SARA: a Simple Active Router-Assistant Architecture [1] [2]

David Larrabeiti, María Calderón, Arturo Azcorra, Manuel Urueña
Universidad Carlos III de Madrid
Av. Universidad, 30
Leganés – 28911 Madrid
Spain
{dlarra,maria,azcorra,muruenya}@it.uc3m.es

Jens E. Kristensen, Lars Kroll Kristensen
Ericsson Telebit A/S
Skanderborgvej 232
DK-8260 Viby J.
{Jens.Kristensen, Lars.K.Kristensen}@lmd.ericsson.se

Abstract. The usage of external processors to support active processing seems a safe and cost-effective approach to add this functionality to existing routers. This paper reviews this router-assistant way of making active nodes and describes a new system (SARA) designed upon this technique using an enhanced commercial router. The features new to this type of architecture present in SARA are transparency, IPv4 and IPv6 support, and full control over layer 3 and above. Quantitative results on the usage of JAVA-based execution environments enhanced with raw socket facilities are also reported.

Keywords: active network, programmable network, IPv6, router-assistant, raw-socket, JAVA

1 Introduction

Most existing systems regard explicit addressing of intermediate nodes as a transitory constraint to be overcome when active processing gets moved from prototype host-based implementations to real routers. However, in practice, this limitation usually remains a permanent feature during the whole life of the project because of the many difficulties of porting an active execution environment to a real router's architecture without an important penalty on performance. This is usually due to the fact that, given the current incipient demand for active services, the only cost-effective place to run an execution environment and applications today is the router's CPU. This means that node-by-node packet processing is actually emulated by server-by-server relaying.

The issue of active node location transparency and the implementation problems implied by this feature have scarcely been studied. By location transparency we mean architectures where end-user applications needn't explicitly address the active nodes that will process their packets through the network. Instead, nodes with active extensions identify packets that deserve specific processing passing through towards their destination. This seems difficult with the host-based approach, unless a tight cooperation router-host is in place.

[1] This work has been funded by the IST project GCAP (Global Communication Architecture and Protocols for new QoS services over IPv6 networks) IST-1999-10 504 and by CICYT (the Spanish Commission of Science and Technology) under projects TEL99-0988-C02-01 and AURAS.

[2] Disclaimer: this paper reflects the view of the authors and not necessarily the view of the GCAP Project.

To this end, this work studies the usage of external processors [1][2] to support transparent active processing and claims that for many active applications, this type of “active outsourcing” should be regarded more than a transition mechanism to native hardware support, unlike considered today. This practical study is centered on the design of a new platform based on a set of structural elements selected for being considered actually usable in a real network wishing to enable active services.

This paper is structured as follows. Section 2 addresses the issue of transparency implementation both in IPv4 and IPv6. Section 3 reviews the router-assistant way of making active nodes and gives a rationale for each design decision of the architecture, including relevant implementation requirements on the behaviour of routers, assistants and end-systems. Section 4 provides an overall description of a system based on this framework: SARA. Finally, section 5 discusses the proposed approach in the light of some related work, before drawing conclusions. In the following, we shall use the shorthands AA, EE and AN to refer to the terms Active Application, Execution Environment and Active Networks.

2. Transparency

The classical mechanism used by hosts to access most high layer services other than packet forwarding is by means of control plane entities that communicate with service agents. This procedure is predominant in IPv6, like it is in IPv4 applications. With this method, the user node is responsible for locating the nearest agents (real servers or proxies) that provide the service. This is especially appropriate for services implemented statically, usually of transactional nature, based on servers. But, how does an end-user access and program network services when the packet processing is distributed and performed by the routers along the path (or tree) to a destination? This is the case of active networking.

Obviously, the address of the nearest active node could be obtained by one of the general procedures described above. But it seems that the explicit addressing of routers and relaying between active nodes does not match the dynamic nature of the IP backbone structure. Moreover, it would not work with native multicast packets where a tree of processor nodes is implied. Last but not least, it is important that active processing can operate on regular packets (packets not to be processed by active entities). Therefore, an important requirement for the effective deployment of network-based services seems to be agent location transparency or, in other words, active node location transparency, also tightly linked to mobile agent technology. This means that senders expecting special processing from the network simply address packets to their final destination, and routers recognize and process them according to a given code.

A clear example is an n-to-n multi-QoS multimedia flow service over a best effort internet, for example for videoconference applications running on end-systems with heterogeneous capabilities or different access link capacities. One valid option is using layered coding and multicast, letting IP reduce the rate of the flow at congested links. A second (and complementary) way to implement this service, more complex but also more effective in terms of overall bandwidth usage is performing intelligent packet discard, rate adaptation, transcoding or layer selection, etc at network nodes branching to heterogeneous receivers on different links. The advantage of this approach is that intelligent processing within the network can adapt the flow to the specific needs of a subtree of receivers and prevent forwarding packets that will never reach its destination due to bottlenecks downstream or that will not be profitable in the playback. In this case, if active node location transparency is enabled, each party can multicast its traffic unaware of which network nodes in the distribution tree will adapt the flow. Another important example is reliable multicast, which can become scalable thanks to distributed retransmission and aggregation (avoiding the well-known nak implosion problem). And the same can be applied to multiple packet processors available in the market today: TCP or HTTP spoofing, layer 4 switching, NAT, masquerading, content filters, etc which are in fact incipient forms of statically preconfigured transparent active networking, in the sense that they provide enhanced processing beyond packet forwarding.

Implementation Details in IP

However, there is an important drawback to the implementation of transparency of active nodes: efficiency (letting alone other important efficiency problems already analysed by other authors like making the router execute user code, that we will assume here solvable by means of ad-hoc active code processors). To apply active networking in a realistic context, it should be assumed that regular packets are to be processed by the same router as active packets, and active routers are supposed to live together with non-active legacy routers. How can a router - whose behaviour is optimized for packet forwarding just by checking the destination address - keep its performance up if we require a special treatment to packets not explicitly addressed to them?

This problem is not new. A solution already deviced for signaling protocols that need such a feature is the Router Alert Option of IP (RFC-2113) (figure 1, top). As described in this document, the Router Alert Option has the semantics: "routers should examine this packet more closely (check the IP Protocol field, for example) to determine whether or not further processing is necessary". A new option type was necessary because some IP options were already implemented in the fast path of some routers. Only options not supported in the fast path will push the packet into the slow path and hence, in principle, no performance penalty is caused to regular data packets. Current protocols using this option include IGMPv2 (RFC-2236) and RSVP (RFC-2205). In IPv6, Router Alert (RFC-2711) (figure 1, bottom) is a Hop-by-Hop Option with the same general semantics. However, unlike IPv4 where only value 0 is defined, in IPv6 three values have already been reserved, one of them (value=2) for active network messages. The others are assigned to RSVP and to Multicast Listener Discovery messages included in ICMPv6.

IPv4:
```
            Type    Length        Value
         +--------+--------+--------+--------+
         |10010100|00000100|  2 octet value  |
         +--------+--------+--------+--------+

   Value:
     0 - Router shall examine packet
     1-65535 – Reserved
```

IPv6:
```
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0 0 0|0 0 1 0 1|0 0 0 0 0 0 1 0|        Value (2 octets)       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

      Value:

         0        Datagram contains a Multicast Listener Discovery
                  message [RFC-2710].
         1        Datagram contains RSVP message.
         2        Datagram contains an Active Networks message.
         3-65535  Reserved to IANA for future use.
```

Fig. 1. Router Alert option format for IPv4 and IPv6

Another important feature of IPv6 very useful in active networking is the base header flow label field. Router alerting packets are extremely useful to program the behaviour of active nodes along a given path or domain.

However, per-flow processing also requires fast paths to specialized processors. This can be achieved by basing internal flow management on flow labels. The base IPv6 header is a privileged position for a label assigned by IP at the source host for a single data flow and preserved throughout the network. Its purpose is preventing the analysis of higher layer protocol headers when packet classification is necessary. Once a pair <source address, flow label> has been identified by a previous signaling router alerting packet as an active flow, the node gets programmed to provide ad-hoc processing to that flow at full speed. In this sense, active networking can be used as a vehicle for the rapid development of protocols for traffic engineering.

If active node location transparency is not desired, or the user just wants to simply select the network provider that will implement active processing, IPv6 also features a standard powerful tool: the routing header. Thanks to the ubiquitous support of this type of source route specification - as well as the necessary security methods enabling it in practice - as a conformance requirement for all IPv6 implementations, it is possible to specify the active nodes that will implement the service. This is very important in networks where forward and backward paths are different. The requirement that the reply from the endpoint will carry the same routing header is important to make sure that signaling requests and responses of the active protocols designed will follow a coherent sequence of nodes i.e. the active entities are the same, despite the direction of packets. This feature can be used in junction with anycast addressing to pass through the nearest active nodes or let the network decide the cheapest active route at a given time.

3. The Router-Assistant paradigm

With the purpose of defining a pragmatic framework to implement basic active networking functions, valid both for IPv4 and IPv6, and realistic in an industrial context, a new active networking architecture design is being developed in the context of the IST project GCAP [3]. This section defines this framework which we will refer to in the remainder of the paper as the “router-assistant” approach. This way of building active networks is based on a set of techniques –many of them also proposed by other authors– selected according to its industrial applicability.

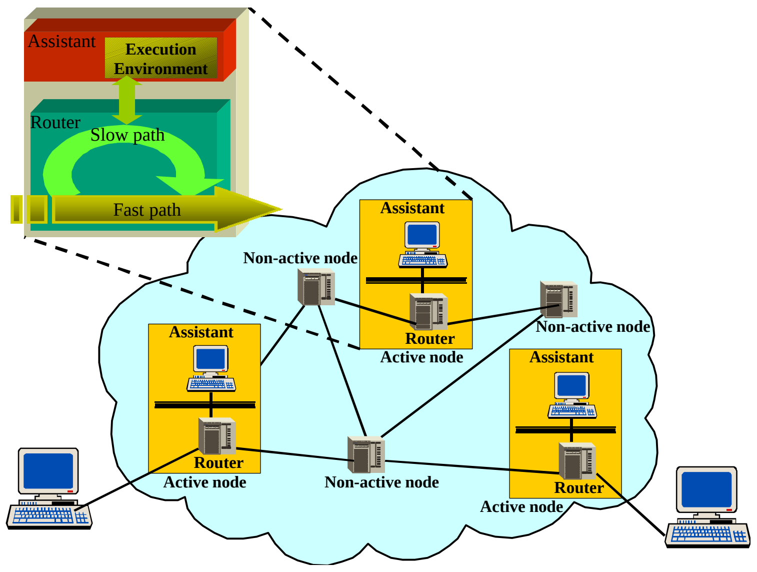

Fig. 2. Active network following the Router-Assistant paradigm

The key features of this procedure and the rationale behind them are:

- **The Execution Environment runs on the Assistant's processor**. Every router wishing to enhance its functionality with active extensions can delegate this task to a host called Assistant (or, in general, an array of hosts), directly attached to the router on a high-speed LAN. The assistant runs the execution environment and OS supporting active applications. Hence, active applications may not run on the router hardware and, consequently, the performance penalty on the router is bound to the cost of identifying and diverting active packets. Therefore, a primary conformance requirement is as simple as requiring the router to divert all active packets traversing the router to the assistant, except those coming from the assistant itself.
- **The assistant host processes active packets**. A packet diverted from the router must be transparently input to the execution environment, processed from layer 3 up by the active applications running on it and forwarded if that is the case.
- **Delegation of Active Functions**. For the purpose of running active applications on behalf of the real router, the assistant must cooperate hand in hand with the router. This cooperation is based on three points: active packet diversion, router state communication and router-assistant protocol (RAP) with regular packet diversion. Two conformance levels have been defined. Level 1, for routers supporting only the first two items, and level 2, for routers supporting the full three items.
  - **Active packet diversion**. It has already been defined above.
  - **Router state view**. A procedure by which the execution environment makes available the router state to active applications. This can be done via SNMP, as proposed in [1]. To avoid causing excessive overhead to the router, the execution environment itself is the only entity allowed polling it. Router state variables such as interface load, route table, etc should be cached and shared by active applications. It is also an interesting option to program traps on specific events (e.g. average load exceeds threshold, route update, reboot, etc).
  - **Router-Assistant Protocol**. If processing regular packets at layers equal or higher than 3 is a must, a more specific protocol where the router must play a more active role is required. In this case, the assistant must command the router to divert specific flows, or to output packets over specific interfaces, like in reliable multicast applications.

- **Routing is a router's task**. Having into account that a router's primary function is routing and that its consistency is tightly bound to other active and non-active router's operation, it seems dangerous to delegate this function on a concurrent active entity (unless no other routing protocol is running). Applying the principle that, by default, active traffic should not affect regular traffic, it seems sensible not to delegate such a dynamic (and in many ways uncontrollable) function to an external device. Clearly, the assistant should be reported of route changes, but it should make use of tunnels to override the default routing when required, for example to implement traffic engineering procedures.
- **Active Node Location Transparency**. The reasons why this functionality is important and how it can be taken to practice efficiently in IP have already been described in the previous section. The usage of a specific router-alert value to mark active packets is claimed both for IPv6 (standardized) and IPv4 (value not yet reserved for Active Networks). Active packets are diverted to the assistant despite their destination address.
- **Processing of non-active flows.** There is a second reason behind transparency: in principle, any regular flow (not marked as active) could be programmed by the end-user or network manager to be applied a special processing inside the network (thus becoming active) with the help of the router-assistant protocol. This belongs to conformance level 2. Due to scalability reasons, this per-flow feature would only be suitable for edge routers.

- **Dynamic Code Download.** Any code loading/execution approach is feasible in this system. However, the recommended method from our viewpoint is active packets carrying references to code (and security credentials). If the code is not loaded yet it can be retrieved by existing methods (e.g. https) from code servers or by proprietary protocols.
- **Standard Security Methods.** Standard resources such IPsec and SSL can be used to guarantee the basic security objectives, although a careful study of performance implications is work to be done.
- **Safety checked in advance**. From the authors' viewpoint, it does not seem realistic to have any end-user load untrusted new code into the network in a real environment. Practical experience shows that the scope and expressiveness of easily verifiable programs is rather constrained. Therefore, the approach of only allowing the user to run registered harmless-proofed code on the network seems much more realistic. Thus the presumed target scenario is one in which a central administration provides active services loaded on the fly from a choice of known applications that have been provided by the customer or network manager.

Interestingly enough, the reader can notice that there is a clear symmetry between the concept of router-assistant and the concept of switch controller in IP switching [4]. In the first case, a router delegates higher level processing on the assistant; in the second, the controller delegates forwarding of a given flow on the ATM switch in order to improve performance or provide QoS. Both require a control protocol to by-pass flows to specialized devices.

The first advantage of this approach is the very low development cost required to integrate a router with a given host-based execution environment featuring location-transparency with a minimum penalty on regular packet forwarding performance. In fact, the minimal solution only requires that packets with the active network router alert option be forwarded to a specific interface. This does not interfere with routing efficiency of non-active packets at all, since all packets are examined for router alert options anyway. A second advantage is that forwarding and active processing get loosely coupled and hence resource control is less critical. Scalability can also be achieved if the concept of assistant is extrapolated to, for example, hashing over a pool of assistants based on source and destination address.

There are also important drawbacks in this approach. Network latency for active packets through such a network made of router-assistant nodes is higher than through routers with native AN support. This can be kept under control if a dedicated high-speed LAN is used. More importantly, the loose coupling between router and assistant hinders those applications that require either real-time knowledge of a router's variable (for instance, the output load per millisecond at a given interface) or real-time manipulation of hardware resources (for instance, replace the router's queue management or buffering allocation algorithms). Such hard real-time procedures and low level control mechanisms are clearly out of the scope of this solution, as is the case for most existing AN platforms, although there exist relevant proposals aimed at this ambitious goal based on a common abstraction of a router's hardware [5]. From our viewpoint, the main axiom to be followed by AN is: do not (even partially) replace the router functionality, just enhance it.

4 SARA (Simple Active Router Assistant)

SARA [6] is an active node prototype developed in JAVA to study the router-assistant paradigm. As explained above, the system is capable of transparently processing active packets passing through the router (packets carrying the router alert option with the AN value). The current public release supports IPv4 and IPv6, featuring full packet control by active applications, and it is router-assistant level 1 compliant.

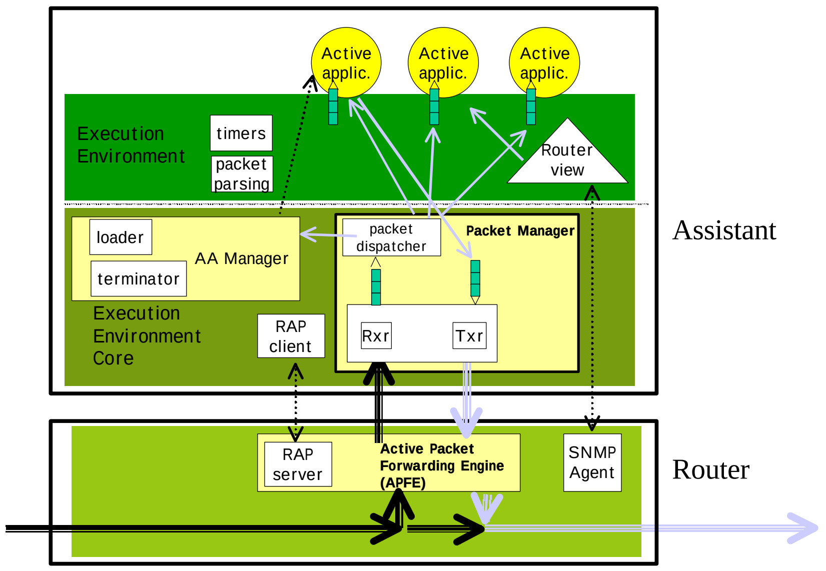

Fig. 3. SARA Architecture

4.1 SARA Execution Environment

The execution environment is composed of two functional blocks (see figure 3): the execution environment, visible to active applications, and the EE core. Actually, this second module featuring the role of OS controls the whole system. Besides managing applications, the core dispatches packets as they are filtered out over the assistant by the real router (at the bottom) on their way to their destination.

To sum up, this simple EE is implemented by the following modules:

- *Timers Module*, that services code execution at a given time. With this, applications have a common easy way to manage timed procedures like retransmissions, etc.
- *Packet parsing*. Its purpose is to ease the analysis of the different headers to the core and active applications. Access to IPv4/IPv6 and UDP headers is provided. It can be easily extended with new protocol header parsers. This could be the case of Active Application headers.
- *Router view*. This feature was also proposed in [1] to support network management sessions. Active applications running at the assistant can have a shared view of the router state thanks to a cached SNMP view of the router. Note that this element can be very important, since the assistant has to process packets on behalf of the router. Interface statistics (load,drops,queue length,etc) and route table are the most common objects used by active applications. Caching reduces the overhead of SNMP gets on the router. As data age is essential to get a proper view of the router state, a flexible mechanism to obtain data with a maximum age or to simply override a cached object value is also provided. Traps are also an optional facility that can trigger a set of complex operations under circumstances where a remote management station would react too late; this is specially useful in applications related to traffic engineering or congestion control.
- *Active Application manager*. This module is in charge of controlling all active applications in the EE. It includes the dynamic code download module, which is invoked when the first packet of an application

arrives to the EE, and the terminator module which is invoked when an application time-to-live expires without being refreshed. Each application has a list of alternative URLs its java code can be downloaded from associated in order to improve reliability. The manager also issues code access control.

- *Packet manager*. This module is in charge of forwarding packets to the target applications and to the next router once processed. Active packets include an Active Application Identifier in the SARA header. This identifier is used by the packet dispacher to deliver packets to the target application. Asynchronous buffers are used for communication between the different components.

4.2 End-Systems architecture

SARA active packets are just UDP datagrams with a Router Alert IP option. UDP payload must carry the SARA header, showed at figure 4, and any AA data.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Active Application Id |            AA TTL             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 4. SARA header

Thus, end-system applications requesting active services make use of UDP sockets enhanced to set the Router Alert option and to add the SARA header, by means of an *ActiveSocket* JAVA class. ANEP can be optionally used to make it coexist with other execution environments.

4.3 SARA prototypes

Two testing platforms are available today. One fully based on linux (playing both roles: router and assistant as a development scenario) and a hybrid platform where the router used is an Ericsson-Telebit AXI462 running a prototype kernel adapted to interwork with an active assistant. A main goal of this platform currently under conformance level 1 is to demonstrate that it is possible to build an active network platform based on commercial routers without a significant drop of performance on regular packets. The most important obstacle to achieve this objective is the limited communication facilities of JAVA. The lack of support for: IPv4 options (router alert options control), IPv6 (very recently released), and layer 3 control to manipulate packets being routed, implied that packets should be encapsulated into UDP messages from the router to the assistant. This was necessary to make available the full packet (header included) to the active applications. Consequently, an important design decision was taken: enrich the standard JAVA communication facilities with IPv4 and IPv6 raw socket support, by making a JNI extension in C/linux. This means a bit of loss of portability for the sake of two aims: cause a minimum overhead on the router (and on the assistant) and offer an enhanced communications API available to active applications. The cost is worth it, as the preliminary tests show, since the router simply has to divert the active packets. This avoids encapsulation tasks typically executed on main processors.

4.4 Sample Applications

In order to test the behaviour of this node, several simple active applications have been developed in SARA. One of these applications is called *a-clink* [7]. This tool is a path characterization tool based on the freely available *clink* tool enhanced by active support to improve the efficiency and accuracy of estimations yielded by existing end-to-end performance estimation tools such as *pathchar, pchar, clink* and *nettimer*, by using active network support. The experience shows that SARA provides good support to the automatic deployment of code along a path and enough flexibility to modify any field of a packet crossing the network.

Another important practical example, developed in the context of GCAP, is a multimedia relay for QoS adaptation for a JMF-based videoconference application, that provides ad-hoc static IPv4-IPv6 multicast translation to support a pan-European experiment in January 2002.

4.5 Performance tests

The following paragraphs describe the preliminary performance results of SARA. The target of these tests are: to measure the practical limit of a JVM/linux-based assistant, to show the impact of the assistant on the router's performance and, finally, to issue an advanced analysis of the applicability domain of a node developed in this specific hardware platform and designed with the principles previously defined.

Testbed. All measures and tests have been done in a simple network formed by an AXI462 router and three Linux boxes, one of them acting as router assistant, and the other two as end systems for the communication tested. The PCs are Pentium III 733 MHz with 128 MB and Intel PCI EtherExpress Pro 100 fast ethernet adapters, running a SuSE Linux 7.2 distribution. It has Glibc version 2.2.2 and a 2.4.12 kernel patched with the Usagi IPv6 stack (snapshot 20011015). Sun's J2sdk 1.3.0 for Linux is employed as Java virtual machine. SARA has been developed entirely in Java (Sun J2sdk 1.3.0 for Linux), except for the native communication raw-socket methods.

Test conditions. Measures have been taken six times and average values, excluding best and worst case, have been taken. Tests tried to show real performance of a real active application so no tweaks have been done, all processes have the standard priority and each host was running standard processes and daemons. In each test an end host sent a flow of fixed length packets, until 25 MB of data (including header). Speed is kept by sending packets at regular intervals. Three scenarios have been set:

1. No Execution Environment was running at router (noEE). To test protocol stack performance at end hosts.
2. Execution Environment was running at the router although no active application was running to capture packets (EE). To test execution environment performance forwarding active packets targeted to an unknown AA.
3. Active packets were delivered to a simple active application (AA 1) which just parses and forwards received packets. To test active application base performance.

Packet sizes are in the range from 512 bytes per packet up to 5120 bytes. Speed also ranges from 2.5 Mbps to 90+ Mbps.

Test results. Figure 5 shows the incremental processing overhead caused by the EE and the AAs.

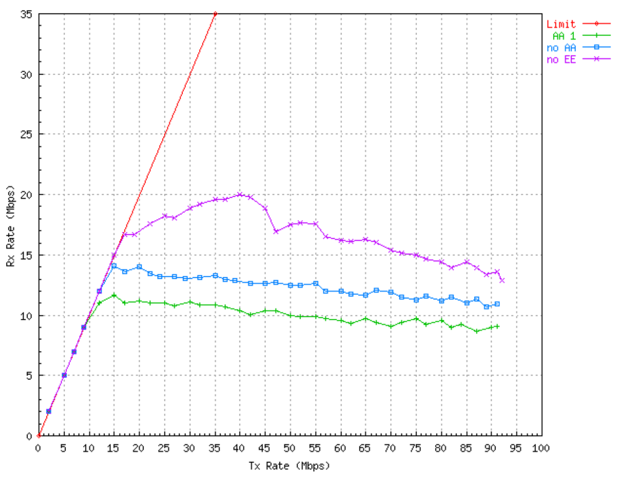

Fig. 5. Effective throughput for EE and AAs with 1024-byte packets

Figure 6 illustrates the effect of active packet size on performance. The relation between packet size and overall performance follows the same pattern as that of a C socket communication, with the logically minor JAVA throughput. As expected, small packet sizes, and hence, more packets per second, decrease throughput a lot. The gain from 512 to 1024 bytes per packet is higher than 100%, the rest get lost in reception. This demonstrates that passing packets from OS to Java is the real bottleneck, so big packets, although subject to fragmentation, perform better, as data exchanges and OS context changes occur less frequently and more work is carried out at kernel level. The AXI462 router suffered no significant load variation during the experiment, due to the moderate rates employed, and the background non-active traffic set through other interfaces perceived a constant throughput.

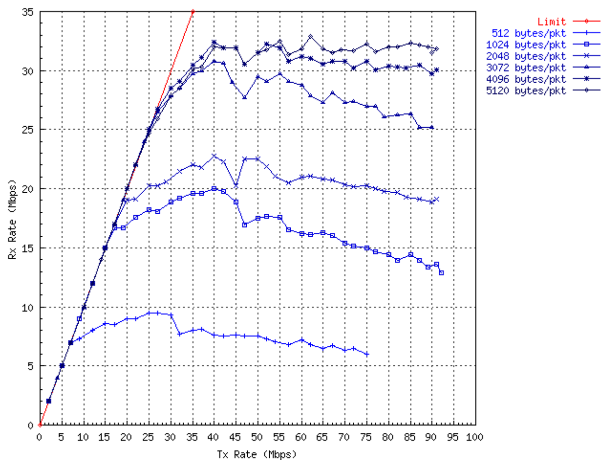

Fig. 6. Effective throughput on different active packet sizes

These preliminary results show that no significant overhead is caused on the router by one assistant, that the performance of Java can be enough for edge router user applications -even multimedia applications- but are constrained to control plane programs on core routers.

5 Related work

There is not enough room in this paper for an exhaustive comparison of related characteristics with existing AN platforms as we deal with complex systems where every design partially relies on features already present in another. To cite but a few, let us focus on the ones that show relationship with the design goals of the router-assistant architecture. ORE [8], Pronto [9] and the presented approach are efforts aimed to offer active behaviour in commercial-grade routers without performance penalty. There are several similarities between Pronto and ORE, both are flow-oriented, provide service transparency, are centered in the NodeOS level, and aim to the same purpose but differ on its approximation to the problem. Pronto is more a set of design principles for new AN devices than a model to be applied to current commercial hardware as ORE is.

The Oplet Run-time Environment (ORE) provides secure downloading, installation and safe execution of Java services, called "Oplets", while controlling the allocation of system resources. It works in conjunction with its Java Forwarding API (JFWD) to control a virtual forwarding path in a platform-independent way. The really important achievement comes from the fact that this platform has been ported to Nortel Networks Accelar Routing-Switches family. The router-assistant approach achieves a higher isolation and scalability of active processing at the price of a lower degree of interactivity between the forwarding engine and the applications.

The Pronto platform defines several "service models" to be chosen by active applications in a tradeoff between functionality and performance. On one extreme a Pronto service could operate purely in the control plane. On the other extreme, the service could reside fully on the data path, processing every packet. There are also two intermediate cases, "local multicast" to send a copy of the packet to the active application and the last one which reduces the bandwidth between the NodeOS and the execution environment by peeking only part of the packet content. These service models allow asynchronous interaction between services and data forwarding. One objective behind the router-assistant approach is going further into this idea by setting apart both functions on different devices, with the added-value of still having a real router processing regular packets. This may not be an advantage anymore if the present linux-based Pronto implementation gets moved to a high-performance router architecture preserving asynchrony. Another convergent point regarding service models is that a “copy”, “partial copy” and “cut” semantics, useful for traffic analysis and full flow control respectively, are also present in the router-assistant protocol for flow diversion.

Regarding the idea of using external assistants, [2] introduces the concept of router *delegate* keeping certain similarities with our approach. In this system, the delegates provides control plane extensibility by controlling how traffic is handled in the data plane through a router control interface (RCI) that enables changing the router’s behaviour per flow. However, unlike in the router-assistant approach, the router itself is in charge of the task of packet processing rather than delegating it on a external processor. Hence, it fits better for applications requiring lower level functions such as bandwidth allocation (functionality that must be present in this more complex RCI), representing a higher implementation cost on the enhanced router. Another difference with this prominent work is that the second type of delegate defined to do CPU-intensive processing -the *data delegate*- is defined as a non-location-transparent server.

The ABLE [1] architecture also proposes separation between router and active engine, SNMP is employed to communicate them, and “blind addressing” scheme offers a kind of network transparency. However, its design is oriented to Network Management domain and it works at application level. Instead, SARA might be located at Network level as it allows manages IPv4/6 headers, although high level applications are also possible. The method to divert active packets is also different as SARA employs the Router Alert IP AN option rather than filtering based on the ANEP UDP ports.

6 Conclusions

Active networking ideas are moving slowly from theory and host-based prototypes to industrial products. This paper has introduced and demonstrated an architecture whose target is to bring a practical set of Active Network functionality to routers in a pragmatic and cost-effective way. The essential feature of this approach is that processing of active packets is carried out in a separate processor in a transparent way to the end user as if it were a router's internal processor devoted to this task (the assistant). This can be achieved thanks to a close cooperation router-assistant based on three functions: active packet by-passing, SNMP-based router state caching –as proposed in [1]-, and an extensible router-assistant protocol.

Compared to existing host-based implementations, the main advantage of this architecture is featuring transparent packet processing of layers 3 to 7 while preserving performance on regular packets, since this latter packets do not have to cross the host running the execution environments. Compared to native AN support in real routers, the advantages of this approach are economy, flexibility, scalability and isolation of AN processing tasks. On the contrary, the router-assistant approach is not suitable for specific applications relying on fine-grained real time information from the router, or processes tightly bound to hardware resources, like packet scheduling, queue management, etc. However, given the entity of the problems implied by the implementation of these QoS routines itself, this type of low level AN interfaces may take a long way to the market.

The architecture here described has been tested on a prototype called SARA running a Java-based execution environment supporting full packet control both for IPv4 and IPv6, in cooperation with the AXI462 router. The preliminary results look promising and show enough flexibility and performance for a fair amount of applications.

References

[1] D. Raz and Y. Shavit. "Active networks for efficient distributed network management." IEEE Communications Magazine, 38(3), Mar. 2000.

[2] J. Gao, P. Steenkiste, E. Takahashi, A. Fisher, "A programmable router architecture supporting control plane extensibility", IEEE Communications magazine. March 2000.

[3] GCAP IST project home page. http://www.laas.fr/GCAP

[4] P. Newman, T. Lyon, G. Minshall, "Flow labelled IP: connectionless ATM under IP", Networld-Interop presentation. April 1996. http://www.ipsilon.com/staff/pn/presentations/interop96.

[5] S. Karlin, L. Peterson. "VERA: An Extensible Router Architecture". IEEE OPENARCH 2001.

[6] SARA home site. http://matrix.it.uc3m.es/~sara.

[7] M. Sedano, B. Alarcos, M. Calderón, D. Larrabeiti. "Caracterización de los enlaces de Internet utilizando tecnología de Redes Activas". III Jornadas de Ingenería Telemática. Barcelona, September 2001.

[8] R. Jaeger, S. Bhattacharjee, J. K. Hollingsworth, R. Duncan, T. Lavian and F. Travostino, "Integrating Active Networking and Commercial-Grade Routing Platforms", 2000.

[9] G. Hjálmtýsson, "The Pronto Platform - A flexible Toolkit for Programming Networks using a Commodity Operating System", IEEE OPENARCH 2000.