A practical approach to network-based processing[1] [2]

D. Larrabeiti, M. CalderÛn, A. Azcorra, M. UrueÒa
University Carlos III of Madrid
{dlarra,maria,azcorra,muruenya}@it.uc3m.es

Abstract

The usage of general-purpose processors externally attached to routers to virtually play the role of active co-processors seems a safe and cost-effective approach to add active network capabilities to existing routers. This paper reviews this router-assistant way of making active nodes, addresses the benefits and limitations of this technique, and describes a new platform based on it using an enhanced commercial router. The features new to this type of architecture are transparency, IPv4 and IPv6 support, and full control over layer 3 and above. A practical experience with two applications for path characterization and a transport gateway managing multi-QoS is described.

1. Introduction

Most industrial experiences that have tried to apply active network concepts to enhance the functionality of real routers have shown that the routerís CPU is not the best place to run an opened execution environment and active applications (see, for instance, [11] demonstrating ANTS [13] onto the Accelar routing switches). The main drawback is that the overall performance figures drop dramatically due to the extra interruption rate induced by active processing. One valid, yet costly, alternative is the usage of ad-hoc active co-processors devoted to this task. Thus, in a distributed processing architecture, the router CPU would be in charge of routing protocols and system management, interface processors would concentrate on fast packet forwarding, and finally, active co-processors would run active applications. However, given the broad scope of active applications to be supported, only general-purpose processors could issue the required versatility. Another factor to consider in the design of this new generation of routers is the active processing capacity needed. One estimation could be given by the assumption that the minimal capacity expected by the users would be at the same level as the latest general-purpose processors in the market . In this case, however, even the best designed modular and expandable internal architecture would not be able to integrate but a single generation of processors without major structural redesign [10].

Under these assumptions it might be more convenient to locate the active processors on hosts directly attached to the router through a high-speed local area network. In this way, the upgradeability and availability of active processors appears to be guaranteed, and packet forwarding and active processing functions are kept decoupled. This paper defines a framework that develops this idea and shows an implementation whose industrial applicability is being assessed.

2. The router-assistant paradigm

With the purpose of defining a pragmatic framework to supply basic active networking functions, valid both for IPv4 and IPv6, and realistic in an industrial context, a new active networking architecture has been developed in the context of the IST project GCAP [3]. This framework referred to as ìrouter-assistantî architecture is based on a set of design principles selected according to its industrial applicability. These principles, and the rationale behind them, are the following:

1. **Routers divert active packets.** In other words, active processing is outsourced. Every router wishing to enhance its functionality with active extensions can delegate this task to a host called Assistant (or, in general, a pool of hosts), directly attached to the router over a high-speed LAN. This assistant runs the execution environment and OS supporting the active applications. Hence, active applications may not run on the router hardware and, consequently, the performance penalty caused to the router is bound to the cost of identifying and diverting active packets, despite their processing cost (hard to predict in comparison with forwarding). Therefore, a primary conformance requirement is as simple as requiring the router to divert all active packets traversing the router

[1] Most of 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. Further development and application to practical scenarios is being supported by IST project Opium (Open Platform for Integration of UMTS Middleware) IST-2001-36063 and the Spanish MCYT under projects TEL99-0988-C02-01 and AURAS TIC2001-1650-C02-01.
[2] Disclaimer: this paper reflects the view of the authors and not necessarily the view of the projects referred.

to the assistant, except those coming from the assistant itself.

2. **The assistant host processes active packets**. A packet diverted from the router must be transparently input to the execution environment, processed at layer 3 or above by the active applications running on it. This processing ranges from cpu-intensive complex payload transformations to simple IP address translations, packet filtering or copying. Once processed, packets coming from the assistant(s) must be forwarded by the router just as any regular packet.
3. **Router and assistant are lightly coupled**. For the purpose of running active applications on behalf of the real router, the assistant must cooperate with the router. This cooperation is based on three functions: active packet diversion, router state communication and Router-Assistant Protocol (RAP) with regular packet diversion. However, it is important to note that there is a trade-off between functionality available to active applications and processing overhead. In order to permit a several degrees of coupling, 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 in item 1. Notice the parallelism found between flow diversion to active applications and the flow diversion mechanisms found in control protocols that set up per-flow forwarding short cuts in MPLS antecessors [4].
   - **Router state view**. A procedure by which the execution environment makes available the router state to active applications. This can be done, as proposed in [1] for management-oriented applications, via SNMP. To avoid causing excessive overhead to the router, the execution environment itself is the only entity allowed to poll it. Router state variables such as interface load, route table, average queue length, etc are 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) to trigger asynchronous cache updates. It is true that the usage of SNMP may cause a considerable processing overhead on the router even though its impact is regulated by the execution environment. Nevertheless, note that this conformance level 1 feature allows a primary and ubiquitous access to the state of the legacy router at no implementation cost. Hence, it should not be disregarded so easily, and it is adopted as a transition mechanism that can ease the router activation process. A more efficient state conveyance mechanism based on the router-assistant protocol is under study.
   - **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. These and other control functions enriching the communication router-assistant are encapsulated in this protocol. RAP runs on top of TCP and extends the API provided by the execution environment with flow manipulation. Flow CUT and COPY commands allow the design of applications that alter or simply snoop a flow respectively.
4. **Routing is essentially a routerís task**. An execution environment defined upon this architecture can easily provide active applications with the capability to change the route table. However, having into account that a routerís primary function is routing and that its consistency is tightly linked to the existing routing protocol, it seems not advisable to delegate this function on a concurrent active entity (unless no other routing protocol is running and a specific active application is devoted to this task). Furthermore, the risk of instability grows if non-active nodes are present in the network, as it is normally the case. Therefore route changing is made available to applications with specially assigned privileges and is not a recommended practice. Nevertheless, the route table is likely to be an essential information for an active application; therefore the assistant must be reported of route changes by the router and thereby, the applications subscribed to these notifications. Consequently, active applications should make use of tunnels when they need to override the default routing, for example to run traffic engineering procedures or to set up an overlaid logical network of active nodes .
5. **Transparent processing of active packets**. The IP address of the Active Nodes in the path(tree) source-destination(s) can be transparent to the end users. This way an active packet can automatically launch the active applications needed to deploy a given service in all the active nodes traversed, and forget about choosing the right active servers and be unaware of network topology. For instance, an active packet can fetch enough path information to determine the best node to locate a TCP spoofing active entity or any other transparent gateway one can imagine.

   To take this feature to practice efficiently, the usage of a specific router-alert value to mark active packets is recommended for IPv6 (standardized in RFC-2711) and IPv4 (non-standard approach, as this value is not yet reserved for Active Networks in RFC-2113). This way active packets are easily recognized and diverted

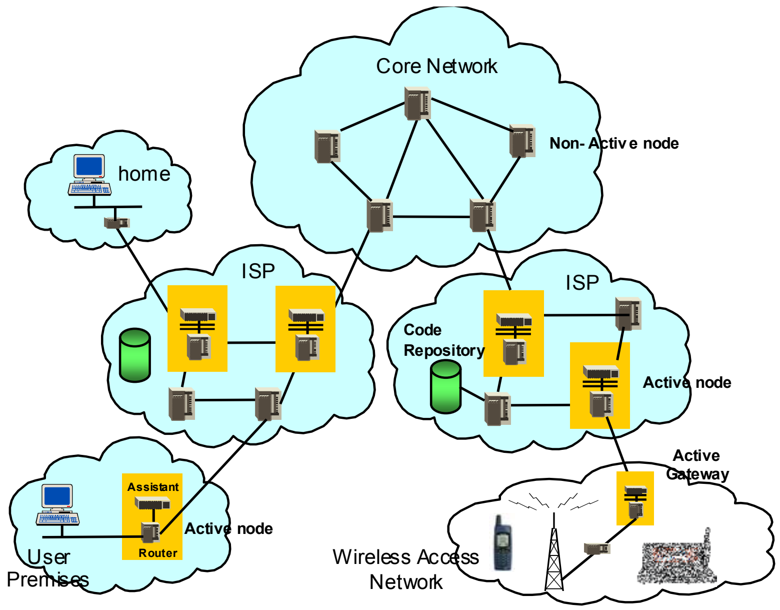

Figure 1. A target application scenario

to the assistant despite their destination address. In level-2 compliant implementations, transparency is extended to regular packets.

Transparency is not compulsory. A user can address a specific network node, and active applications, once started may open their own sockets to communicate with other active nodes or end systems, employing explicit addressing. In this case, the usage of multicast incremental ring searches or scout packets can be useful to locate the nearest active node and establish neighborhood.

6. **Transparent processing of non-active flows.** 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 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.
7. **Dynamic code download and persistency.** Any code loading/execution scheme is feasible in this system. However, the recommended method 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.
8. **Standard Security Methods.** Standard resources such IPsec and, mainly, SSL can be used to guarantee the basic security objectives of this framework, including authentication, confidentiality and even non-repudiation. However, there is also an interesting trade-off between scalability and security to be studied in another work.
9. **Safety checked in advance**. From the authorsí viewpoint, it does not seem realistic to have any end-user load just any 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 permitting 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. This is in fact a networked application service provision (ASP) with many analogies in practice, like the Intelligent Networks model. Another interesting analogy comes from the comparison to the Unix ô inetd, where a set of trusted servers are launched on demand.

Application scenario

To give an idea of the concepts described above and its context of applicability, figure 1 shows the internal structure of an active node, composed of an enhanced router and an assistant, as envisaged by the authors. Validated (safe) network services are available at replicated secure repositories whose URL is configured in advance by the active network manager. The figure also shows the most coherent location for these programmable nodes: at edge routers. This way

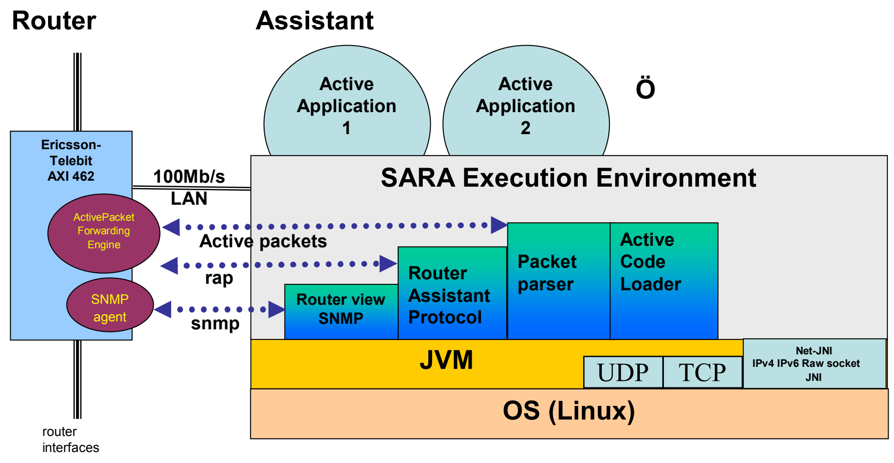

Figure 2. Overview of SARA software structure

active nodes can help users launch new network services (content filtering, content pushing, secure tunneling, multimedia relays and translators, etc) on demand in a scalable way to add on existing ones deployed statically (NAT, firewall, transparent proxies). Such scheme is specially suitable for wireless networks, where services must keep moving close to mobile terminals: to enhance their capabilities (storage, computing and displaying capacity), to improve latency (by breaking end-to-end retransmissions, caching, etc), and to optimize link usage (e.g. pushing, trans-coding), goals difficult to implement via server-based approaches.

Discussion

The first advantage of the router-assistant 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 limitations in this approach. Network latency for active packets through such a network made of router-assistant nodes is higher than through routers with native active support. This can be kept low enough for most applications if a dedicated very high-speed LAN is used and the assistant load is kept low itself (or the execution environment is built upon a real-time OS). More importantly, the loose coupling between router and assistant hinders those applications that require either real-time knowledge of a routerís variable (for instance, the output load per millisecond at a given interface) or real-time manipulation of hardware resources (for instance, take over the routerís 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 active network platforms, although there exist relevant proposals aimed at this ambitious goal based on a common abstraction of a routerís hardware [5].

From our viewpoint, the main axiom to be followed to push forward active functions is: do not (even partially) replace the router functionality, just try to enhance it.

3. An implementation : SARA (Simple Active Router Assistant)

SARA [6] is an active node prototype developed in JAVA (and C) 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 Aactive Network value). Active packets are diverted by the router to the assistant (with no extra encapsulation) where they are analyzed. If the target active application is not currently loaded on the execution environment, it is downloaded from one of a set of code servers and run for a lifetime determined by the active packets. Active applications are (native) threads and can make use of active packet parsing and sending facilities, read any of the router state objects available in the mib-2 cache with a specified maximum age, use enriched socket functions, etc

The current public release supports IPv4 and IPv6, featuring full packet control by active applications, and it is router-assistant level 1 compliant.

One of the most relevant implementation problems met during the development of SARA is the limited communication facilities of JAVA. The lack of support for: IPv4 options (router alert options control), IPv6 (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. In order to save this burden to the router, the standard JAVA communication facilities were enhanced with IPv4 and IPv6 raw socket support, by making a JNI extension in C/linux (see Figure 2).

Two prototype 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 an enhanced kernel adapted to interwork with an active assistant. A main goal of this platform is to demonstrate that it is possible to build an active network platform based on commercial routers at low cost and without a significant drop of performance on regular packets.

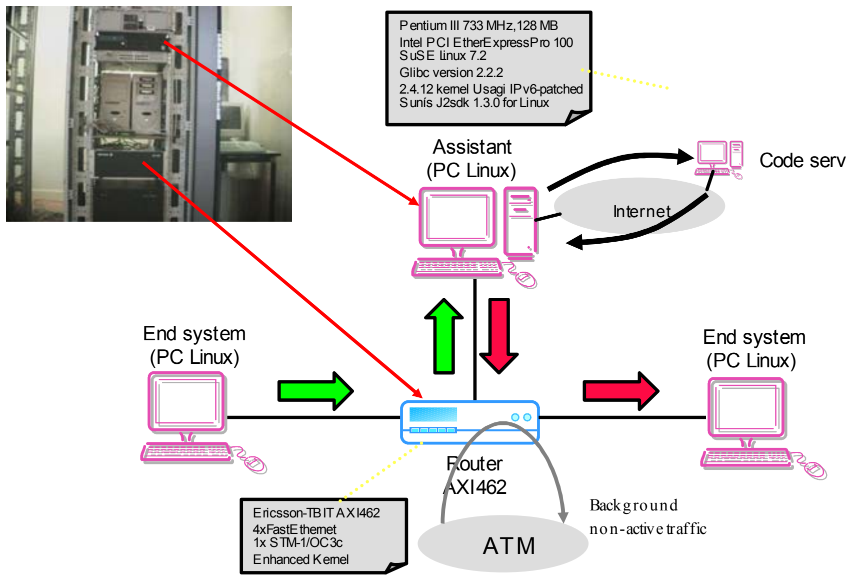

Figure 3. Test Scenario

Figure 3 shows a testing scenario set up to analyze the maximum throughput achievable with the prototype execution environment. Packet flows were generated by an end station, diverted by the router, parsed/registered on the assistant, and forwarded to a receiver PC. The experience showed that the real bottleneck of the system was in JAVA, concretely in the low performance of the communications interface to Linux. The proof of this fact was the performance gain obtained for using packets of a size over the MTU. In this latter case, the kernel takes up most of the communication processing work, improving the effective throughput (see results in Table 1, obtained for the particular hardware components described in figure 3). In these conditions, the impact on the routerís performance was not significant.

Table 1. Effective throughput vs packet size

Packet size (Bytes)	Effective throughput (Mb/s)
512	10
1024	20
2048	22
3072	28
4096	31
5120	33

As a conclusion, from our point of view, JAVA is excellent as a platform-independent language to make universal active code, but in general it is not appropriate to process multimedia data flows generated by sets of users (see also [11]). However, we believe that JAVA is still the right choice for the control plane of an active network architecture, i.e. JAVA is perfect to process exclusively active packets, to retrieve the active code and drive its execution. Hence, part of this code must look like another JAVA active application to the execution environment, but most of it must be native code in order to process regular packets at the required data rates. Finally, it is important to note that Linux may not be the best platform to run JAVA.

4. Sample applications

In order to test the behaviour of the node described, several simple active applications were 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 purpose of *a-clink* is obtaining per-link bandwidth estimations based on probing with different-sized packets sent from active nodes, rather than from the source host; this minimizes the accumulated error and causes less probing traffic. 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 [3], is a multimedia relay for a JMF-based videoconference application, that provides ad-hoc static IPv4-IPv6 multicast translation that supported a pan-European videoconference experiment in January 2002.

A third application tested is an intelligent QoS adaptor based on entities deployed transparently along the path server-receiver. This type of applications are getting more and more important in n-to-n multi-QoS multimedia flow services 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 the usage of layered coding and multicast each layer on a different group, 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.

5. Related work

There are many important works demonstrating active networks on host-based execution environments and on router-based platforms. Let us cite the ones that keep some design principles in line with the ones presented in this paper or that we consider a good starting point to solve problems not addressed by our architecture.

The ABLE [1] architecture also proposes separation between router and active engine, SNMP is employed to communicate them, and ìblind addressingî 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.

[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 routerís 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.

A development of the idea of creating a lower level abstraction of router, compared to the one needed by an external device as SARA, can also be found in [5]. In our context, controlling the routerís hardware makes only sense for very specific applications, not for our target execution level where applications are supposed to be independent and do not alter the normal behaviour of the router, except for the functional treatment of certain flows.

Finally, [10] defines an architecture based on NetBSD that allows code modules, called *plugins*, to be dynamically added and configured at run time, and provides transparency, like [12]. One of the interesting features of this design is the ability to bind different plugins to individual flows. A similar concept is found in SARA, where active applications request the router a specific flow to be diverted and processed by them. SARA is not specially designed to support quality of service, unlike [10], but, in contrast, it can provide transparent active processing and isolation from regular packet forwarding tasks at network points where a real router is necessary.

6. Conclusions

This paper describes a pragmatic approach to active networking that brings a practical set of Active Network functionality to routers in a pragmatic and cost-effective way. Compared to existing host-based implementations, the main advantage of this architecture is the capability to provide transparent packet processing from 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 Active Network support in real routers, the advantages of this approach are economy, flexibility, scalability and isolation of active processing tasks. The architecture here described has been tested on a prototype called SARA (Simple Active Router Assistant) running a Java-based execution environment supporting full packet control both for IPv4 and IPv6, in cooperation with an Ericsson-Telebit AXI462 router. The preliminary results look promising and show enough flexibility and performance for a fair amount of applications.

6. Acknowledgements

The authors wish to thank J. E. Kristensen, L. Kroll Kristensen, and R. J□rgensen from Ericsson-Telebit for their comments on the definition of the router-assistant architecture, and the implementation of the AXI462 router software extensions to support an external assistant. Many thanks also to M. Sedano and B. Alarcos (U. Alcal·) and D. Garduno (LAAS/CNRS) and E. Exposito (Ensica), for choosing SARA to develop their active applications and their useful feedback.

7. References

[1] D. Raz and Y. Shavit. ìActive networks for efficient distributed network management.î *IEEE Communications Magazine*, 38(3), March. 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, ìFlow labelled IP: connectionless ATM under IPî, Networld-Interop presentation. April 1996. http://www.ipsilon.com/staff/pn/presentations/interop96.

[5] S. Karlin, L. Peterson. ìVERA: An Extensible Router Architectureî. IEEE OPENARCH 2001.

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

[7] M. Sedano, B. Alarcos, M. CalderÛn, D. Larrabeiti. ìCaracterizaciÛn de los enlaces de Internet utilizando tecnologÌa de Redes Activasî. III Jornadas de IngenerÌa Telem·tica. 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. Hj·lmt˝sson, "The Pronto Platform - A flexible Toolkit for Programming Networks using a Commodity Operating System", IEEE OPENARCH 2000.

[10] D. Decasper, Z. Dittia, G. Parulkar, B. Plattner . ìRouter Plugins. A Software Architecture for Next Generation Routers.î SIGCOMM 1998.

[11] Tal Lavian, Phil Yonghui Wang. ìActive Networking On A Programmable Networking Platform.î IEEE OPENARCH 2001.

[12] D. Scott Alexander, et al , ìThe SwitchWare Active Network Architectureî, IEEE Network Special Issue on Active and Controllable Networks, vol. 12 no. 3, July 1998

[13] D. J. Wetherall, J. Guttag, and D. L. Tennenhouse, ìANTS: A Toolkit for Building and Dynamically Deploying Network Protocolsî, IEEE OPENARCH'98, San Francisco, CA, April 1998.