Deploying OS filtering capabilities for the improvement of software active routers

Jean-Patrick Gelas, Jérôme Guilloux, Laurent Lefèvre
RESAM laboratory - INRIA Action RESO
Ecole Normale Supérieure de Lyon
46, allée d'Italie, 69364 LYON Cedex 07, France
Jean-Patrick.Gelas@ens-lyon.fr, Jerome.Guilloux@ens-lyon.fr, Laurent.Lefevre@inria.fr

Abstract *Achieving high performance with software active routers remains a great challenge. Recent versions of Linux OS provide networking functionalities for packet filtering (NetFilter). We take advantage of these features to provide more performant software active routers. We deploy active services from high level execution environment to kernel modules. We focus our experiments on the Tamanoir execution environment over a recent Linux kernel implementation. We describe results of performance obtained with this approach.*[1]

Keywords: NetFilter, Linux, active networks, Tamanoir

1 Introduction

In active routers, a service (personalized function) is applied on all transported active IP packets. In software active routers, all active (and even passive) packets must be processed by the active execution environment (running in a application level, mainly in a JVM). This operation is one of the main key limitating factor in terms of performance.

Within software based active routers, operating systems play an important role. Recent version of Linux provide possibilities to support filtering functionalities in the kernel (NetFilter module). With *hooks* linked to specific packet actions, users can run personalized applications.

The Tamanoir system is an active Execution Environment which allows users to efficiently deploy personalized services inside the network. We propose to deport simple active services from high level JVM to low level NetFilter modules. Our goal is to provide new levels of performance to software active routers.

The paper is organized as follows. In section 2, the NetFilter module is briefly introduced. Section 3 describes the Tamanoir architecture developed in our laboratory. Section 4 presents the deport of active services from JVM to NetFilter modules and performance gains we obtained are described in section 5. We finish by some conclusions and future directions for the improvement of software active routers.

2 Linux Filtering Capabilities : NetFilter architecture

2.1 NetFilter architecture

Recent versions of the Linux kernel (2.4.x) are well furnished with networking functionalities and protocols : QoS, Firewall, routing and packet filtering. For packet filtering, kernels 2.0 had IpFwadm, kernels 2.2 had IpChains and IpMasqadm, and kernels 2.4 proposes IpTables with NetFilter. NetFilter is a framework for packet modification, outside the normal Berkeley socket interface (see [1]). With IPv4 communication protocol, NetFil-

[1]This work is supported by the RNRT VTHD++ project and by ANVAR Active Network platform.

ter provides 5 *hooks*, which are defined points on the IP packet way. These *hooks* are (see fig. 1) : `NF_IP_PRE_ROUTING`, `NF_IP_LOCAL_IN`, `NF_IP_LOCAL_OUT`, `NF_IP_FORWARD` and `NF_IP_POST_ROUTING`.

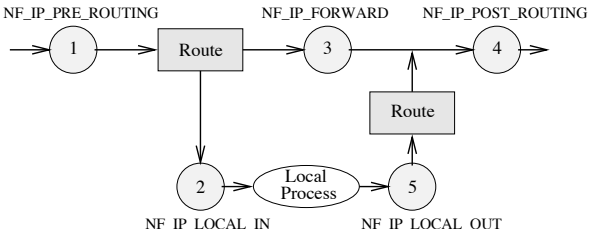

Figure 1: NetFilter Hooks

Through these 5 hooks, NetFilter is able to filter IP packets in order to provide firewall functions, perform Network Address Translation (NAT), modify, drop and generate packets...

NetFilter is able to realize all these functions thanks to IpTables : a packet selection system which has been built over the NetFilter framework. NetFilter includes three tables which correspond to the three principal functions seen previously :

- Filter table : contains all filtering rules;
- NAT table : defines all NAT functions, just as well of the source as of the destination;
- Rule table : contains the packet modifying rules.

2.2 Filtering : Filter table

Filter table contains three "firewall chains", the chain *INPUT*, *OUTPUT* and *FORWARD* (see Fig. 2). Unlike IpChains, chains *INPUT* and *OUTPUT* are not used that if packets are destined to a local process. IpChains use *INPUT* and *OUTPUT* for the body of the traffic.

When a packet arrives through the network card, the kernel first looks the destination of the packet (routing functionality); If it's destined for this box, the packet passes downwards in the diagram, to the *INPUT* chain.

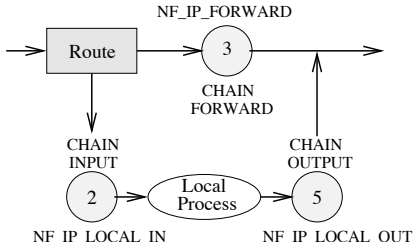

Figure 2: NetFilter Hooks : Filter table

If it passes this, any processes waiting for that packet will receive it. Otherwise, if the kernel does not have forwarding operation enabled, the packet is dropped. If forwarding is enabled and the packet is destined for another network interface, the packet goes rightwards on figure 2 to the *FORWARD* chain. If accepted, the packet is sent out. Finally, a program running on the box can send network packets. These packets pass through the *OUTPUT* chain immediately, then the packet continues out to the destined interface.

2.3 Network Address Translation : NAT table

NetFilter is a very powerful software tool for Network Address Translation(Fig. 3). NAT table allows to perform on the fly address translation in order to offer several networking services : address masquerading of local network, sending packets on another host, deploying transparent proxies...

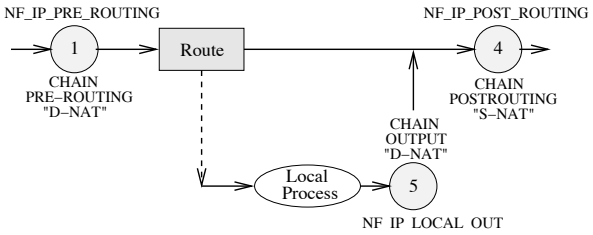

Figure 3: NetFilter Hooks : NAT table

The table of NAT rules contains three lists called 'chains': each rule is examined in order until one matches. Two most important chains are called *PREROUTING* (for Destination NAT, as packets first come in), and

POSTROUTING (for Source NAT, as packets leave) (see Fig. 3). At each of the points above, when a data packet arrives, it is associated with the corresponding connection.

3 Tamanoir Architecture

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[2] might be seen as a natural consequence.

In our active networking vision, routers and any network equipments (like gateways, proxies,...) can perform computations on user data in transit, and end 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 the architecture, the users and the service providers requirements.

The Tamanoir[3, 4] (see Fig. 4) suite is a complete software environment 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. The both main transport protocol (TCP/UDP) are supported by the TAN for carrying data. We use the ANEP (Active Network Encapsulated Protocol)[5] format to send data over the active network.

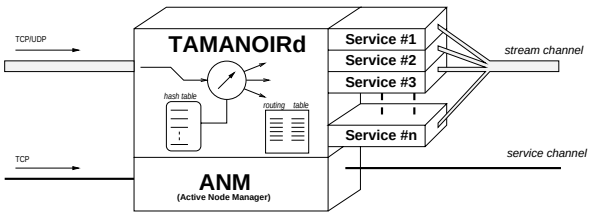

Figure 4: A Tamanoir Active Node (TAN)

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 is written entirely in JAVA [6]. JAVA 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.

4 Tamanoir and NetFilter architecture

4.1 From high level active services to low level kernel active services

Our main purpose is to efficiently deport active functionalities from the high level execution environment (JVM) into the OS kernel (*Kernel* and *Execution Environment* layers in Fig. 5).

Figure 5: TAN Architecture

The hooks provided by NetFilter allow us to develop modules on the kernel level. The function *nf_register_hook* is used to attach a

personalized function to a specific hook. When a packet knocks the hook, it is automatically transmitted to this personalized function.

4.2 Configuring NetFilter from a Tamanoir Service

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, can configure the NetFilter module by sending a control message (see Fig. 6). This message is captured by the NetFilter module and used to parameterize actions provided by NetFilter (forward, packet marking, drop...). This configuration on-the-fly allows to dynamically deport personalized function inside the kernel (see section 5.2).

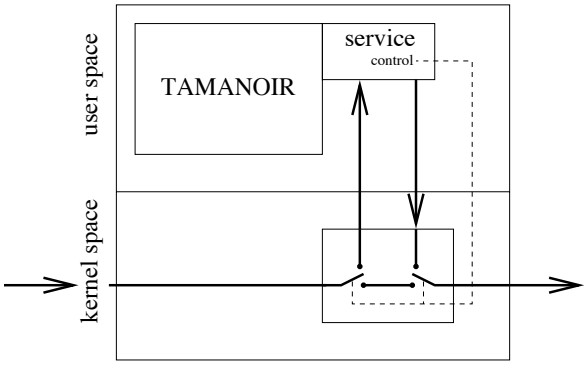

Figure 6: NetFilter module runs in kernel space. Tamanoir service runs in user user space and controls "switches" of the NetFilter module to switch "active packets".

5 First experiments

To efficiently measure the latency of an ANEP packet crossing a TAMANOIR router, we place a start time function (*do_gettimeofday* ,equivalent of *gettimeofday* on the user space)in the PRE_ROUTING and a stop time function in the POST_ROUTING hook. The experimental platform consists of Bi-Proc. SMP Pentium III 1Ghz, with Fast Ethernet cards.

5.1 Results with UDP and TCP protocol

We experiment the deport of active forwarding service inside the kernel for TCP and UDP active packets (Fig. 7 and 8).

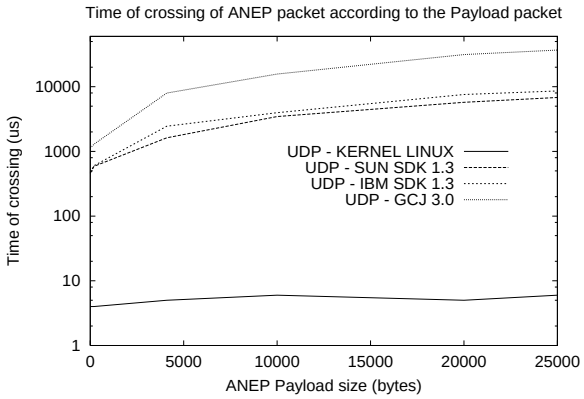

Figure 7: Crossing time for ANEP packets using UDP protocol

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. 7 and 8). On the kernel level, the size of ANEP packet does not really affect performances. This is due to the fact that in our function, we have zero-copy protocols. Indeed, packets are simply redirected towards the hook of exit by changing on-the-fly destination address. We can note that there are no real differences of performance for TCP or UDP packets in the kernel level.

As we expected, when a packet is forwarded by the execution environment in the JVM level, performances are really impacted. Results obtained with standard Java Virtual Machines (SUN [7], IBM [8]) are quite similar (around 1.6 ms for 4KB UDP ANEP packets). Compiled execution environment obtained with GCJ [9], also running in user space, do not improve performances (around 7.9 ms for 4KB UDP ANEP packets). This surprising low performances are due to the lack of optimized native code provided by GCJ compiler.

With TCP transport (Fig 8), lower performances obtained with small packets are around

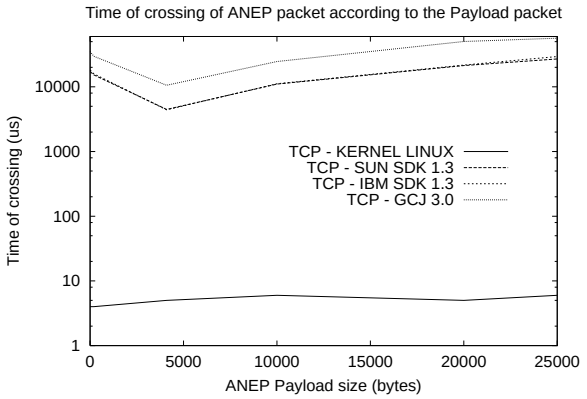

Figure 8: Crossing time for ANEP packets using TCP protocol

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.

5.2 Communications between NetFilter and Tamanoir active service

First experiments of deported services show the dynamicity of configuring the NetFilter module from Tamanoir Service. Figure 9 describes the case of an active service which needs to propagate half of data packets to the Execution Environment. By using standard messages, we can easily configure and active NetFilter module to dynamically support low level services (forward, packet marking, intelligent drop...) in the kernel.

Figure 10 describes performances obtained from a forwarding and packet marking service previously executed in the Execution Environment and moved inside a NetFilter module (after 500 packets).

6 Related works

Our execution environment (EE) run over standard operating system (OS) to deploy ac-

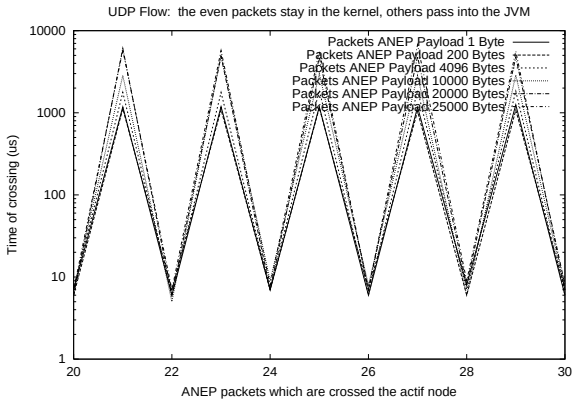

Figure 9: A packet on two goes up in the JVM, others are forwarded by the kernel

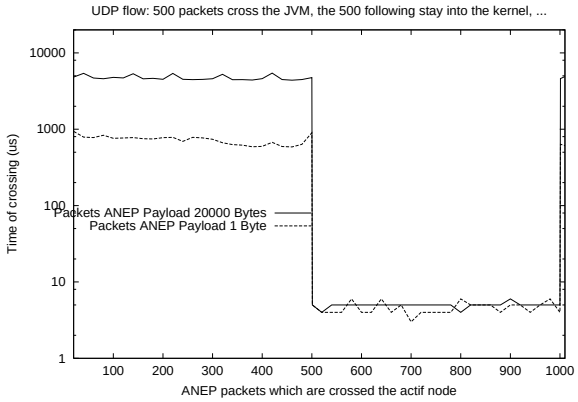

Figure 10: First 500 packets are processed inside the JVM, the following remain in the kernel

tive routing functionalities. But for accessing router memory, communication hardware and computational resources in an efficient way, it should be better to use an OS interface especially designed for the requirements of an Active Node. This interface is generally called a *NodeOS*, mapped between the EE and the OS which defines a set of functions to access and manage the active node resources.

One of the most famous project in this domain is CANEs (Composable Active Network Elements) [10]. CANEs is actually, an EE currently running over *Bowman* which implements a subset of the NodeOS interface services and can support other EEs in addition to CANEs. *Bowman* incorporates an efficient and flexible packet classification algorithm, supports multi-threaded per-flow processing, and utilizes real-time processor scheduling to achieve deterministic user-space performance.

Larry Peterson et al.[11] report their experiences in implementing an OS interface, for Active routers over three different OS environments (Scout, OSkit, and exokernel). The first implementation is layered on top of *Scout* which encapsulate flow of I/O data in an explicit path like NodeOS specifications, they take advantage of this similitude to implement both the traditional and active forwarding using exactly the same mechanism. The second is JANOS (Java-oriented Active Network Operating System) which aims to give resource management and control, and first class for untrusted active application written in Java. It is implemented above *OSkit* and includes a resource-aware, multi-heap Java Virtual Machine, and an active network protocol EE who is an extension of ANTS called ANTSR. The last one is AMP layered on top of *exokernel*. Its goal is to provide a secure platform upon which EEs and active applications can run, without unduly compromising efficiency. It uses also ANTSR.

OS interface run in user space. It should be more efficient to provide adapted services in kernel space for performance reasons. But writing a specific OS is a hard and complex task. We think it is better to keep a well known standard Operating System, like Linux, especially because it is today shipped with modules, providing very high flexibility for accessing low level layers in the kernel area.

7 Conclusion and future works

The research described in this paper is the first step in the design of high performance software active routers. While most of existing projects in active networks use an OS interface running in user space, we propose solutions to efficiently deport services and functionalities in the kernel without changing it. By using filtering capabilities of the OS, we greatly improve performances for active packets transport. First experiments show important gains (from 10 ms to 7 μs) for low level active services (forwarding, packet marking...). Of course, low level routing functionalities are better performed with hardware support. A latency of 5 μs is still too important for Gigabit high performance routing. But in the case of Active Networks providing dynamic services, our approach allows new possibilities for active transport without modifying OS or deploying hardware support.

Our next step will consist on fully integrate and evaluate active services in NIC and kernel implementation. We focus our research on programmability of network interface cards (like Myrinet [12], Ramix [13]) to efficiently deport active functionalities near the wire, inside the network cards.

References

[1] Rusty Russell. Linux filter hacking howto. NetFilter description and usage, july 2000.

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

[3] 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.

[4] 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.

[5] Scott D. Alexander, Bob Braden, Carl A. Gunter, Alden W. Jackson, Angelos D. Keromytis, Gary J. Minden, and David Wetherall. Active network encapsulation protocol (anep). RFC Draft, Category : Experimental, http://www.cis.upenn.edu/switchware/ANEP/, July 1997.

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

[7] SUN. Sun java development kit. http://java.sun.com/.

[8] IBM. Ibm java development kit. http://www-106.ibm.com/developerworks/java/jdk/linux130/.

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

[10] 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.

[11] Larry Peterson, Yitzchak Gottlieb, Mike Hibler, Patrick Tullmann, Jay Lepreau, Stephen Schwab, Hrishikesh Dandekar, Andrew Purtell, and John Hartman. An os interface for active routers. *IEEE Journal on Selected Areas of Communication*, 19(3), march 2001. http://www.cs.utah.edu/flux/janos/.

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

[13] Ramix network programmable cards. http://www.ramix.com.