE INFOCOM '2000*, mar 2000.

[16] Rusty Russell. Linux Filter Hacking HOWTO. netfilter description and usage, july 2000.

[17] SUN. Kit de dveloppement java de sun. http://java.sun.com/.

[18] David Tennenhouse and David Wetherall. Towards an active network architecture. *Computer Communications Review*, 26(2):5–18, April 1996.

[19] David Wetherall, John Guttag, and David Tennenhouse. ANTS : a toolkit for building and dynamically deploying network protocols. In *IEEE OPENARCH '98*, April 1998.

[20] Tilman Wolf and Dan Decasper. Tags for high performance active networks. In *OpenArch2000, Tel Aviv, March 2000*, 2000.

[21] Tilman Wolf and Jonathan S. Turner. Design issues for high performance active routers. *IEEE Journal on Selected Areas of Communication*, 19(3):404–409, March 2001.

[22] Wensong Zhang. Linux Virtual Server for Scalable Network Services. In *Ottawa Linux Symposium*, 2000.