Management and Performance of Virtual and Execution Environments in FAIN

Thomas Becker[1], Lawrence Cheng[2], Spyros Denazis[3], Dusan Gabrijelcic[4], Alex Galis[2], George Karetsos[5], Antonis Lazanakis[5]

[1]Fraunhofer Institute for Open Communication Systems FOKUS, Germany
e-mail: becker@fokus.fhg.de
[2]University College London, United Kingdom
e-mail:{l.cheng, a.galis}@ee.ucl.ac.uk
[3]Hitachi Europe Ltd., Hitachi Sophia Antipolis, France
e-mail: spyros.denazis@hitachi-eu.com
[4]Jozef Stefan Institute, Laboratory for Open Systems and Networks, Slovenia
e-mail: dusan@e5.ijs.si
[5]National Technical University of Athens, Greece
e-mail: {laz, gkaret}@telecom.ntua.gr

Abstract. Next generation network nodes are required to function within heterogeneous network environments, where new services and protocols are rapidly deployed on demand. In such emerging environments, traditional node architectures that offer a predetermined and preloaded set of services, are increasingly incapable of coping with these new requirements. Accordingly, there is a need for new node architectures that offer higher degrees of flexibility measured by their capability to extend the functionality of the node and change its behaviour on demand. This paper makes use of programmable and active network technologies as developed during the FAIN project[1], to present a novel secure active node architecture, called the FAIN node architecture, capable of supporting virtual environments (VEs) for the allocation of the required amount of resources in which new services are dynamically deployed together with their entire execution environments (EEs). To this end, multiple VEs and services run simultaneously and interact securely with the node resources and mechanisms through open interfaces and the FAIN node management framework. We also present the implementation of the FAIN node architecture and two case studies that demonstrate its extensibility aspects and novel features.

[1] The FAIN project is partially funded by the Commission of the European Union as IST project 10561 (www.ist-fain.org).

1 Introduction

One of the biggest obstacles faced by the networking industry today is the difficulty traditional network nodes and management stations have, in coping with increasing degrees of heterogeneity. This heterogeneity is manifested in the form of different types of networks, i.e. access, edge or core, hardware and software technologies, and protocols. Accordingly, the goal for rapid and autonomous service creation, deployment, activation and management becomes even more elusive, with detrimental effects on the operator's and service providers' revenues and on their willingness to upgrade their infrastructures and support innovation. Equally important is the type of heterogeneity generated by the different communities of users and their varying requirements. Services designed and engineered to address the needs of these communities place different and, most notably, conflicting demands on the use of network resources, the degree of quality of service, the levels of security etc. Furthermore, these services must coexist with each other and evolve as their corresponding user communities evolve.

Dealing with these different types of heterogeneity is a complex problem. Its solution calls for services that are free to choose the software technology that is best suited for the needs thereof and deployed at strategic places in the network for higher performance and scalability gains. In contrast, networks must be capable of hosting such technologies, sharing and allowing the open control of their resources by a multitude of services. They must also provide the means of extending and updating their functionality on demand, thereby adapting their behaviour according to real time service/application requirements in present and future. Similar solutions are advocated by emerging hardware technologies such as network processors [1], research initiatives such as active and programmable networks [2][3][4][5] and standardisation efforts such as IETF ForCES [6] and IEEE P1520 [7][8].

In this paper, we attempt to provide an elaborate answer to the question about the characteristics and features of the next generation network architectures that support the aforementioned network capabilities. Central to such network architectures is the network node architecture the detailed description of which is used in order to demonstrate how such design goals have been met. Section 2 starts with a summary of the main concepts of the FAIN project [19]. In section 3, we present the FAIN node architecture with its major architectural components, namely, the VE management framework, the Resource Control Framework, the Demultiplexing/Multiplexing system, and the Security system. In section 4, we describe the implementation of different types of EEs on FAIN nodes. Next, we outline two case studies built as part of the FAIN project with initial performance measurements. In section 6, we contrast FAIN concepts and innovation with related work from the state of the art. Finally, our conclusions and future work are given in section 7.

2 FAIN Overview

The FAIN node architecture heavily draws on the FAIN reference architecture and the two main concepts of the FAIN project, namely, the Virtual Environment (VE) and the component based EE (a particular type of EE). A detailed description may be found in [9][10]. The FAIN reference architecture proposes a node that supports the partitioning of its resources assigned to VEs. VEs act as containers in which multiple

EEs may be deployed. In turn, EEs act as hosts to the services running in them, and consuming resources allocated to VEs. EEs also represent different implementations (technologies), implying that services must be implemented in the same technology as the EE, e.g. Java EE, in order to be deployed in the EE. Services may be distributed across different EEs, which, in turn, may be interconnected with each other. The interconnected EEs may reside in different operational planes, namely, control, management and transport. This approach has also been recently adopted by the ForCES model [11]. Finally, access to node resources and services is achieved through open interfaces [12], which act as an interoperability layer among the different implementations.

The FAIN node provides additional support for the component-based type of EE, being one of the most flexible programming environments for service deployment. Its existence implies that services must be designed and developed along the lines of the component-based approach in order to exploit the EE's capabilities. According to this approach, complex services are composed of simpler ones, which are then connected together to form specific structures representative of the service. In this way, services become extensible, their lifecycle management is simplified, and they can be readily introduced in the network. VEs are part of a virtual network owned by a provider. The provider uses the virtual network and consequently the VEs to deploy customer services and control the allocated resources according to its policies. As services may require the presence of specific EEs, these EEs must also be deployed. Each node must then make sure that the allocated resources are solely used by the owner of the virtual network and charged only to the appropriate services running in it. In other words, a VE provides a place where services together with their execution environments may be instantiated and used by a community of users or groups of applications while remaining isolated from others residing in different VEs. In the next section we describe the FAIN node architecture specified according to these design principles.

3 FAIN Node architecture

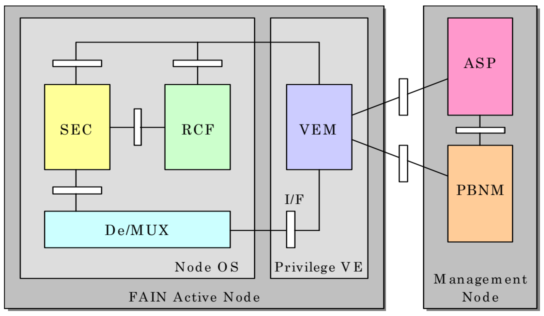

Fig. 1. FAIN active node architecture and its components

Fig. 1 depicts the FAIN node architecture with its major components, and its interaction with the management node that includes the Active Service Provisioning (ASP) and Policy Based Management System (PBNM) [13]. When the FAIN node boots up, a Privileged VE (pVE), with its VE manager (VEM) component, is automatically instantiated. The VE manager implements the VE management framework that offers access to a number of node services necessary to configure, setup and manage the node. Four major operations may be carried out: a) deployment and instantiation of VEs, b) deployment and instantiation of a number of EEs, c) deployment, instantiation and interconnection of service components with an EE, and d) control and management of a service component or a resource by means of open interfaces, leading to the interconnection of EEs residing in different operational planes (see Fig. 1). The VEM interacts with the Security component that offers a set of security services and enforces node policies, the Resource control component responsible for implementing the FAIN resource control framework (RCF) and the demultiplexing/multiplexing component which is configured to deliver packets to the right VE and EE when the latter has been deployed. In the subsequent sections, we describe each one of these components.

3.1 The FAIN VE Management Framework

The purpose of the VE management framework is to manage virtual environments and their allocated resources as well as the services installed therein. According to this framework, services and resources are both represented as components, which all offer a specific set of ports. Ports are used to inter-connect components and to make a component's particular functionality available to the outside. In the case of a resource component, it also represents a certain share of the available computation, storage, or communication capacity of the node. Components may be combined in various formations creating in this way more complex resources and services from simpler ones.

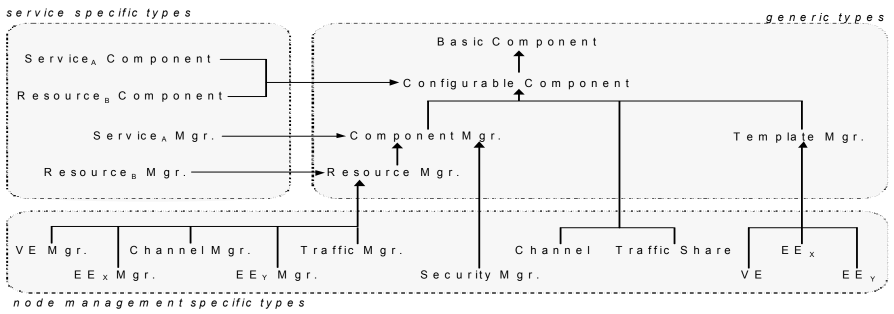

Fig. 2. Inheritance Hierarchy of Component Types

Fig. 2 depicts the inheritance hierarchy of the component model. At the root of the hierarchy of component types we find the *Basic Component*. It abstracts common functionality necessary for every type of component such as, discovery of the ports provided by the component, access control to ports based on policies defined in an individual security context, unique identification, and defined ownership. The *Configurable Component* adds to this the possibility to get and set the current internal configuration of the component in the form of a list of name-value pairs. It also offers operations to set up and manage connections between its own ports and the ports of other components. Using these operations of the Configurable Component, one can create services out of a set of available components by interconnecting and configuring them appropriately.

Components of a certain type need to be discovered, instantiated, linked with other components, and deleted when not in use. This is the job of the *Component Manager*. Since components may represent resources in the node we have extended the component manager with the *Resource Manager* that manages resources according to their allocated resource quotas. These resource instances represent a certain resource share for which the current usage can be monitored and callbacks can be registered for notifications when specific thresholds are reached.

As mentioned earlier, resources are allocated to specific VEs and services are deployed together with their entire EEs within VEs. Accordingly, the component model has been enhanced with a number of specific component types and managers in order to adhere to the FAIN design principles. They consist of management of VEs and EEs, security, traffic, and packet dispatching. More specifically, the *VE Manager* is acting as a factory and finder for VEs. It will allocate requested resource shares using other basic managers during the creation of a new VE and monitor the overall resource usage during its lifetime. Various technology specific *EE Managers* take care of the management of EEs providing the runtime environments for component instances, e.g. a Java virtual machine. The *Channel Manager* is used to create channels which forward packets from the network to connected component instances based on a set of rules and also send packets coming from component instances to the network. The *Traffic Manager* partitions the node's available bandwidth into shares and allows to assign packet flows to them. The *Security Manager* is used to set up and manage the individual security contexts which are assigned to each component instance during its creation and used to control the access to the instance's ports.

After the creation of the requested resource shares the VE Manager will attach them to the new VE so they are available to service components which will be installed and instantiated inside the VE later. In order to install components we introduced the notion of a *Template Manager*. A template is identified by a name and a version number and includes a particular implementation of a component type together with its corresponding component manager type. Both VEs and EEs are Template Managers and thus support the installation and management of component types. While an EE provides a concrete technology dependent runtime environment for component instances the VE abstracts from this. During the installation of a template the VE will try to find an appropriate EE in the list of attached EEs and forward the installation request to the EE. The EE is then responsible to carry out the technology dependent steps to make the new component type available.

The generic component framework serves as a starting point for new services or resource abstractions (see upper left part of Fig. 2). A developer would implement specific component types by deriving from the basic or configurable component types and simply add the implementation for the service specific ports, if any. For service components the developer would derive a manager type from the generic component manager type while for resource components the resource manager type would be used.

3.2 The FAIN Resource Control Framework (RCF), Demultiplexor (De/MUX), and Security

In the design of the FAIN node, major attention was given to the management of the node resources and their sharing among the various users of the node. The part of the node responsible for this task is called Resource Control Framework (RCF). An overview of the RCF is given here. Detail descriptions of the RCF can be found at [30][32]. The RCF is considered very important as it supports one of the major design goals of the FAIN project: the partitioning of resources (capacities) among VEs and their residing services. Through resource partitioning, supported by the RCF, the various VEs are kept isolated from each other and allowed to consume only the allocated amount of resources. In a similar manner, VEs are enabled to further allocate, manage and control their own (virtual) resources according to the VE's own policies and logic, customised according to the type of services that are running in them.

As a FAIN node supports multiple VEs with multiple EEs running in them, arriving packets must be delivered in a secure way to the right entity inside the node. The functional entity responsible for delivering packets is the Demultiplexing/Multiplexing (De/MUX) component. An overview of the De/MUX architecture is presented here. For details readers should refer to [30][31]. The design of the De/MUX is also influenced by and built according to the component model principles, thereby making use of the VE management framework and its abstractions. The main components of the De/MUX are the Channel Manager and the Channel components. The Channel Manager is responsible for creating and deleting the Channel object. The Channel object is created for each VE. The specific types of Channels correspond to different types of flow packets arriving at the node: namely active and non-active (data) packets but these may be extended to new types of packets by introducing new Channels and deploying them through the VE management framework mechanisms. Furthermore, the De/MUX also interfaces with the Security architecture, which provides a safety framework to protect against unauthorised active packet processing. When the Channel Manager receives the active packets, it calls a security interface to execute security check, the result of which determines whether the Channel Manager forwards or discards the active packets. For outgoing active packets, the Channel Manager calls the security interface to insert security information into them, used for executing the security checks at the next active node.

Active networking raises a number of different security issues, and FAIN designed and implemented a complete security architecture to address these issues. As this security architecture is thoroughly explained by another paper [27] we only discuss here the functionality and aspects relevant to the VE management framework. To this end, we distinguish between two different categories of facilities the security functionality offers: a) *Communication security facilities* are set of services, protocols and mechanisms that provide at system level data origin authentication and integrity services for packets exchanged in between nodes. Combination of per hop protection, which uses symmetric cryptography, and protection regarding the originator, which uses asymmetric cryptography, were used to protect active packets data tackling the issue that parts of the packets (i.e. the packets' payload) may be changed in the network, whilst certain parts of the packets (i.e. static executable codes) should remain unmodified; *b) System security facilities* are set of services and mechanisms that provide authorization and policy enforcement on a node, system integrity, code verification and accountability of system operation.

4 EE Implementation

The implementations of EEs may differ in the employed technology and the way deployed service components are executed. Two different types of EE were implemented in FAIN: a) the Java EE, and b) the Active SNMP EE. The Java EE type is used for the installation and execution of components implemented in Java. It also serves as the EE for the node's management layer described in section 3.1. For the management of templates (i.e. component types) a Java class loader is employed. At the operating system level, the Java EE is mapped as a separate process executing a Java virtual machine (JVM) charged to the VE it belongs to. The CPU and memory usage of all Java EE processes are monitored by the Java EE manager and will be temporarily suspended when the previously agreed resources are overused for a certain amount of time. Although the underlying component model allows connections between component ports using arbitrary protocols, the Java EE implements component ports as CORBA interfaces. Consequently, the communication inside the management layer is based on CORBA. The enforcement of access control policies for those ports is realised with CORBA portable object adaptors and interceptors interacting closely with the respective component's security context. The implementation of access control enforcement also supports delegation so that a chain of calls via a series of interfaces can be checked against policies. The Active SNMP EE [14][17][29] extends the functionality of Safe and Nimble Active Packet (SNAP) active packet protocol [15] through exploitation of the SNMP protocol. SNAP packets essential contain a series of byte code instructions. It is claimed to be light-weight, efficient, safe and practical. The discussion of SNAP is beyond the scope of this paper. For details of SNAP, readers should refer to [15]. It is a management EE we have built in FAIN and is used in the Diffserv scenario presented in the next section and in [30]. Other examples of management EE are for instance the Smart Environment for Network Control, Monitoring and Management (SENCOMM) [33]. The FAIN management EE consists of two components: a) the *SNAP Activator*: consisting of the SNAP Sender/Receiver and the snapd (snap daemon), and b) the *ANEP-SNAP Packet Engine* (ASPE). The SNAP Sender is responsible for generating SNAP packets and injecting them into the network. The snapd is extended to include SNMP functionalities and is responsible for the execution of SNAP active packets. The ASPE is responsible for providing ANEP encapsulation/de-encapsulation for SNAP packets, and co-ordinates Active SNMP EE operations with De/MUX and SEC for transmitting active packets and for securing active packets during-transit across the network, respectively [14][17].

5 Case Studies

The Diffserv scenario [16][30] demonstrates the deployment of *heterogeneous technologies* (EEs) on *different planes* (i.e. control / management planes) for enhancing flexibility and extensibility in the FAIN architecture; this is demonstrated through the co-existence of *multiple heterogeneous* EEs within a VE, and subsequently a generic approach for *inter-EE communications*. In addition, the *dynamic deployment of service components* (i.e. a DiffservController) within an EE to support new services through *interactions* between various FAIN components was also demonstrated. The full Diffserv scenario was presented in [30]. The WebTV scenario was proposed in order to demonstrate the deployment of a service component together with its corresponding EE. According to it, a WebTV Service Provider requests the creation of a VE in order to use the node resources for transmitting video to his customers. A new customer that wants to get this service is not capable of receiving the stream in the transmitting format. For this reason the Service Provider (SP) dynamically downloads to a nearby node a service component in the form of a transcoder that is compatible with the customer's system. This component adheres to the definition of the component model which makes it possible to seamlessly deploy it using the VEM. The service component is instantiated in a Java EE and the flow is rerouted to be processed by the new component [13] [16], which sends the TV stream in the correct format. In contrast with the Diffserv scenario, the WebTV scenario deploys an EE in the transport plane wherein packet receive additional processing through the newly deployed service component. This component may be further configured e.g. changing the value of the transcoder to a new one, by another EE that resides in the control or management plane.

5.1 Performance Trials

Performance trails for the Diffserv scenario were carried out with the objective of evaluating the viability of the features of the FAIN active node; namely the *VEM bootup time* (i.e. node instantiation time and manager installation time), and *service deployment time*. A PC with an Intel 746MHz Pentium III CPU and 512MB memory was used to measure the VEM bootup time. Note that the bootup sequence of VEM on a FAIN active node involves two steps: node initialisation and managers installation. The average total time for node initialisation is 313.8 ms. This is the average time to initialise an ORB (4.6 ms), a pEE (94.1 ms) and a pVE (165.2 ms). The average manager installation time for installing the VE manager is 389.6 ms, 72 ms for the Java EE manager, 2146.5 ms for the channel manager, 76.7 ms for SEC manager, 75.9 ms for traffic manager, 23.1 ms for Diffserv manager, 28 ms for Active SNMP EE manager. An addition 6.1 ms for finishing the boot sequence. Thus the average total time for booting up VEM on a FAIN active node is 3031.9 ms. Note that different times for installing different managers mainly depend on the individual size and numbers of classes to be loaded. The channel manager also has to load an external library for connecting to Linux netfilter. For service deployment performance measurement, a PC with an Intel 995MHz Pentium III CPU and 112MB RAM was used. The average active packet processing time in the Active SNMP EE is 19.3 ms.

Note that in a real-life scenario, the VEM is expected to be brought up once only when a FAIN active node boots up. Consequently, a service EE deployed in a VE is brought up only when there is a new service request. Thus neither node instantiation nor manager installation is frequent. Moreover, VEM will operate for a long period of time commensurate with the lifetime of the SP virtual network. Although service (i.e.

EE) deployment is comparatively more frequent, it should be noted that dynamic service creation on networks through service EE deployment is much more efficient than manually configuring each node to enable new services. Given that the enhanced level of flexibility of the FAIN active node architecture, the results of these feasibility trials are encouraging, and are proving the feasibility of deploying FAIN active nodes in a real network.

Another test was run in order to get an assessment of the scalability of the Java based EE and thus the implementation of the FAIN active node's management layer. For this test a PC with an Intel 746MHz Pentium III CPU and 512MB was used. After the boot-up of the privileged VE eight separate VEs with attached Java EEs were created (average of 8 seconds per VE/Java EE). Using a local test client a Java based test service was installed in each of the eight VEs (average of 366 milliseconds per installation) and for each VE 1000 instances were created and configured. As an example for user-space packet processing the test service connects itself to the respective VE's packet dispatching channel during configuration which requires also interaction with the privileged VE where the dispatching channels are managed. For the test security was enabled, i.e. the communication between the test client, the privileged VE and the particular VE was secured and access to the involved components was controlled by the security manager. Fig. 3 shows the times in milliseconds for the creation and configuration of instances versus the number of instances for each VE. For the first few instances the minimal time is decreasing due to already loaded classes in the Java EEs. After that the average time is increasing constantly with the number of instances already created. The sporadic peaks are most probably due to garbage collection inside the Java EEs. The average time for service creation is still below 500 milliseconds for 1000 service instances per VE. Considering the non-trivial service configuration which requires interaction with the packet dispatcher inside the privileged VE plus the control of all interactions by the security manager we argue that this figure is acceptable and the Java based implementation of the FAIN active node's management layer scales well.

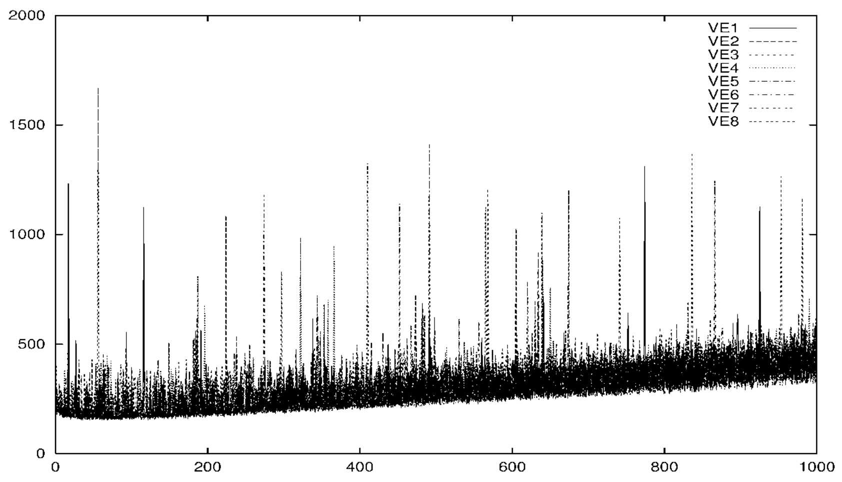

Fig. 3. Service creation in 8 VEs with attached Java EEs

6 Related Work

The definition of EEs was defined by DARPA. EEs can be treated as the runtime environment of a process or a process itself [20], or toolkits for building active applications (services) [18], or a programming environment characterised by the implementation language, like Java [4] [21]. EEs have also been proposed as extensions of the Node OS [22] whereas in [23] [24] EEs are characterised not by the choice of technologies but rather by the services they offer and the architectural plane they operate at, namely, control, management, and transport. In other cases, EEs are treated as VEs acting as the principal abstraction for authentication, authorisation and resource control [21] [22]. As a result the reference architecture proposed in [21] makes it very difficult to support inter-EE communication either within the same node or in different ones. All this ambiguity has an impact on defining a systematic approach around a consistent model that facilitates the dynamic deployment of new services taking into consideration the de-facto heterogeneity found in the network in the form of hardware and software technologies, protocols etc

In contrast, FAIN attempted to deal with the problem of service deployment in heterogeneous networks by defining a reference architecture [9] based on a clear separation between VEs and EEs. Another approach to isolating active services from each other was described in [34], [35], [36].To this end, we have used the same approach in defining and using VEs as in [12], while EEs are distinguished between *EE type* (the programming environment and programming methodology) and *EE instance* (the specific implementation of the EE type) [9]. The benefit of using these two concepts as distinct entities enables us to build virtual networks that are service specific (e.g. service overlays). The VE abstraction allows for the allocation, control and charging of resources whereas EEs allows us to deploy services in nodes together with the whole implementation environment. We have demonstrated this in the Diffserv scenario where we deployed a Java control EE and a management EE using SNMP and SNAP (the latter was developed by the Switchware team). In the same way, using the VEM framework other EEs developed by third parties, e.g. ANTS may be deployed and combined with EEs already present in the node. Note that during the ANEP encapsulation process at the ASPE, static contents of a SNAP packet program (e.g. the byte codes) are determined by the ASPE and are encapsulated into ANEP

Payload. The entire SNAP packet program (which includes both the static and the dynamic contents of SNAP) is kept in one of the ANEP Option fields. The static contents of SNAP packet can be easily determined since SNAP clearly defines its static contents to be its byte codes stored on its stack. The static contents of SNAP packet that are kept in ANEP Payload are protected by a signature related to the principal. The signing process is performed at SEC. Note that ASPE is *independent* of SEC. ASPE is in fact developed as part of the SNAP package. Integrating ASPE into the SNAP package is currently undergoing. It is the responsible for the ASPE to recognised the packet format of SNAP and to determine which parts of the SNAP packet are static. Whereas the ANEP Option that keeps the SNAP packet program (which includes both the static and dynamic contents of SNAP) is protected by hop protection again at SEC. Symmetric cryptography techniques are used for hop-protection, a solution that is similar to deploying the Authentication Header in IPSec [16] [17] in a per-hop fashion. The advantage of keeping the *entire* SNAP packet in an ANEP Option is that no packet marshalling at the Active SNMP EE is needed: the entire SNAP packet is put into the ANEP Option. This is in contrast with existing authentication methods such as SANTS (Secure ANTS) [28] where static and dynamic contents of an active packet are actually *split* and encapsulated into corresponding ANEP fields. When encapsulating ANTS packets in ANEP in the SANTS approach, it was said that "The variable area of our (ANEP) packet includes the variable fields of the EE header (of ANTS) and the variable portions of the data payload (of ANTS)" [28]. This implies certain (variable) parts of ANTS packets must be extracted and placed in the variable area of ANEP packet. This incurs overhead on packet marshalling at each node. By avoiding packet splitting the overall efficiency of the security processing on FAIN architecture is improved. Also, unlike the SANTS approach in which a modified ANEP format is used to keep active packets, the standard ANEP format is used in FAIN to avoid interoperability problems. Full explanation on the functionalities of the ASPE can be found in [17] [29]. The process of integrating the ASPE functionalities into the SNAP package is currently undergoing.

The component-based approach heavily draws on the Netscript and the IEEE P1520. We have generalised it to treat also node resources as components that can dynamically be deployed upon request and accessed via open control interfaces. These control interfaces may be instantiated on demand in any EE irrespective of its implementation. In this way services are enabled to control their allocated resources from their specific implementation environment.

In FAIN we have also built a high-performance EE [25] residing in the Linux kernel-space in order to deploy service components with time restrictions. This EE is deployed using the same toolset offered by VEM and controlled by control EEs implemented in Java residing in the user-space. Furthermore, we have used the reference model of the architecture to automate the process of service provisioning in the network. This is achieved through the network Active Service Provisioning system [26] which uses VEM to enforce the deployment decisions taken at the network level.

7 Conclusions

In this paper we have presented the FAIN Active Node architecture, capable of combining and coordinating different Execution Environments that represent different technologies which are then used to host service components and interact with each other as part of the overall service operation. To this end, the FAIN node may change or extend its functionality and seamlessly operate in a heterogeneous network. This has been achieved through the definition of the VE management framework that combines EEs, VEs, and service components.

The VE management framework is realised through a number of classes with methods that allow EEs to be deployed in VEs, in turn, service components to be deployed and linked with existing services in EEs, and exports control interfaces of these components for their configuration. The operation of the EEs, and of the services running in them, is regulated by the FAIN resource control framework. This framework is based on the VE abstraction, which is used as a principal for accounting and resource allocation and partition. Moreover, all the operations take place in a secure environment founded on the flexible FAIN security architecture, which provides authentication, authorisation of the use of resources and verification of packets. The FAIN node architecture and its components have been implemented, and a number of different EEs residing in different operational planes and interworking with each other have been created. Their functionality and mechanisms achieve the design goals. The JavaEE, residing in the management plane, binds together all the FAIN node components as well as the other EEs. Finally, the flexibility of the FAIN node and its new features has been demonstrated through two case studies, namely, Diffserv and WebTV deployment.

Acknowledgments

This paper describes work undertaken and in progress in the context of the FAIN – IST 10561, a three year project (2000-2003), that is partially funded by the Commission of the European Union. The authors would like to acknowledge other members of the FAIN consortium.

References

[1] IEEE Network Magazine, Special Issue on Network Processors, Harrick Vin and Raj Yavatkar (eds), Vol. 17, No 4, July/August 2003.

[2] Open Signalling Working Group, http://www.comet.columbia.edu/opensig/.

[3] Campbell, A. T., H. De Meer, M. E. Kounavis, K. Miki, J. Vicente, and D. Villela "A Survey of Programmable Networks", ACM Computer Communications Review, April 1999

[4] Johnathan M. Smith, Kenneth Calvert, Sandy Murphy, Hilarie K. Orman, and Larry L. Peterson, "Activating Networks: A Progress Report", IEEE Computer 32(4):32–41, April 1999.

[5] Jonathan M. Smith, Scott M. Nettles, "Active Networking: One View of the Past, Present and Future", Special Issue of IEEE T-SMC on technologies promoting computational intelligence, openness and programmability in networks and Internet services, Autumn 2003 (in press)

[6] IETF ForCES, http://www.ietf.org/html.charters/forces-charter.html

[7] Biswas, J., et al., "The IEEE P1520 Standards Initiative for Programmable Network Interfaces", IEEE Communications, Special Issue on Programmable Networks, Vol. 36, No 10, October, 1998

[8] Vicente, J., S. Denazis, et al., "L-interface Building Block APIs", IEEE P1520.3, P1520.3TSIP016, 2001.

[9] 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, February 2004 (in press)

[10] "Overview FAIN Programmable Network and Management Architecture", FAIN Project Deliverable 14

[11] Yang, L., J. Halpern, R. Gopal, R. Dantu, "ForCES Forwarding Element Functional Model", March 2003.

[12] J.E. van der Merwe, S. Rooney, I.M. Leslie and S.A. Crosby, "The Tempest - A Practical Framework for Network Programmability", IEEE Network, Vol 12, Number 3, May/June 1998, pp.20-28.

[13] Christos Tsarouchis, Chiho Kitahara, Spyros Denazis, et. al., "A Policy-Based Management Architecture for Active and Programmable Networks", Special Issue on Network Management of Multi-service, Multimedia, IP-based Networks, IEEE Network Magazine, May/June 2003.

[14] Eaves, W., Cheng, L., Galis, A., “SNAP Based Resource Control for Active Networks”, IEEEGLOBECOM 2002.

[15] Moore, J., Hicks, M., Nettles, S., “Practical Programmable Packets”, Proceedings IEEE INFOCOM 2001.

[16] FAIN Project Deliverable D40 - FAIN Demonstrators and Scenarios, http://www.ist-fain.org/deliverables

[17] Cheng, L., Eaves, W., Galis, A., “Strong Authentication for Active Networks”, IEEE-Softcom 2003.

[18] Wetherall, D.J., “ANTS: a toolkit for building and dynamically deploying network protocols”, OpenArch 1998, San Francisco, CA, April 1998, pp.117-129, IEEE.

[19] A. Galis, S. Denazis, C. Brou, C. Klein, (ed) – “ Programmable Networks and Programmable Network Management “ ISBN 1-58053-745-6; contracted for publishing in Q4 2003 by Artech House Books, 46 Gillingham Street, London SW1V 1AH, UK; www.artechhouse.com

[20] Steven Berson, Bob Braden, and Livio Ricciulli, “Introduction to The ABone”, June 15, 2000.

[21] “Architectural Framework for Active Networks”, Draft version 1.0, K.L. Calvert, ed., July 27, 1999. http://protocols.netlab.uky.edu/~calvert/arch-latest.ps

[22] “Node OS Interface Specification”, AN Node OS Working Group, Larry Peterson, ed., November 30, 2001.

[23] Bhattacharjee, S., “Active networks: Architectures, Composition, and Applications”, Ph.D. Thesis, Georgia Tech, July 1999

[24] B. Braden, A. Cerpa, T. Faber, B. Lindell, G. Phillips, J. Kann and V. Shenoy, “Introduction to the ASP Execution Environment (Release 1.5)”, Nov 30, 2001.

[25] R. Keller, L. Ruf, A. Guindehi, B. Plattner. PromethOS: A Dynamically Extensible Router Architecture Supporting Explicit Routing, Proceedings of the Fourth Annual International Working Conference on Active Networks IWAN, Zurich, Switzerland, December 2002. Lecture Notes in Computer Science 2546, Springer Verlag.

[26] Y. Nikolakis, E. Magana, M. Solarski, A. Tan, E. Salamanca, J. Serrat, “A Policy-Based Management Architecture for Flexible Service Deployment in Active Networks”, IWAN 2003, Kyoto, Japan 2003

[27] D. Gabrijelčič, B. J. Blažič, J. Tasič, Future Active IP Networks Security architecture, Computer Communications Special Issue on Activated Internet, 2004, in revision.

[28] S. Murphy, “Strong Security for Active Network”, IEEE OpenArch 2001.

[29] L. Cheng, “Active Networks Authentication”, LCS 2003.

[30] T. Suzuki, S. Denazis, L. Cheng, T. Becker, D. Gabrijelcic, A. Lazanakis, “Dynamic Deployment & Configuration of Differentiated Services Using Active Networks”, IWAN 2003.

[31] Chapter 4, FAIN Project Deliverable D7 – Final Active Node Architecture and Design, http://www.ist-fain.org/deliverables/del7/d7.pdf

[32] Chapter 3, FAIN Project Deliverable D7 – Final Active Node Architecture and Design, http://www.ist-fain.org/deliverables/del7/d7.pdf

[33] A. Jackson, “Active Monitoring and Control: The SENCOMM Architecture and Implementation’, Proceedings of the DARPA Active Networks Conference and Exposition (DANCE), 2002 pp.379-393.

[34] M. Brunner, “Service Management in Multiparty Active Networks”, IEEE Communications Magazine, 2000, p.144-151

[35] M. Brunner, B. Plattner, R. Stadler, "Service Creation and Management in Active Telecom Environments" Communications of the ACM, March 2001

[36] A. Galis, D. Griffin, W. Eaves, et. al - “Mobile Software in Active Virtual Pipes: Support for Virtual Enterprises“ pp. 427-450 in “On The Way To Information Society,“ ed. Magedanz et. al., IOS Press, Amsterdam, The Netherlands, ISBN 1 58603 007 8, April 2000.