Dynamic Deployment and Configuration of Differentiated Services Using Active Networks

Toshiaki Suzuki[1], Chiho Kitahara[2], Spyros Denazis[3], Lawrence Cheng[4], Walter Eaves[4], Alex Galis[4], Thomas Becker[5], Dusan Gabrijelcic[6], Antonis Lazanakis[7], and George Karetsos[7]

[1] Hitachi, Ltd., Central Research Laboratory, Japan
toshiaki@crl.hitachi.co.jp, Tel:+81-42-323-1111, Fax:+81-42-327-7868
[2] Hitachi, Ltd., Systems Development Laboratory, Japan
kitahara@sdl.hitachi.co.jp, Tel:+81-44-959-0266, Fax:+81-44-959-0853
[3] Hitachi Europe Ltd., Hitachi Sophia Antipolis, France
spyros.denazis@hitachi-eu.com, Tel:+33-4-89-87-41-72, Fax:+33-4-89-87-41-51
[4] University College London, United Kingdom
{l.cheng, w.eaves, a.galis}@ee.ucl.ac.uk, Tel:+44-20-7419-3946, Fax:+44-20-7387-4350
[5] Fraunhofer Institute for Open Communication Systems FOKUS, Germany
becker@fokus.fhg.de, Tel:+49-30-3463-7393, Fax:+49-30-3463-8393
[6] Jozef Stefan Institute, Laboratory for Open Systems and Networks, Slovenia
dusan@e5.ijs.si, Tel:+386-1-4773-757, Fax:+386-1-4232-118
[7] National Technical University of Athens, Greece
laz@telecom.ntua.gr, karetsos@cs.ntua.gr, Tel:+30-210-7721511, Fax:+30-210-7722534

Abstract. This paper presents the detailed components of the Future Active IP Networks (FAIN) [1] [2] [3] active node framework based on the novel Virtual Environment (VE) concept. It also presents the dynamic and autonomic deployment of differentiated services and the configuration capabilities thereof enabled. The FAIN node supports the dynamic deployment and instantiation of multiple active VEs, each one of them capable of hosting multiple Execution Environments (EE) and supporting communication among different EEs in the same node. The EEs may, in turn, be deployed and instantiated on demand thereby introducing new features and functionality in the node according to new requirements and arising needs. We tested the FAIN active network by developing and dynamically deploying a control EE, which was designed and tested for the QoS configuration of the Diffserv-enabled pan-European FAIN testbed [4]. The work presented in this paper was performed in the European Union research and development project under the Information Society Technologies programme.

Keywords: FAIN, Active Node, Virtual Environment, Execution Environment

1 Introduction

In the world of network architectures, we are experiencing a significant paradigm shift resulting in new technologies and approaches. The motivation behind this shift is

N. Wakamiya et al. (Eds.): IWAN 2003, LNCS 2982, pp. 190–201, 2004.

the still elusive goal of rapid and autonomous service creation, deployment, activation and management, resulting from new and ever changing customer and application requirements. Research activity in this area has clearly focused on the synergy of a number of concepts: programmable networks and services, managed networks, network virtualisation, open interfaces and platforms, application level programming and increasing degrees of intelligence inside the network. Next generation networks must be capable of supporting a multitude of service providers that exploit an environment in which services are dynamically deployed and quickly adapted over a common heterogeneous physical infrastructure, according to varying and sometimes conflicting customer requirements.

Programmable and Active Networks have been proposed as a solution for the fast, flexible and dynamic deployment of new network services. These networks aim at providing easy introduction of new network services by adding dynamic programmability to network devices such as routers, switches and applications servers. The basic idea is to enable third parties (end users, operators, and service providers) to inject application-specific services (in the form of code) into the network. Applications are thus able to utilize the required network capabilities in terms of optimised network resources and as such they are becoming network-aware. Programmable networks allow dynamic injection and deployment of code as a promising way of realising application-specific service logic, or performing dynamic service provision on demand. The viable architectures for programmable networks have to be carefully engineered to achieve suitable trade-offs between flexibility, performance, resilience, security and manageability.

In this paper we present the detailed components of the FAIN active node, and the dynamic deployment of differentiated services and the configuration thereof enabled by a novel active and programmable node architecture. More specifically, in section 2 we describe the FAIN active node architecture and its major components. The focus is on the deployment and instantiation of VEs and EEs as host environments for the introduction of new functionality. In Section 3 we present the design and implementation of a secure control EE that is based on the SNMP [11] and hosts a control protocol. In Section 4 we demonstrate the uses of the deployed control EE for the dynamic configuration of a Diffserv network in order to provide the required levels of QoS. Finally conclusions and future work are presented in section 5.

2 FAIN Active Node

Figure 1 depicts the major components of the FAIN programmable node and its interaction with the management node, which includes at both network and element levels an Active Service Provisioning (ASP) and a Policy Based Network Management System (PBNM) components [9]. When the FAIN node boots up a Privileged VE is automatically instantiated and a new component is installed, namely the VE manager, which implements the VE management framework. This component offers access to a number of node services that are deemed necessary to configure and setup the node. It is used for instantiating new VEs together with appropriate EEs and to install components therein which potentially offer new control interfaces that allow services inside VEs to customise resources according to application-specific requirements.

The VE manager is complemented with the security component (SEC) that offers a set of security services and enforces node policies, the resource control component (RCF) responsible for implementing the FAIN resource control framework and the demultiplexing/multiplexing component (De/MUX) which delivers packets to the right VE and EE. In the subsequent sections we describe each one of these components in detail.

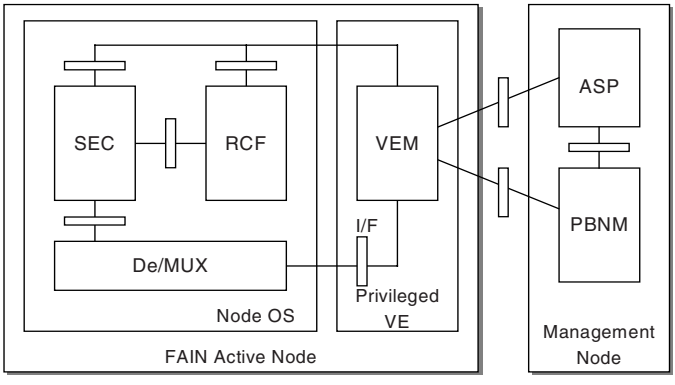

Fig. 1. Initial Components of the FAIN Active Node

In our architecture, multiple EEs can be executed in one VE since the unit for management is VE not EE. Therefore interaction between EEs is allowed. On the other hand, the conventional works for active networks are mainly contributed to make new possibilities in one EE. Therefore interaction between EEs looks out of scope in the past researches. In this sense, the concept of VE is the most crucial idea. Another benefit of VE is related to the accounting. The real VE is composed of multiple resources. Therefore it provides easy accounting based on the resource consumption.

2.1 Virtual Environment Management

As we have mentioned before, services are installed in VEs and eventually instantiated and executed inside the associated EEs. In FAIN, services are described according to a component-based approach. As such a service is defined as a graph of service components, which in turn can be developed and deployed independently as and when needed [3][5]. It is an advantage when the hosting environments for services, i.e. the execution environments are also component-based. This allows to move aspects such as lifecycle management, dynamic configuration, access control, monitoring, etc. to a supporting framework and avoid re-implementation inside the service code. The implementation of the VE management is an example for this.

During the boot procedure of the FAIN active node, the privileged virtual environment is started together with a default execution environment. Any subsequently created virtual environment will need some basic resources in order to support service installation and component instantiation. For this reason various resource managers are installed inside the privileged virtual environment during the boot procedure as it is shown in figure 2.

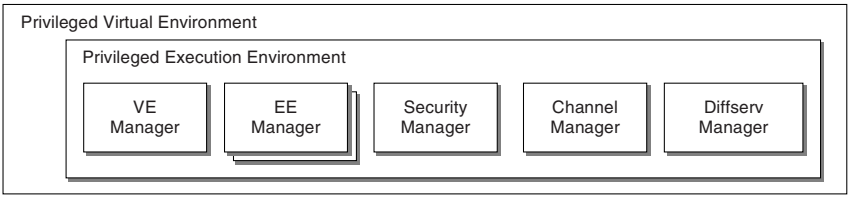

Fig. 2. Initial manager components of FAIN active node

The privileged execution environment runs in the context of the privileged virtual environment. Inside the privileged execution environment there exist the resource managers for the basic services. They will be used to create resources for other virtual environments. These basic resource managers comprise the following:

A Virtual Environment Manager is used for the creation of new virtual environments. This manager will examine the resource profile submitted as part of a VE creation policy and try to create any referenced resource using other basic managers. The resulting resource components will be inherited by the new virtual environment. A number of Execution Environment Managers are used for the creation of specific execution environments. Since running instances of services can exist only inside execution environments, there has to be at least one execution environment attached to any virtual environment. A Security Manager is a core component of the security architecture that exports a minimal set of interfaces to other node subsystems. A Channel Manager is used for creating channels to receive and send packets from and to the network. A Diffserv Manager is used for creating Diffserv controllers to control particular packet flows based on priorities.

2.2 Demultiplexer and Multiplexer: De/MUX

Our framework supports multiple VEs and multiple EEs running in VEs and as such the packets are delivered to the right entity inside a node. To this end packets must carry all the necessary information based on which the De/MUX component may forward the packet to its destination inside the node. In this case, we need to specify both environments, the VE and EE, to execute real processing to active packet data. We have adopted the Active Network Encapsulation Protocol (ANEP) [7] for the FAIN active packet data and extended its definition by introducing two new options: one for the VE identifier and one for the EE identifier.

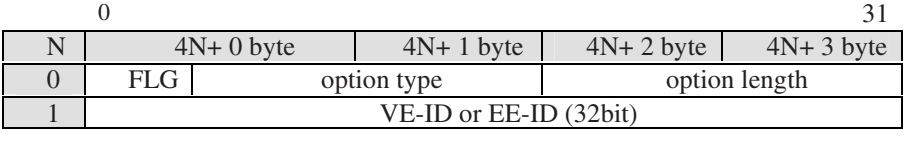

Fig. 3. VE/EE identifier

The formats for two new options are the same as shown in figure 3. The FLG indicates how to handle the option data. The owner of the option defines the value of the

FLG. The option type indicates a type of option. The value of the option type for VE and EE identifiers are 101 and 102 which were defined for the FAIN VE environment. The option length specifies the size of option field in 32 bit words. The value of the option length for the VE-ID or EE-ID is 2 in 32 bit words. The VE-ID or EE-ID indicates an identifier to transmit active packets to a proper VE/EE.

In figure 4, we present the block diagram of packet data delivery.

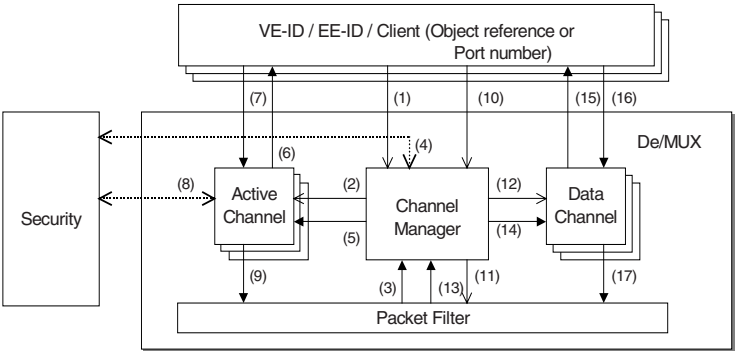

Fig. 4. Block diagram of packet delivery

Active (ANEP) packet data delivery. (1) A client requests a Channel Manager to create a new Active Channel for receiving ANEP packet data by registering a VE-ID, an EE-ID and an object reference of itself or a socket port number. (2) The Channel Manager creates the Active Channel by registering an active consumer object, which includes the VE-ID, the EE-ID and the reference or the socket port number, into an internal table for active packets. (3) The packet filter transmits the received ANEP packet to the Channel Manager since the manager sets conditions to intercept ANEP packets at the booting process. The filter could be implemented by the Netfiler [8]. (4) The Channel Manager calls a security function for checking the ANEP packet before sending it to a proper client. (5, 6) After executing the security check, the Channel Manager sends the ANEP packet data to the proper client through an appropriate Active Channel by getting a target from the internal table. The table includes the relation between VE-ID/EE-ID and the target receiver. (7) If there is an ANEP packet to be sent to another node, the client sends the ANEP packet to the proper Active Channel. (8) The Active Channel inserts the security information into the ANEP packet by interacting with the security component before sending it to the outside network. (9) After that, the Active Channel transmits the ANEP packet to the outside network.

Data (Non-active) packet data delivery. (10) A client requests the Channel Manager to create a new Data Channel for receiving non-active packet by registering flow conditions and object reference of itself or a socket port number. (11) The Channel Manager sets the filter conditions such as a source IP address and so forth. (12) The Channel Manager creates the Data Channel object, which includes the flow conditions and the reference or the socket port number. (13) The filter transmits data packet to the Channel Manager. (14, 15) The Channel Manager sends data packet to a proper

client through an appropriate Data Channel. (16) If there are some packet data to be sent to another node, the client sends them to the proper Data Channel. (17) The Data Channel transmits data packet to the outside network.

2.3 Resource Control Framework

RCF (Resource Control Framework) has been designed by taking strongly into consideration and exploiting the capabilities of the component model that has been introduced in the VE management framework that was previously described. The part of RCF that is responsible for the management of the resources is actually part of the VE Management framework while the run-time control of the resources is done by other lower level RCF components. RCF can be defined as the aggregation of all the FAIN active node components that operate and interact in order to control and manage resources. As it is depicted in figure 5, there are the component families of the RCF.

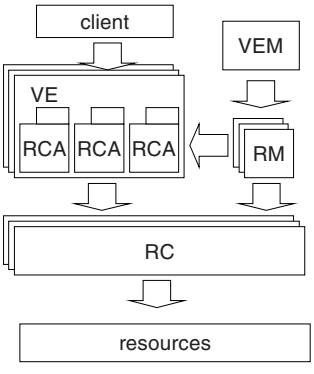

Fig. 5. RCF Architecture

Resource Controller (RC). RC is the responsible entity for the runtime control of a resource inside the active node. RC can be a component running in the Kernel Space of the node for a software router or can be a specific device of a hardware router. It can be the Traffic Control framework of the Linux. Every RC has an interface that allows its runtime configuration, which includes the allocation and monitoring of the resources.

Resource Manager (RM). For every RC, an RM exists in the user space. It is responsible for the configuration of the corresponding RC in order to enforce the resource partitioning among the various VEs. Moreover, the RMs are responsible for the RCAs creation, configuration and management. Among others the RMs are responsible for the Admission Control of the incoming requests for new allocations and for the realization of the allocation by configuring the corresponding RCs.

Resource Controller Abstraction (RCA). For every resource capacity that is allocated to a VE, an RCA of that resource is created. For every VE, a resource controller abstraction (RCA) exists that represents part of the RC functionality to the VE client: the part of the resource that has been allocated to the VE. The RCAs export interfaces and accept requests by VE owners and/or users for resource access. Resource access includes requests for resource consumption and management. RCAs check those requests against resource status and the requested entities' privileges and enforce the valid requests by configuring the corresponding RCs accordingly.

In FAIN we adopted a two-phase approach for admission control, namely, the creation phase and the activation phase. We note here that the creation and activation of a VE is part of a larger activity, which results in the creation, and activation of a virtual network across the entire active network. During the creation phase, the client requests the creation of a new VE. The VEM passes these requirements to the corresponding RMs. Every RM decides if the requested allocation can be carried out or not. When the VEM has collected all the replies from the RMs, it decides if the new VE may be admitted or not. If any of the RMs is unable to provide the requested resource capacity, the admission of the VE is aborted while the VEM informs all RMs that they have to release any pre-allocated resources.

When all the replies from the RMs are positive, the VEM replies positively as well. But even then, the VEM does not activate the newly created VE and the resources remain preallocated. If the creation of all VEs across the entire active network nodes has succeeded, only then the newly created VEs are activated. The activation request arrives at the VEM from the Network management station, which collects and checks all the responses from the VEMs of the nodes across the active network. In this case, the VEM gets in contact with all the involved RMs in order to activate the RCAs, configure the RCs accordingly, and enforce the appropriate resource allocations.

We have implemented two types of resource controller. The one is a bandwidth controller of a software router based on Linux. The other is a controller of the Diffserv functions of a hardware router. In this paper, the Diffserv control and management are presented as an example of the RCF implementation. The Diffserv controller and the hardware router are regarded as the RCA and the RC respectively.

2.4 FAIN Security Architecture

We propose high level security architecture as shown in the figure 6. Basic security services are positioned in the privilege VE because of the following reasons: we want to treat all possible technologies and their implementations, implementing VE and services in the one and only one manner, reducing the risk of multiple implementations, and the services offered in the pVE are protected again with the same services and mechanisms. This doesn't preclude VEs or services from implementing their own security services or mechanisms when it is reasonable to do so.

The FAIN security architecture was designed as a complete security solution for programmable and active node. It provides two level communication security, authorization and policy enforcement on the node, static and dynamic code verification, system integrity and accountability through audit service and logging. Only the first three aspects will be briefly described.

Two level communication security is provided for active packets with hop-by-hop protection in between two neighbor nodes based on symmetric cryptography and end-to-end authentication of data origin of the static parts of the active packets based on asymmetric cryptography. Two ANEP options were designed to support two level protection: hop-by-hop option, with security association identifier, replay protection field and keyed hash, and credential option with credential filed, type and location, optional timestamp and target and digital signature field. Multiple credential options can be in the packet related to different users, which digital signatures cover static parts of the packet. Communication security is supported with a protocol for automatic establishment of security association between nodes and protocol for exchanging user credentials between nodes in a hop manner. For efficient operation credentials are cached on nodes. Management sessions to nodes use CORBA over SSL and credential used (X.509v3 certificates with extended attributes) were handled in the same manner as user credentials supplied in the active packet.

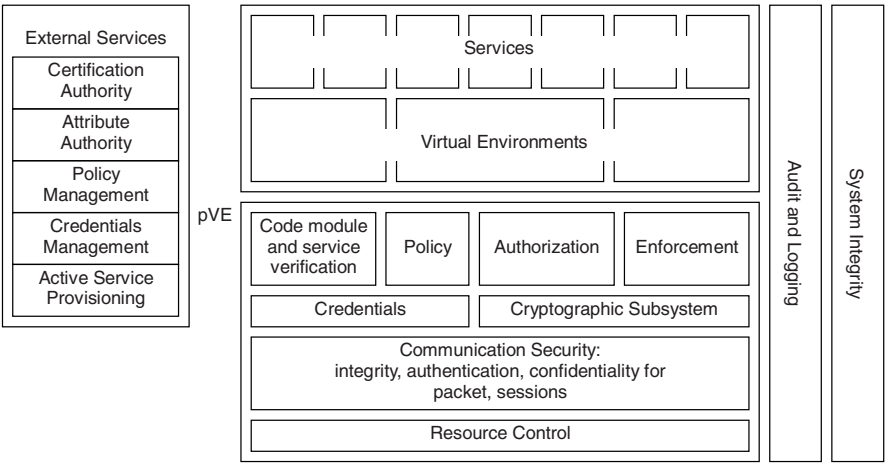

Fig. 6. High level security architecture

Authorization and policy enforcement on the nodes is provided by transparent enforcement layer, based on CORBA interceptors, components/packets security context and two level policy. Security context is build from user supplied credentials either in active packet or exchanged during SSL session negotiation. Low level policy is mandatory and it is enforced based on VE and service/EE identifiers, while high level policy allows fine grain discretional control. Multiple types of security policies can be supported by the system; we have implemented a simple one based on user roles like VE manager, manager, observer, user etc. Enforcement layer enables us to control access to the level of component interface or port while component run time instance can be a thread or a process. Static Code verification is based on digital signature mechanism while dynamic verification, like in SNAP case, see section 3., was enabled with separation of variable data (SNAP packet) and its static part, data origin

authentication, same as used for communication security, and code verification performed by ASPE, see section 3.2.

3 A Secure Control EE (Active SNMP-EE)

The Active SNMP-EE is the control EE in FAIN. This EE is realized with the Safe and Nimble Active Packets (SNAP) interpreter [6][10][12]. The Active SNMP-EE consists of two components: a) SNAP Activator that generates SNAP packet programs with various SNMP commands; b) ANEP-SNAP Packet Engine (ASPE) that provides ANEP encapsulation and security provisioning for SNAP active packets across nodes. In the Active SNMP-EE, the active extensions were realized by using an extensible SNMP [11] agent. The Active SNMP-EE is able to execute SNMP primitives. The SNAP Activator generates SNAP packet programs that carry SNMP commands. SNAP packet programs are encapsulated into ANEP for the purpose of integration and interoperability. Further details of the SNAP Activator can be found at [1]. Noted that SNAP is a light-weight protocol, it has no facility for authentication at all. The ASPE works together with the Security Manager to provide the necessary security facilities to protect the end-to-end and hop-by-hop authenticity of SNAP packet programs generated by the Active SNMP-EE.

In active networks, hop-by-hop authentication should be included as well as end-to-end authentication. Principle authentication must be performed at an intermediate node since the traversing active packet will be modified. In order to enforce both end-to-end and hop-by-hop authenticity of active packets, we determine the static data of SNAP packet programs, and then encapsulate these data into ANEP separately from the SNAP dynamic data. We define the SNMP commands that are carried in the SNAP packet programs to be the static data; whereas the dynamic data is the SNAP packet itself (SNAP consists of a heap and a stack which are used to carry variable data) [13]. The eventual goal is to merge the ASPE with the SNAP Activator so that ANEP encapsulation can be performed whilst SNAP packets are being generated. Our current implementation fingerprints the SNAP packet (which contains both static and dynamic contents) before enforcing hop-by-hop protection. This static field will be digitally signed by the principle's private key (the digitally signing is performed at SEC), and the signature will be verified by each of the intermediate modifying nodes. The SNAP packet program is encapsulated entirely into Option-5. When an ANEP-SNAP packet arrives at a hop node, the SNAP packet will be extracted from Option-5, subjected to integrity and authenticity check by SEC. We use symmetric cryptographic techniques. Under this arrangement, neighboring hops would have established a trusted relationship among themselves i.e. by creating a negotiated security association (SA). With a SA, neighboring nodes can achieve peer authentication plus inter-node integrity and confidentiality protection of active packets.

3.1 Packet Flow in the ASPE

In figure 7, a block diagram of the ASPE is depicted. (1) The SNAP Activator generates a SNAP packet. (2) The SNAP Analyser determines the VE-ID, EE-ID, destination IP address, the SNAP Packet ID and the SNAP static command from the SNAP

packet. (3) The Communication Manager uses the SNAP Packet ID as a reference to extract the corresponding Security ID (SID) in the Option-4 of this SNAP packet from its database, if no SID is found then this SNAP packet will be treated as a fresh packet. (4) The Digester provides additional internal integrity check for the SNAP packet. (5) The SNAP Encapsulator encapsulates the SNAP packet and the SNAP static command into ANEP Option-5 and the Payload field respectively. The SNAP Encapsulator then assigns the VE-ID, EE-ID, destination address and the SID of this SNAP packet to Option 1 to 4 respectively. (6) After that the ANEP-SNAP packet is transmitted to a local SEC for security provisioning before being forwarded to its next hop. (7) When an ANEP-SNAP packet arrives at its next hop, the De/MUX dispatches the packet to the SNAP De-Encapsulator after successful security checks performed at SEC. (8) The SNAP De-Encapsulator extracts the SNAP packet Option-5. (9) The Digester performs internal security checks for Option-5. (10) If the destination of the ANEP-SNAP packet is not local then the SNAP Analyser will extract the SNAP packet ID and the Security ID from the SNAP packet and Option 4 respectively, (11) The Communication Manager keeps this SNAP packet ID-SID pair in its database. The ID pair is needed by the SNAP Encapsulator for future ANEP-SNAP encapsulation. (12) The SNAP De-Encapsulator passes the SNAP packet to SNAP Activator for service control purposes. The same process is repeated at every traversing node.

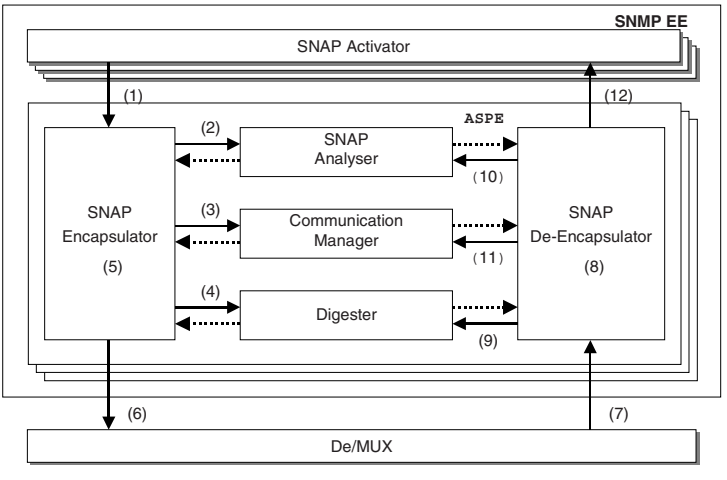

Fig. 7. Block diagram for ANEP-SNAP packet flow

4 Dynamic Diffserv QoS Configuration and Consideration

To test our architecture, we executed dynamic Diffserv QoS configuration. With regard to QoS management by active networking, the management has been reported[14]. However our main objective is not executing QoS management. The point is execution of application in the VE/EE.

In this test, a video flow was transmitted from Cambridge test-bed to Berlin test-bed. A user tries to execute Diffserv transmission. The user connects Cambridge and Berlin through hybrid active network nodes (Hardware router and Active Proxy (VE with Diffserv controller)). The hardware router is just a box for transmission of packet based on priorities. The Active Proxy provides an environment to instantiate a Diffserv controller. Then the user assigns DSCP-0 to a video flow from the Cambridge test-bed and DSCP-8 is assigned to a jam flow in Berlin test-bed in the beginning. In this case, the jam traffic has a high priority. Therefore if the jam traffic fully consumes available bandwidth for an output port of the hardware router, the video flow will suffer service degradation. To avoid this, SNAP active packet is injected from SNAP sender at Berlin to the video sender at Cambridge and the DSCP value of the video flow is changed from DSCP-0 to DSCP-56 at Cambridge. In this case, the video flow acquires higher priority than that of the jam flow. Therefore the video flow is transmitted unaffected by the jam flow.

To realize the function, the VEM creates a new VE and basic components such as the De/MUX, the SEC and the RCF components including the Diffserv Controller in the Java-EE. In addition, the ACTIVE SNMP-EE is also created. To configure the active nodes, the ACTIVE SNMP-EE injects a SNAP packet and the De/MUX intercepts it. Integrity of the ANEP-SNAP packet is guaranteed by the SEC function and the RCF component assigns QoS to a specific flow dynamically. Through this test, the FAIN network provides a secure and dynamic network configuration. In addition, users can create their Diffserv controller in their VE and configure them independently. Besides, we have realized communication between the Java-EE and the ACTIVE SNMP-EE. Usual active network framework is constructed based on one EE. However, our framework is constructed based on resources. The users can create multiple EEs in VE. Therefore it supports communication between EEs. In this sense, our architecture has high flexibility.

5 Conclusions and Future Work

In this paper, we have described the detailed components of the FAIN active node framework based on the novel Virtual Environment (VE) concept. The VE framework provides the natural coexistence of multiple EEs in each VE and it is supporting interactions between EEs. It has been validated by the communication between two types of Execution Environments: Java-EE and Active SNMP-EE. In addition, we have provided a secure control framework by the active packet. It has been guaranteed by the hop-by-hop and end-to-end authenticity and integrity checking of the active packet. Further more, we have controlled the Diffserv functions. In the near future, we will extend our framework to other resources and to other cases of autonomic deployment of services bringing just the right services to the customer at just the right context.

Acknowledgement. This paper describes work undertaken in the context of the FAIN – Information Society Technologies (IST) 10561 project. The IST program is partially funded by the Commission of the European Union.

References

1. FAIN Project WWW Server (FAIN Deliverable) - http://www.ist-fain.org
2. A. Galis, S. Denazis, C. Klein, C. Brou (eds.), book: "Programmable Networks and their Management", Artech House Books (www.artechhouse.com) , ISBN: 1-58053-745-6 (to be published 4th Quarter 2003)
3. S. Denazis, T. Suzuki, C. Kitahara, T. Becker, D. Gabrielcic, A. Lazanakis, W. Eaves, L. Cheng et al., "Final Active Node Architecture and Design", FAIN Deliverable 7, May 2003
4. P. Flury, E Boschi, T. Suzuki, C. Kitahara, D. Gabrielcic et al., "Evaluation Results and Recommendations", FAIN Deliverable 9, May 2003
5. S. Denazis, S. Karnouskos, T. Suzuki, S. Yoshizawa, "Component-based Execution Environments of Network Elements and a Protocol for their Configuration", IEEE - Transactions on Systems, Man and Cybernetics, Special Issue on Technologies that promote computational intelligence, openness and programmability in networks and Internet services, 2003 (to appear).
6. SNAP (Safe and Nimble Active Packets) - http://www.cis.upenn.edu/~dsl/SNAP/
7. D. Alexander, B. Braden, C. Gunter, A. Jackson, A. Keromytis, G. Minden, D Wetherall, "Active Network Encapsulation Protocol (ANEP)", Intrenet Draft - http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt
8. Netfilter - http://www.netfilter.org/
9. C. Brou, C. Kitahara, C. Tsarouchis, J. Vivero et al, "Final Specification of Case Study Systems", FAIN Deliverable 8, May 2003
10. J. Moore, M.Hicks, S. Nettles, S., "Practical Programmable Packets", Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society.
11. D. Harrington, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", RFC3411, December 2002, IETF.
12. W. Eaves, L. Cheng, A. Galis, "SNAP Based Resource Control for Active Networks", GLOBECOM 2002.
13. L. Cheng, W. Eaves, A. Galis, "Strong Authentication for Active Networks", accepted for presentation at and to appear in the proceedings of IEEE-Softcom 2003.
14. S. Vrontis, I. Sygkouna, M. Chantzawa, E. Sykas, "Enabling Distributed QoS Management utilizing Active Network Technology", 2003 IFIP-IEEE International Conference on Network Control and Engineering, 11-15.10, 2003, Muscat, Oman.