Project Number :	IST-1999-10561-FAIN
Project Title :	Future Active IP Networks

D9

CEC Deliverable Nr :	D9
Deliverable Type :	PU
Dissemination:	Int
Deliverable Nature :	
Contractual date :	31st January 2001
Actual date :	6 May 2003

Editor :	Placi Flury, ETH
Workpackage(s) :	WP5
Abstract :	
Keyword List :	Scenarios, Testbed, Evaluation

Project Number :	IST-1999-10561-FAIN
Project Title :	Future Active IP Networks

D9

Editor :	Placi Flury
Document No:	D9
File Name	D9.doc
Contributors :	
Version :	0.9
Date :	Wednesday, 14 May 2003
Distribution :	WP3, WP4, WP5

The FAIN Consortium consists of:

Partner	Status	Country
UCL	Partner	United Kingdom
JSIS	Associate Partner to UCL	Slovenia
NTUA	Associate Partner to UCL	Greece
UPC	Associate Partner to UCL	Spain
DT	Partner	Germany
FT	Partner	France
KPN	Partner	Netherlands
HEL	Partner	United Kingdom
HIT	Partner	Japan
SAG	Partner	Germany
ETH	Partner	Switzerland
GMD	Partner	Germany
IKV	Associate Partner to GMD	Germany
INT	Associate Partner to GMD	Spain
UPEN	Partner	USA

The FAIN Consortium

University College London	(UCL)
Josef Stefan Institute	(JSIS)
National Technical University of Athens	(NTUA)
Universidad Politecnica De Catalunya	(UPC)
T-Nova Deutsche Telekom Berkom GmbH	(DT)
France Télécom / R&D	(FT)
Koninklijke KPN NV, KPN Research	(KPN)
Hitachi Europe Ltd.	(HEL)
Hitachi Ltd.	(HIT)
Siemens AG	(SAG)
Eidgenössische Technische Hochschule Zürich	(ETH)
GMD Forschungszentrum Informationstechnik GmbH	(GMD)
IKV++ GmbH Informations- und Kommunikationstechnologie	(IKV)
Integracion Y Sistemas De Medida, SA	(INT)
University of Pennsylvania	(UPEN)

Project Management

Alex Galis
University College London
Department of Electronic and Electrical Engineering,

Torrington Place

London WC1E 7JE

United Kingdom

Tel +44 (0) 207 458 4463

Fax +44 (0) 207 388 9325

E-mail: a.galis@ee.ucl.ac.uk

Authors

Arso Savanovic (JSIS)

Chiho Kitahara (HIT)

Dusan Gabrijelcic (JSIS)

Epi Salamanca (UPC)

Lukas Ruf (ETH)

Matthias Bossardt (ETH)

Richard Lewis (UCL)

Thomas Becker (FhG)

Walter Eaves (UCL)

Bernhard Plattner (ETH)

Drissa Houatra (FT)

Elisa Boschi (FhG)

Juan Luis Manas (INT)

Marcin Solarski (FhG)

Placi Flury (ETH) - Editor

Spyros Denazis (HEL)

Toshiaki Suzuki (HEL)

Yannick Carlinet (FT)

Table of Contents

1 INTRODUCTION 1

2 FAIN FUNCTIONAL CONCEPTS 2

3 THE FAIN DISTRIBUTED TESTBED 3

3.1 ACTIVE NETWORK NODES (ANN) 3
3.1.1 AN Node Type A 4
3.1.2 AN Node Type C 4
3.1.3 FAIN Network and Element Management Stations 4
3.2 NETWORK TOPOLOGY AND INTERCONNECTION 4
3.2.1 Tunnel Configuration 4
3.2.2 Partner Network Data /Properties 5
3.2.3 Domain Name Service 5
3.2.4 Sites Overview 6
3.2.5 Monitoring Tool 6

4 SCENARIOS 7

4.1 SCENARIO DEFINITION FRAMEWORK 7
4.1.1 Scenario Building Block 7
4.1.2 Generic Application Scenario 8
4.1.3 Application Scenario 8
4.2 SCENARIOS BUILDING BLOCKS 8
4.3 GENERIC APPLICATION SCENARIOS 12
4.3.1 DiffServ Scenario 12
4.3.2 WebTV Scenario 14
4.3.3 Web Service Distribution Scenario 14
4.3.4 Video on Demand Scenario 16
4.3.5 Mobile FAIN Demonstrator 17
4.3.6 Managed Access Scenario 18
4.3.7 Security Scenario 19
4.4 APPLICATION SCENARIOS 20
4.4.1 DiffServ Scenario 20
4.4.2 Security Scenario 24
4.4.3 WebTV Scenario 25
4.4.4 Web Service Distribution 27
4.4.5 Video on Demand Scenario 33
4.4.6 FAIN Mobility Demonstrator 34

5 EVALUATION OF THE ARCHITECTURE AND IMPLEMENTATION 37

5.1 EVALUATION METHODOLOGY 37
5.1.1 Templates and Representation 38
5.1.2 Classification of the FAIN Components 39
5.2 EVALUATION RESULTS 40
5.2.1 Flexibility 40
5.2.2 Security 43
5.2.3 Interoperability 47
5.2.4 Openness 50
5.2.5 Portability 51
5.2.6 Performance 54

6 CONCLUSIONS 63

7 ACRONYMS 64

8 REFERENCES 66

Table of Figures

FIGURE 3-1: TESTBED TOPOLOGY 5
FIGURE 3-2: NODES OVERVIEW 6
FIGURE 4-1: SCENARIO DEFINITION FRAMEWORK MODEL. 7
FIGURE 4-2: SEQUENTIALLY CONNECTED SBBS. 8
FIGURE 4-3: ACTIVE NETWORK NODE CONFIGURATION BY ACTIVE PACKET 13
FIGURE 4-4: DIFFSERV DEMONSTRATION SCENARIO. 13
FIGURE 4-5: SERVICE NODES AND REDIRECT SERVERS IMPLEMENT ACTIVE WEB SERVICES 15
FIGURE 4-6: GENERAL INFRASTRUCTURE OF THE DEMOS 18
FIGURE 4-7: DIFFSERV DEMONSTRATION SCENARIO 21
FIGURE 4-8: DIFFSERV NETWORK CONFIGURATION 22
FIGURE 4-9: GUI OF POLICY EDITOR 26
FIGURE 4-10: WEB TV MAPPED TO TESTBED 26
FIGURE 4-11: NETWORK TOPOLOGY USED BY THE DEMOS 28
FIGURE 4-12: NETWORK SET-UP FOR DEMONSTRATION AT FHG 28
FIGURE 4-13: STRUCTURE OF DEMO2 29
FIGURE 4-14: STRUCTURE OF DEMO3 30
FIGURE 4-15: STRUCTURE OF DEMO4 31
FIGURE 4-16: STRUCTURE OF DEMO6 32
FIGURE 4-17: NETWORK TOPOLOGY FOR VIDEO ON DEMAND SCENARIO. 34
FIGURE 4-18: NETWORK TOPOLOGY USED BY THE DEMOS 35
FIGURE 4-19: STRUCTURE OF DEMO2 36
FIGURE 5-1: EVALUATION MODEL FOR FEATURES AND PROPERTIES. 37
FIGURE 5-2: REFERENCE MODEL FOR OPERATIONAL PLANES AND LEVEL/LOCATION LAYERS 38
FIGURE 5-3: TWO LEVEL EVALUATION TEMPLATE FOR PROPERTY TYPES. 39
FIGURE 5-4: DEMUX TEST SYSTEM 54
FIGURE 5-5: SYSTEM DIAGRAM FOR DEMULTIPLEXING EVALUATION 55
FIGURE 5-6: BLOCK DIAGRAM OF THE DEFAULT DATA TRANSMISSION 56
FIGURE 5-7: TOPOLOGY FOR EVALUATION MEASUREMENT 58
FIGURE 5-8: NETWORK FOR PERFORMANCE MEASUREMENTS 60
FIGURE 5-9: EVALUATED COMPONENTS 61

Table of Tables

TABLE 5-1: CLASSIFICATION OF THE FAIN COMPONENTS 39
TABLE 5-2: TABLE FOR FLEXIBILITY PROPERTY TYPE 40
TABLE 5-3: TABLE FOR SECURITY PROPERTY TYPE 44
TABLE 5-4: TABLE FOR INTEROPERABILITY PROPERTY TYPE 47
TABLE 5-5: TABLE FOR OPENNESS PROPERTY TYPE 50
TABLE 5-6: TABLE FOR PORTABILITY PROPERTY TYPE 52
TABLE 5-7: SPECIFICATION OF THE PACKET SENDER 54
TABLE 5-8: SPECIFICATION OF THE ACTIVE NODE 55
TABLE 5-9: SPECIFICATION OF THE SENDER NODE 56
TABLE 5-10: SPECIFICATION OF THE FLOW THAT IS 5KBYTE LONG DATA 56
TABLE 5-11: SPECIFICATION OF THE FLOW THAT IS 1KBYTE LONG DATA 56
TABLE 5-12: SPECIFICATION OF THE FLOW THAT IS 2.5KBYTE LONG DATA 57
TABLE 5-13: BOOTSTRAPPING MEASUREMENTS 58
TABLE 5-14: NMS MEASUREMENTS 59
TABLE 5-15: EMS-SANTANA MEASUREMENTS 59
TABLE 5-16: EMS-KUBRICK MEASUREMENTS 59
TABLE 5-17: PROMETHOS MEASUREMENTS 61

1 INTRODUCTION

This document discusses and evaluates the results obtained in the FAIN project. It is to be read in conjunction with the other deliverables connected with milestone 6 (D7 and D8), and the document detailing the overall concepts of FAIN (Common part of D7, D8, and D9: Active & Programmable Network Management and Services Background).

The methodology used to evaluate the results of the FAIN project rests on three pillars:

1. The project built a testbed, in which the FAIN software results can be tested by any FAIN partner, live and in real-time. More importantly, still, the FAIN testbed is a distributed infrastructure, allowing for executing services that have been created and deployed on-demand by users or specific applications.

   The FAIN testbed was mainly used to test whether the functional requirements posed to the FAIN results was achieved, and whether the software components developed by the different partners would interoperate.

2. The project defined and developed a number of scenarios, which eventually were used to test the capability of the FAIN software to support close-to-real-life dynamically deployed services. The scenarios are split into *generic application scenarios*, which are used to explain how specific service building blocks are being used to realize a service. Generic application scenarios are abstract in the sense that they should not be seen in the context of a specific network topology; thus they are not executable, as the mapping onto a physical realization is not defined.

   Application scenarios, however, as defined in section in section 4.4, are executable, i.e. they refer to a specific context and specific network topology. Thus, application scenarios represent a description of what a spectator attending a demonstration of such a scenario would see.

3. The project developed a novel, table-oriented evaluation methodology which serves to assess the degree by which the FAIN results fulfill the requirements that have been identified in earlier deliverables (such as D2). The properties deemed important are: Flexibility, security, interoperability, openness, portability and performance.

The deliverable starts with identifying some key functional concepts that will further be detailed and referred to, in chapter 2. The chapters 3, 4 and 5 discuss the three pillars of the FAIN evaluation, as previously described. The document closes with a list of acronyms and a references section, which provides the links to other documents of importance.

2 FAIN FUNCTIONAL CONCEPTS

Looking back at the original project objectives as described in the Technical Annex (TA), we find that although they are still valid as general objectives, we can clarify and refine them as a result of experience gained during the project. We can also map the main objective onto a number of few more manageable objectives. We therefore rephrase our overall objective as follows:

To develop Active Network architecture oriented towards service deployment and execution in heterogeneous networks.

From this overall objective the following sub-objectives result:

1. Design and implement an AN node that is dynamically extensible and simultaneously supports different types of technologies and communities.
2. Design and implement a platform independent approach to service description and deployment.
3. Achieve Network Interoperability for service execution.
4. Increase the pace of standardization.
5. Design and implement a Policy-based Network Management Architecture suitable for the global management of active networks: it should be not only capable of delegating management functionality but also management responsibility to multiple authorities.

The FAIN project has originated a number of innovative concepts in order to achieve these objectives. It is these concepts that we need to identify here and explain in what sense they meet our objectives.

Creating Virtual Environments as part of Virtual Networks Creation

A series of Virtual Environments (VEs) has been established across an Active Network as part of the Virtual Network topology proposed during the Service Level Agreement (SLA) negotiation. The VEs also include admission control of the virtual network: resources are reserved and/or released and a number of node interfaces are instantiated and exported that allow VE clients to access and control their own partition; a common format for resource profiles and policies that are used for enforcement and configuration of the node, are also included in the VE.

Here, by creating VEs as part of the same Virtual Network, we provide different communities with their own resource space, from a single physical infrastructure, by which to deploy and use services in their own way. In this way objective 1 is partly achieved.

Resource Control for hard Resource Partitioning

Policing, resource partitioning, authentication and authorization are operations that are supported by the Active Node. Security makes sure that packets from different VEs are not mixed, and VE flows stay within their contract as defined by their SLA etc.

Resource control makes sure that the infrastructure is shared fairly among the different customers, while making sure that they are fully isolated from each other. Isolation involves the cooperation of a number of components in the network such as security, resource managers and policers etc. This is another contribution to the achievement of objective 1.

Deployment of different Types and Instances of EEs

A number of different types of Execution Environments (EEs) are available for example Java EE, Kernel based EE (PromethOS), SNAP (Active SNMP). These EEs run in different operational planes namely Transport and Control plane. The EEs must be deployed before the service components can be deployed within them.

In order to justifiably claim platform independent deployment of any given service, Active Service Provision (ASP) identifies which EEs are required to host which service components. The Active Node embodies the necessary mechanisms for EE deployment, and for the deployment of the service components. In some sense this is technology deployment followed by service deployment.

With technology (EE) deployment mechanisms the network is programmed to behave as required for any given service, so meeting objective 1 and indirectly objectives 2 and 3.

Creating and Operating Component-based EEs

The Active Node incorporates a component-based data path creation capability, implemented in a variety of ways to a single specification. A specific service comprises several components deployed and linked in meaningful ways. Flexibility in the network is achieved through the availability of different types of EE (component-based), and a variety of components.

By means of enabling technologies like Network Processors or Java VM we build EEs that can accept a service in the form of linked components that, in turn, may be introduced at different times. Defining and implementing these types of EE we are able to increase the degree of flexibility while allowing new functionality to be dynamically introduced. In this way we increase the pace of standardization thereby achieving objective 4 (speedier standardization) and objective 1 (extensibility).

Interoperable Infrastructure

With the need to deploy a service across different EEs and different platforms, the Active Service Provision system (ASP) identifies the different implementations of EEs and collaborates with VE manager to deploy the service components. The ASP performs these functions, using in combination two of the Active Node concepts previously identified (i.e. Deployment of different Types and Instances of EEs , and Creating and Operating Component-based EEs) to build (deploy) an interoperable infrastructure. In this way objective 3 is achieved.

3 THE FAIN DISTRIBUTED TESTBED

This section provides an overview of the FAIN active testbed, which serves as a permanent experimental network for active network technologies up to the end of the FAIN project and possibly beyond. The testbed is completely operational. In the remainder of this section we will describe the structure of the testbed, will precise where the different facilities and components are located and briefly describe the type of nodes running at various sites.

3.1 Active Network Nodes (ANN)

The FAIN testbed comprises different types of FAIN nodes:

- FAIN Active Network Nodes
- FAIN Element Management Station (EMS)
- FAIN Network Management Station (NMS)

A FAIN Active Network Node runs active services and contains programmable management, data and control planes. Two versions of this node exist, types A, and C and are described below in more detail. Nodes type B were planned at the beginning but during the course of the project it was decided not to implement them.

All node types exhibit the similar functionality vis-à-vis services and management components, i.e. they all support the active service provisioning facilities (ASP). They are different, however, in their respective Node OS architectures and performance characteristics.

3.1.1 AN Node Type A

Nodes of Type A are completely PC-based and provide active network functionality by the following components: VEM, RCF, DeMux, SEC, ASNMP and PromethOS. All the components are further described in deliverable D7.

3.1.2 AN Node Type C

Nodes of Type C (Hybrid Active Router) combine a commercial router Hitachi TC100 with an active network EE provided by a physically separate PC (attached PC). In type C nodes, the commercial router does packet classification and demultiplexing, while active packet processing is done on the attached PC. For this purpose, a Linux based NodeOS running java and SNAP execution environments will be operated on the attached PC.

3.1.3 FAIN Network and Element Management Stations

FAIN developed two types of management stations, the Element Management Station (EMS) and the Network Management Station (NMS). Both stations are based on PCs with Linux OS. As programming platforms both stations need OpenORB CORBA platforms over which management components are built.

FAIN currently allows only for one NMS per network. Therefore only one NMS will be operational during the demonstrations; however, some partners have set up their own NMS for testing purposes in their own realm.

One EMS may manage multiple active network nodes, which may be assigned dynamically. There are multiple EMSs on the Testbed (many partners decided to run an EMS to be able to locally manage their active network node while testing).

3.2 Network Topology and Interconnection

Figure 3-1 depicts the current topology of FAIN testbed, with four sites (ETH, FHG, UCL, JSIS) forming two core triangles and the rest of the sites connected as leaves to one of the core nodes. The decision about which node had to be a core node and to which core node the other nodes had to refer to was taken after measurements of the bandwidth and link quality between the different sites. Essentially, this is a three-level hierarchical tree topology with cross connections at the second level of the tree. The advantage of this topology in comparison with the full mesh is that the later provides only single hop paths between active nodes, while it may be more interesting to test applications over multi-hop paths. On the other hand, a tree with cross connections provides alternate paths between nodes, which is not the case with a simple tree topology. Finally, contrary to full or partial mesh, a carefully constructed tree topology accommodates for the fact that some partners have a lower bandwidth connection to the testbed either due to technical limitations or due to corporate security policy.

3.2.1 Tunnel Configuration

The FAIN testbed has been set up as an overlay (i.e. virtual) network on the existing network infrastructure. The overlay network is based on IP tunneling and is realized by appropriately configuring point-to-point tunnels between specific nodes. There are several different tunneling technologies and the choice of tunneling technology depends on the requirements. In FAIN, we have employed simple IP GRE tunneling, since the major requirement is to prevent interference of experimental traffic from the production traffic. We do not consider testbed traffic to be of sensitive nature (confidential), so there is no need to use IPSEC tunneling to protect this traffic while in transit over public Internet.

For the two tunnel endpoints, a properly configured tunnel looks the same as a physical point-to-point link, i.e. the nodes “think” they are directly connected, even though they use public Internet to communicate with each other.

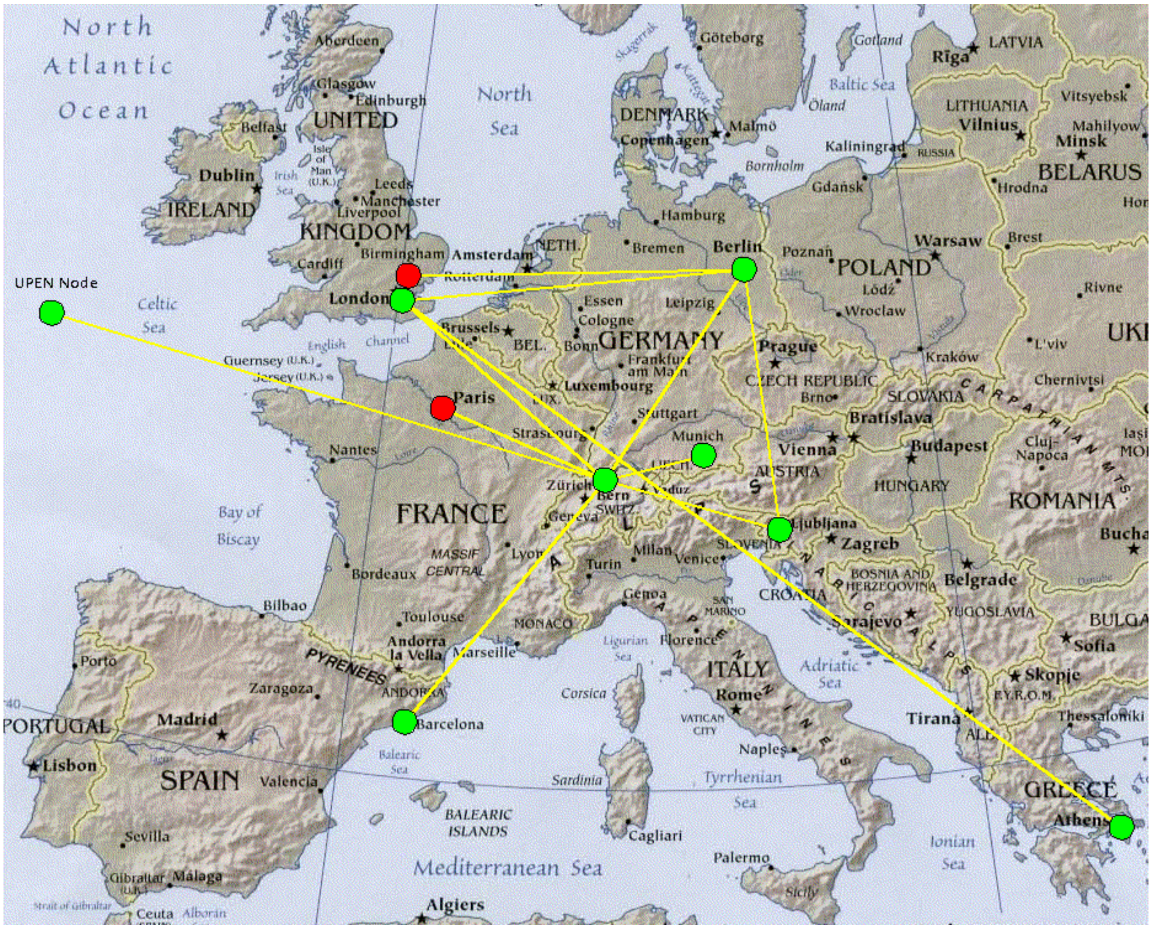


Figure 3-1: Testbed topology

3.2.2 Partner Network Data /Properties

Each partner site has:

- At least one node connected to the public internet acting as a tunnel endpoint
- A testbed subnet behind the tunnel endpoint with the address range of the form 10.0.p.0/24, where p is the partner number from Consortium partner list (in order of appearance in the FAIN Consortium partner list)

Partners can freely use the addresses from the private address range assigned to them.

3.2.3 Domain Name Service

There is a DNS service running within the testbed. The primary DNS server is hosted and maintained by FHG in Berlin and its IP address is 10.0.12.12. A secondary DNS server is hosted at ETH in Zurich and has the IP address 10.0.11.11.

All nodes and hosts within the testbed use .fa as the top-level domain. The FQDN names for hosts within the testbed have the following form

hostname.FAIN-partner-code.fa

For example, the host “onizuka” located at FHG FOKUS is called onizuka.fhg.fa.

3.2.4 Sites Overview

Figure 3-2 shows the actual status of the FAIN testbed. The bold lines represent the tunnels whereas the lighter lines are the links in the private networks at the partners' sites. EMS and NMS could be installed on either active or passive nodes.

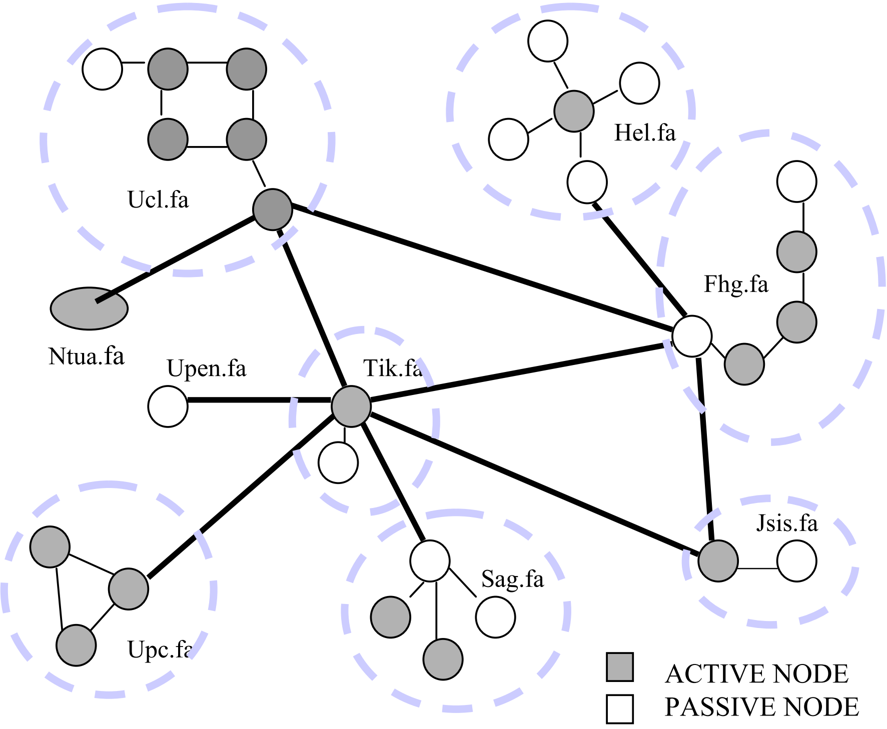


Figure 3-2: Nodes Overview

3.2.5 Monitoring Tool

The FAIN testbed is constantly monitored using a "ping-based" tool. It checks whether the tunnel endpoints and the most important active nodes are up, the main ports are open (e.g. the port where the CORBA naming server is running), and monitors the average delay, loss rate and bandwidth on the links.

4 SCENARIOS

4.1 Scenario Definition Framework

In order to define expressive scenarios, a framework, called *scenario definition framework* is introduced. As depicted in Figure 4-1 the framework consists of three layers. The lowest layer holds so called *Scenario Building Blocks* (SBBs), which are used to assemble the *Generic Application Scenarios* shown on layer two. On the top layer the *Application Scenarios* are located. Those represent instances of the Generic Application Scenarios.

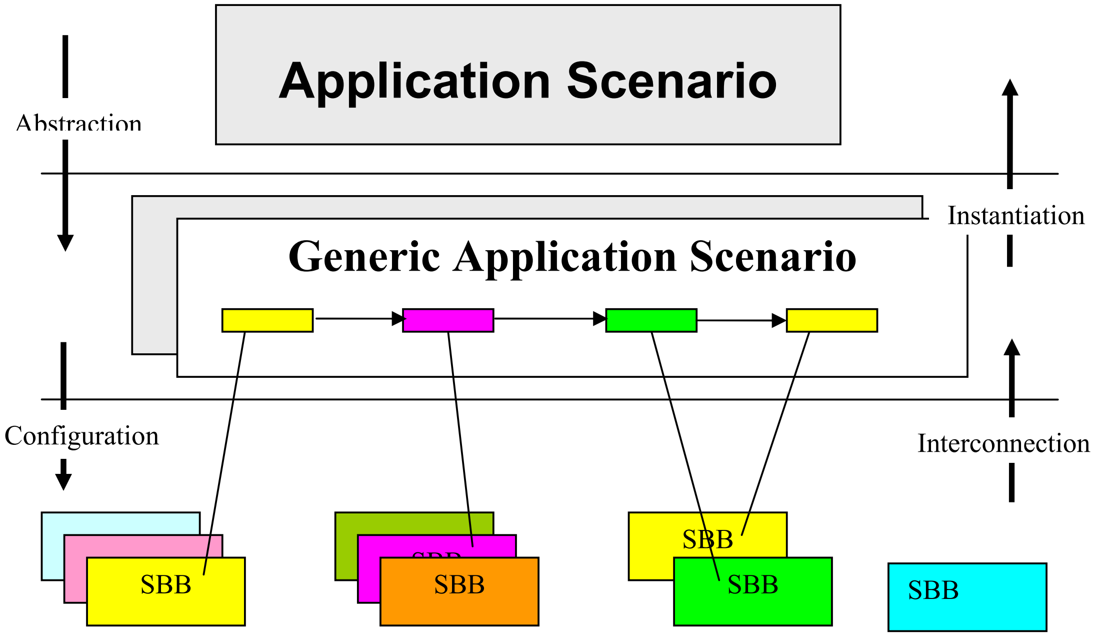


Figure 4-1: Scenario Definition Framework Model.

The framework provides transparency and abstraction on the upper layers and guarantees component reusability at the lowest layer. Implementing different Application Scenarios does therefore not result in individual implementations of per se identical functional concepts.
The functional concepts instead, are broken down to basic functionality that is expressed and implemented as SBBs. The requirements of an Application Scenario (or Generic Application Scenario) need only to be mapped to the corresponding SBBs.

4.1.1 Scenario Building Block

As mentioned previously a SBB expresses and implements elementary functionality of FAIN concepts. The SBB specifies the functional aspects of those elementary subcomponents, the interactions among them and the conditions they depend on.
SBBs are defined for reusability. They often depend on other SBBs. Their functionality should therefore preferably not overlap.
SBBs make up a Generic Application Scenario by sequential interconnection. An example for the interconnection is given in Figure 4-2. In order to keep the SBB specification simple, nested combinations of SBBs are not allowed.

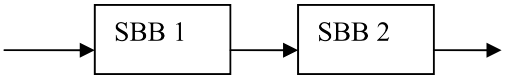


Figure 4-2: Sequentially Connected SBBs.

Typically SBBs rely on conditions or status they expect, in order to be able to perform correctly. Upon correct execution an SBB leaves a proper state that other SBBs are relying on. The definition and identification of SBBs therefore always specifies those pre- and post conditions.

4.1.2 Generic Application Scenario

From a bottom up perspective, a Generic Application Scenario consists of interconnected SBBs, each contributing with elementary functionality, to the overall functionality of the scenario. The Generic Application Scenario declares however, similar to an object declaration in programming languages, the functional scope and purpose of a scenario. This enfolds the FAIN concepts it is going to show, the premises it requires (in terms of hardware and software requirements), the protocol the scenario follows etc. Note, the Generic Application Scenario does not make assumptions that specify ascertained entities as its premises (e.g. like a specific testbed).

4.1.3 Application Scenario

The determination of the role of ascertained entities in the scenario is done by the Application Scenario. Using the analogy of objects in programming languages again, the Application Scenario is the instantiation of the Generic Application Scenario. The Generic Application Scenario is a theoretical and logical definition of a demonstration, i.e. of a scenario. In order to become a presentable demonstration it needs to be translated to a physical environment, which is in FAIN the testbed with its different sites and nodes. The mapping of the Generic Application Scenario to an Application Scenario consists in associating the logical entities and roles that have been identified and specified in the Generic Application Scenario to physical (and logical) entities and locations of the active network. The Application Scenario specifies the participating entities, the responsibilities and the protocol of the demonstration.

4.2 Scenarios Building Blocks

In order to fill the lowest layer of the scenario definition framework with a pool of SBBs, a closer look has been taken on three basic demands on FAIN. The first is concerned with the *Creation of Virtual Private Networks* including the *bootstrapping of the privileged Virtual Nodes*. The second handles the *Flow and Data Path Creation for Service and User Communication* and the last *the Deployment and Instantiation of Services and Service Components*. In addition to those 'basic' FAIN demands, security aspects of FAIN have been treated separately. They are nevertheless deployed integrally in the definition of generic application scenarios.
The basic demands and the security aspects permit to deduce the SBBs listed below:

SBB Calculate VAN describes how the ANSP calculates the topology of the VAN, identifying the involved ANNs, the corresponding profile and set up parameters of their respective VEs.

SBB Create VAN is in charge of assuring that all resources assigned to the SP are available all along the appropriate AN. All these resources are offered in the form of VEs.

SBB Activate VAN is used by the ANSP to activate VE instances, which have e.g. newly been created in the Create VAN SBB. The SBB permits to set the access rights of the SP for the VEs. In doing so management responsibilities are delegated and interferences with other SPs are prevented.

SBB Create MI permits the ANSP to configure the EMSs, which are responsible for the nodes on which VEs have been activated for an SP. This SBB enables the SP to manage its resources.

SBB Boot Node is used to boot an active node. Bootstrapping the privileged VN is a special process, because the VE management infrastructure is not available at this time. The process therefore relies on operating system support, e.g. scripts for starting the privileged VE may be included in the operating system's boot procedure.

SBB Contact Node describes how a client gets the reference to the initial port of the privileged VE in order to contact the node.

SBB Access Port describes how a port of a component is accessed by a client. This involves the authentication of the client

SBB Lookup Manager describes how a client looks up a manager for a particular service. The client provides a description of the service's template and gets in return a list of identifiers of matching managers. The client can use the identifiers to get the initial port of the managers.

SBB Create Instance permits a client to create a component instance of a particular service. In order to do so, the client specifies a resource profile. The manager uses this profile for checking the availability of the resource.

SBB Activate Instance describes how a component instance of a particular service is activated. The client may specify an initial setup for the component instance. After the activation he obtains a reference to the instance's initial port.

SBB Configure Instance permits to the client to configure a component instance. The configuration is done by setting properties of the component. Properties are pairs of names and values.

SBB Lookup VE Manager describes how a client is able to lookup the VE manager of a node. It is composed by several other SBBs. This building block is parameterized with the node's name.

SBB Create VE describes how a VE is created on a node. It is composed by several other SBBs. This building block is parameterized with the node's name and a resource profile for the new VE.

SBB Activate VE permits to activate a VE on a node. It is composed by several other SBBs. Parameters for the building block are the node's name, the ID of the VE (as returned by *SBB Create VE* – not to be confused with the VN ID), and the initial setup for the new VE.

SBB Data Path Creation describes the creation of a data path. A *Data Path* is defined as the path followed by information data within the network in order to be processed as part of a requested service. It comprises the service points, where the data is actually processed, and the connections between those points. The data path definition is closely related to the *data flow* concept, which is described as a *unidirectional stream of packets that have some common attribute(s) (such as source, destination, or protocol*). In an active network, the IP packets will also receive a specific service depending on the data flow they belong to. Thus, additional attributes specifically related to the active network may also characterize the data flow.
This SBB has been broken down in the following subSBBs:

> *SBB Service Deployment Request:* permits the client/customer to formulate the service it requests. It contributes to the separation of service deployment and service configuration.
>
> *SBB Requirement Translation:* The purpose of this SBB is to obtain the requirements associated to the data path elements given the service needs specified by the customer. These requirements are included in the data path descriptor.
>
> *SBB Service Customization:* The purpose of this SBB is to locate and enable appropriate resources according to the data path descriptor. In order to do so, the SBB has been split in the SBBs: *Service Component Deployment SBB* and *Node-level Data Path Creation SBB*.

SBB Data Flow Creation: The purpose of this SBB is to set up the data flows required to send data across an active network, while at the same time the data is allowed to be processed at specific points within such network. The creation of a data flow requires the establishment of appropriate routes in each network node included in the data path. In practice, this means the insertion of routes in the routing table and additional EE demultiplexing information.

SBB Data Flow Configuration defines the data flows required to send data across an active network, while at the same time the data is allowed to be processed at specific points within the network. The data flows are bound to a particular data path. The configuration of the data flow requires the establishment of appropriate routes in each network and additional EE demultiplexing information.

SBB Data Path Creation in PromethOS describes and implements the download, installation, and instantiation of service components in a specific active node in the PromethOS/Linux kernel space. The SBB starts when the VE Manager requests the creation of a VE, a EE, together with the instantiation of service components from PromethOS. The code modules are subsequently installed and instantiated. The job of the SBB ends when all the code modules are instantiated.

SBB Service Deployment starts when the Node ASP manager is requested to deploy a service. Service descriptors are fetched from the service registry and dependencies resolved based on the node capabilities. Code modules are fetched from the service repository. The code modules are subsequently installed and instantiated. This SBB stops when all the code modules are instantiated.

SBB Retrieve Information related to Service Deployment is responsible for retrieving service information.

SBB Processing of Information Related to Service Deployment preprocesses service information.

SBB Mapping describes and implements the mapping process of service requirements to available resources.

SBB Evaluation evaluates the results of the mapping process.

SBB Authentication Engine, which verifies the authenticity of active packets. The SBB depends on authentication data contained within an active packet and on crypto engine to do the necessary cryptographic operations. This SBB consists of the subSBBs:

> *SBB Active Packet Signing and Creation Option Building* describes how to sign an active packet and how to build the credential option. One of the principal's private keys is used for signing. In order to validate the generated signature, the credential option is build. The credential options points to the public key of the private counterpart that has been used to create the signature.
>
> *SBB Authentication in case of Active Packets* tells how the origin of data is authenticated by usage of a digital signature.

SBB Authentication of Sessions enables the authentication of peers at the beginning of a session. After authentication, entities exchange a session key and use symmetric cryptography with keyed hash for the provision of data confidentiality and service integrity.

SBB Security Manager accepts request from enforcement engines, collects request related to credential information and to policy information on accessed objects. The SBB communicates with the authorization engine in order to take authorization decisions. It provides NodeOS interfaces for managing credentials and security policies associated to AN users.

SBB Security Context, permits to setup and store security contexts, which are used by the SBB Security Manger. Security contexts represent all basic information about the component that is needed to perform various security related operations.

SBB SID Creation describes the process of creating a Secure Identifier (SID). In FAIN the SID is a 32 bit random number.

SBB Labeling describes the labeling process. FAIN components that implement a service are labeled during service start up with the label of a particular VE and service ID.

SBB Transition is used during activation phase of the VE creation, i.e. in parallel to the *SBB Activate VE*. In this phase the parent component is starting a component with a different VEId than its own. The *Transition SBB* re-labels it.

SBB Get Security Context by SID is responsible to extract the relevant security context for the *Security Manager SBB.*

SBB General Authorization Decision provides means to make authorization decisions. Three level of authorization decisions have been specified, the first is related to services and VEs. It provides a separation between them on the node. The second is related to capabilities and the third to component and port policies.

SBB Add Principal to Credentials Database provides facilities to add the principal to a credentials database.

SBB Remove Principal from Credentials Database removes the principal from the credentials database.

SBB Fetch principals credentials enfolds the process of fetching or getting credentials of the principal from the remote store or the local node cache.

SBB Get Principal Attributes covers the case when already resolved credentials and attributes are required, like in the case of *general authorization decision SBB*.

SBB Get Credential Information covers the case when information on the principal's credential (digital certificate) is used for e.g. the *SBB Active Packet Signing*.

SBB Resolve Principal Related Credentials covers the cases when resolved credential information is available as in cases of authentication for packets and sessions and registering the principal on the node by management entity. This information is inserted into credentials database.

SBB Get Principal Secure Store covers the need to keep and get from somewhere the information about the secure store of the principal, i.e. where principals keying material is kept. This information is highly sensitive so only selected components will have access to this information.

SBB Setting Principals Capability assigns a capability list to a principal.

SBB Set Component Capability assigns a capability list to a component.

SBB Removing Component Capability removes a capability from the components capability list.

SBB Removing Principal's Capability removes a capability form the principal capability list.

SBB Add Component Policy adds a component policy to a policy database.

SBB Removing Component Policy removes a component policy from a policy database.

SBB Low Level Engine decides whether the component accessing other components belongs to the same VE and service.

SBB Capabilities decides if the certain component has the capability needed to access component or component port (interface).

SBB Policy decides the certain component has access to a component or a component port (interface) regarding to the policy set (bind) to a component or component port.

SBB Management Based Exchange enables the exchange of Secure Associations (SA) between nodes in order to allow proper Connection Manager operation.

SBB Management Triggers Exchange provides facilities to trigger the exchange of SA (*exchange SA protocol*) between a managed node and its peer.

SBB Automatic Discovery and Exchange is triggered during the boot procedure of the node. It starts the peer neighbor search protocol and triggers the *exchangeSA* protocol with the discovered neighbors.

SBB Tearing Down SA handles the tear down of the active SA. In doing so, it prevents the flow of active packets between two nodes.

4.3 Generic Application Scenarios

The generic application scenarios are composed of sequentially interconnected SBBs. In order to keep the description of every scenario reasonable short, the participating SBBs will not be mentioned explicitly in the generic application scenario descriptions.
The purpose of the generic application scenarios is to outline the functional concepts of FAIN in an intuitive and enfolding manner. The generic application scenarios are therefore placed in circumstances that reflect requirements and demands of reality. As has already been told in the section about the scenario framework, generic application scenarios are not making any assumptions on ascertained entities as it premises.

The generic application scenarios defined in FAIN are:

- DiffServ Scenario
- WebTV Scenario
- Web Service Distribution Scenario
- Video on Demand Scenario
- Mobile FAIN Demonstrator Scenario
- Managed Access Scenario
- Security Scenario

4.3.1 DiffServ Scenario

Applications using the Real-time Transport Protocol (RTP) select their ports dynamically. Since service providers usually assign resources to specific applications in advance, they can therefore not identify the RTP flow in advance. The service providers need instead to assign the resources after the application has selected the port for the RTP flow. A method supporting this kind of resource allocation uses active packets.

The resources that need to be allocated are along the path the transmitted data takes. An active packet can thus easily identify the nodes involved in the data transmission in order to allocate resources. An example for RTP is video transmission. Another example applying to the dynamic reservation of resources is Internet shopping, where the protocol switches from http to https. In this example the service provider reserves bandwidth for the https connection.

Figure 4-3 shows an example using DiffServ and active packets for resource allocation. In this example video data is transmitted from a video server to a receiver client. The video stream takes the path over the active nodes AN (1), AN (3) and AN (4).
The video server chooses the port for transmitting the video stream in a first step. As next, the server sends an active packet that holds information for resource allocation and that tells the port number the video stream is going to use. In the depicted case, the active nodes AN (1), AN (3) and AN (4) are configured by the active packet. The active node AN (2) is not affected since the video data is not transmitted over active node AN (2). The shown procedures are executed in order to allocate resources for video transmission from the beginning. Nevertheless they do not guarantee that the video flow does not get impaired during transmission. The remedy for such cases is to dynamically resend active packets.

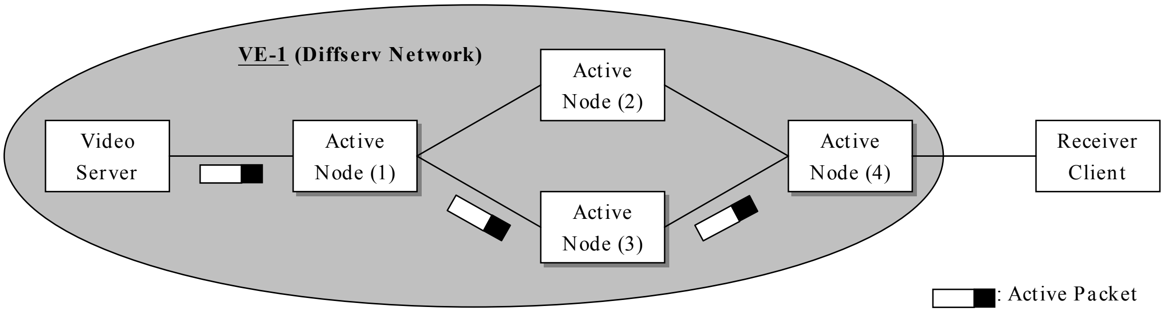


Figure 4-3: Active Network Node Configuration by Active Packet

Architecture/Setup

In this application scenario, a service provider (SP) agrees on a contract with an ANSP. The contract ensures him three levels of priority transmissions, each identified by a distinct DiffServ Code Point (DSCP).

The SP interconnects the three parties A, B, C as shown in Figure 4-4. Traffic entering or leaving A and C pass through the hybrid routers HANN-1 and HANN-2 respectively.

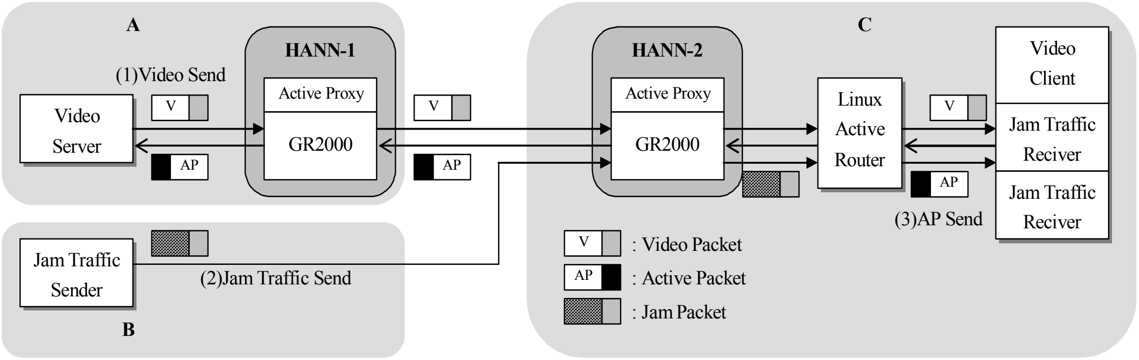


Figure 4-4: DiffServ Demonstration Scenario.

Party A sends a video to party C with low transmission priority. Some time later party B starts sending ‘jamming’ traffic to party C. The jamming traffic is identified by a DSCP guaranteeing higher priority over the video traffic. As long as the jamming traffic doesn’t exceed the output bandwidth of HANN-2 nothing happens. As soon as it exceeds the output bandwidth, user C sends an active packet to A. The active packet is authenticated and authorized at each active node, which it traverses (i.e. at HANN-2 and HANN-1). Upon arrival at A a SNAP program changes the DSCP of the video stream, so that it gets higher priority over the jamming traffic.

4.3.2 WebTV Scenario

An SP, called WebTV-SP, offers a WebTV service to its customers by broadcasting the video program in the Internet. The customers are able to watch the video irrespective of their terminal capabilities. In order to do so, the WebTV-SP requests from the ANSP to set up an Active Virtual Private Network (AVPN) wherein he can deploy services that are customized to customer requirements. Customers need only to subscribe to the WebTV service by contacting a server of the WebTV-SP.

In the scenario, one customer uses a terminal that is not capable of displaying the video stream correctly, e.g. it uses a handheld device with low processing power and low access bandwidth. The WebTV-SP pre-processes the video stream for this particular customer by transcoding the stream into a format understood by the handheld.
Based on a SLA between the ANSP and the SP, policies are sent to the ANSP MI. As a result the ANSP PBNM receives a QoS policy and enforces it on both the NMS and at all appropriated EMS. The active node management framework creates a new Virtual Environment (VE) for the WebTV-SP. If the VE creation is successful the ANSP PBNM enforces a delegation policy through the NMS and all appropriated EMS. The enforcement requests the active node management system to activate the newly created VE.
The ANSP creates a Management Instance (MI) in all the appropriate EMS stations for this WebTV-SP and sets access rights at the active nodes interfaces. The WebTV-SP is now ready to configure his AVPN by sending policies that are specific to customers. The WebTV- SP also installs the transcoder and duplicator service components. In addition, the WebTV-SP deploys service-specific policies in the WebTV-SP PDP of its MI, so that he can define its own service-specific policies that will be enforced in the active node. As the last step the monitoring system is used for the reconfiguration of the transcoder at runtime. This if used when the access bandwidth changes dramatically and the end-user needs a different transcoding format on the video stream for example.

Architecture/Setup

The transcoder and the controller components are deployed via the ASP mechanism upon request in the SP's VE. The deployment is triggered by the SIP proxy when the SIP request from the customer arrives.
The creation of the VAN for the WebTV-SP entails that a management instance is created and configured. There is one MI at network level and one at element level. At the beginning only the PDP Manager is instantiated inside each MI. When requests are forwarded to the MI, the PDP Manager may decide to trigger the deployment of specialized functional domains by contacting the ASP in order to deploy SP PDP into the EMS, and SP PEP into the SP-VE.

At bootstrap of this SP PDP a set of alarms/event is configured by means of policies, which configure the monitoring system in order to receive those events/alarms from transcoder controller.

4.3.3 Web Service Distribution Scenario

For the full and extensive documentation of the Web Service Distribution Scenario please refer to [2].

In the Web Service Distribution Scenario, Web (i.e. HTTP) traffic is distributed and redirected within the network among several distributed servers in order to provide reliability, performance and scalability for web services.[1]
The web service infrastructure exploits the capabilities of AN technology as follows:

[1] The term „Web Service" is often also used to refer to approaches for describing, finding and invoking objects and their services with web-based languages and protocols, e.g. Microsoft's .NET. We use the term "Web Service" to refer to an application service which is offered to an end-user, as it has been invented in [1].

- A Web service provider has access to Active Nodes, called “service nodes” or “Active Web Servers”. The provider can install content and service logic onto those nodes and by such means he can implement features such as content distribution and personalization, aggregation of user replies or fast response times.
- Other active nodes, called “redirection nodes” are also available. They provide features which allow to filter HTTP traffic, to build service sessions (i.e. to deal with per-user state) and to forward the traffic of a particular session to a service node. Onto this nodes, code is downloaded which observes the network load, observes the load and availability of servers and based on this information determines a strategy for redirecting the traffic to the service node which is most suitable for a given web service/user.

The overall scenario is depicted in Figure 4-5. Both redirection nodes and service nodes will usually be physically located within the access network of the end user, the access network of the web service provider, or connected via a separated access network. For this reason, we may also call both of them “Active Web Gateways”. Note that we assume that the core network is non-active and only provides basic IP connectivity.
The overall setting is fully transparent to the end user, i.e., the end user uses the web service via an ordinary web browser, using standard protocols such as IP or HTTP.

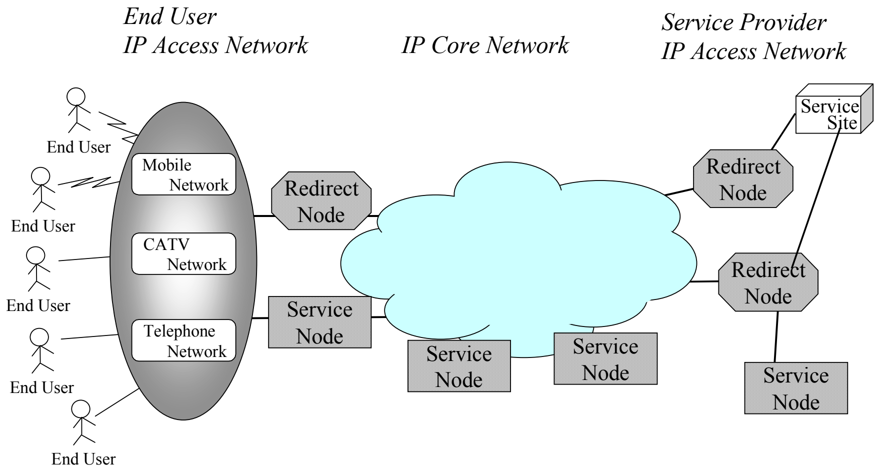


Figure 4-5: Service nodes and redirect servers implement active web services

Benefits of AN technology shown by the scenario include among others:

- *Network aware web services:* Contrary to existing web services, they can operate based on information such as available bandwidth on access and network links, network topology, and load of servers. Because the information is not available at the client- or server side, it should be implemented with AN technology within the network.
- *Ease of service programming and management*: Active networks eliminate the need for cumbersome ad-hoc solutions to specific problems which require separate management; active networks provide common ground for deployment of new services and mechanism inside the network and unify the management of these mechanism and services.

- *Distribution of service logic*: Service logic is executed at several locations, including specific points inside the active network, which is potentially advantageous for large volume services, fine (per user) granularity of services, new service features, etc.. With existing solutions such as caches or content distribution networks, only content, but not service logic can be located within the network.
- *Dynamic, autonomous adaptation*: The task of operators on service site (service provider, network provider) is getting bigger as the number of network customers/consumers is growing. So the task of provisioning should be dynamic. Besides active nodes may cooperate with each other.

4.3.4 Video on Demand Scenario

PromethOS is a Linux kernel-space NodeOS for active nodes. It is managed from Execution Environments (EE) in user space. Since the Virtual Environment Manager (VEM), which is an EE in user space, is also concerned with node management issues, an integration of both, VEM and PromethOS, becomes indispensable for the interoperability of different EE types. The video on demand scenario demonstrates this integration of VEM and PromethOS. It shows how the VEM management interface of PromethOS' user space library has been enhanced in order to control PromethOS.

The ANNs have been designed to support multiple VEs, EEs and EE instances. The scenario demonstrates therefore also how PromethOS is supporting multiple VEs and how it is able to differentiate among the customers by using those VEs. For the scenario, one EE per VE is instantiated in kernel space. The EE runs a Wave Video plugin, which scales its functionality according to the different requirements of the customer.

Architecture/Setup

At one site (e.g. service provider) an active node and a source for a video stream are located. At a different location, the video receiver is installed. Between the source and the active node, a high-bandwidth link provides sufficient bandwidth. The link models a high-bandwidth backbone.

The active network node and the video receiver are connected by two links with different bandwidth. The capacities of the links reflect different Service Level Agreements. The active network node is supposed to adapt the high bandwidth requiring input stream according to the pre-set output capacities of the output streams.

At boot time, the ANN is running the VEM and the management components of PromethOS. As a customer request arrives at the ANN, the VEM orders the deployment of a VE, i.e. it assigns resources to the customer, and it orders the deployment of an EE where the Wave Video scaling plugin is installed.

The request to initiate the service deployment is implemented by the FAIN-WP4 service specification. The service specification is parsed and resolved by the Service Creation Engine. The code required for the Wave Video plugin is fetched from the code server and deployed on the ANN.

As a second request form a different customer arrives, the ANN creates a second instance with the same configuration but with different resources assigned to the VE. No additional code fetching is required anymore, since the code is available from the nodes local cache.

The creation process is fully implemented and controlled by the VEM. As the process completes, the video source gets a signal to begin with the transmission of the video flows.

4.3.5 Mobile FAIN Demonstrator

This generic application scenario introduces a "Wireless LAN" show case, which is implemented within FAIN. It shows how mobile networks benefit from FAIN concepts. A more detailed description can be found at [3]. It describes the idea of a "FAIN Dino Park".

The show case "FAIN Dino Park" was made up, because it shows an interesting and promising use of mobile wireless network technology within the edutainment domain. Especially for mobile wireless networks where the bandwidth isn't abundant, the FAIN concepts exploit their advantages. The use cases of the "FAIN Dino Park" demonstrate how in a challenging wireless environment load balancing and load reduction approaches succeed in avoiding bottlenecks and how they could improve edutainment concepts.
The focus of the demonstration is on load balancing and load distribution in mobile networks. The show case is motivated by already installed famous amusement parks, but additionally it is equipped with a WLAN infrastructure based on products commonplace in the market. To realize load balancing and load reduction concepts, some additional software components are implemented and installed on the servers.

Architecture/Setup

In total, there are 3 different use cases, which demonstrate load balancing and load reduction. Each use case is illustrated by demos. The use cases and the related demos are listed in the following.

Use Case 1: shows a redirection of a connection. The connection to a heavily loaded access point is terminated and a new one is built up between the connecting client and a less burdened access point during peak-period demand whiles the registration.

Use Case 2: This is a similar use case. It shows a simple redirection of a connection request. The connection is built up between the requesting client and a less burdened access point. The load at the burdened access point results by multiple request of video streaming.

Use Case 3: This generic application scenario shows how a personal mobile SW proxy contributes to load balancing by mechanism of load reduction in the overall show case of the FAIN Dino Park.

Each of the cases relies on following components:

- A WLAN Access Controller, which is the central Control Unit. It is a software component and is deployed on a FAIN PromethOS active node.
- At least two WLAN access points, to which the clients are linked by a wireless link. Via these the clients receive data from a content server.
- At least one FAIN active node, which is PromethOS capable. Here, the following components are deployed:
    - PromethOS: A framework for the manipulation of Linux Kernel modules (PromethOS modules) and for redirecting network traffic to such a PromethOS module.
    - Netfilter: A framework for the manipulation of network packets. Netfilter is used by PromethOS.
    - Iptables: This is the user space part of Netfilter. It is used to load and configure Netfilter modules and to redirect network traffic to such a module. PromethOS uses an extended version of Iptables which is able to load and configure PromethOS modules.
    - PromethOS modules used by WLAN access controller: A PromethOS module is used for the actual manipulation and the monitoring of the network traffic.
    - User interface: It is used to configure and monitor the PromethOS module.

- A server providing content used for the demonstration of actual data transfer via the WLAN access points. In the context of FAIN Dino Park, such a content server is dedicated to an exponent or specifically a designated server for registration procedures at the entrance.
- At least two terminals, which are notebooks or may be PDAs, each with WLAN capabilities. The terminals are equipped with a FAIN Terminal Daemon.
- Optionally a load generator is required creating the background traffic which is used to trigger decisions on load balancing. Instead of using a load generator, the handover between access points may be demonstrated by simulating the load in the WLAN access controller.

The general infrastructure of the demos is shown in Figure 4-6:

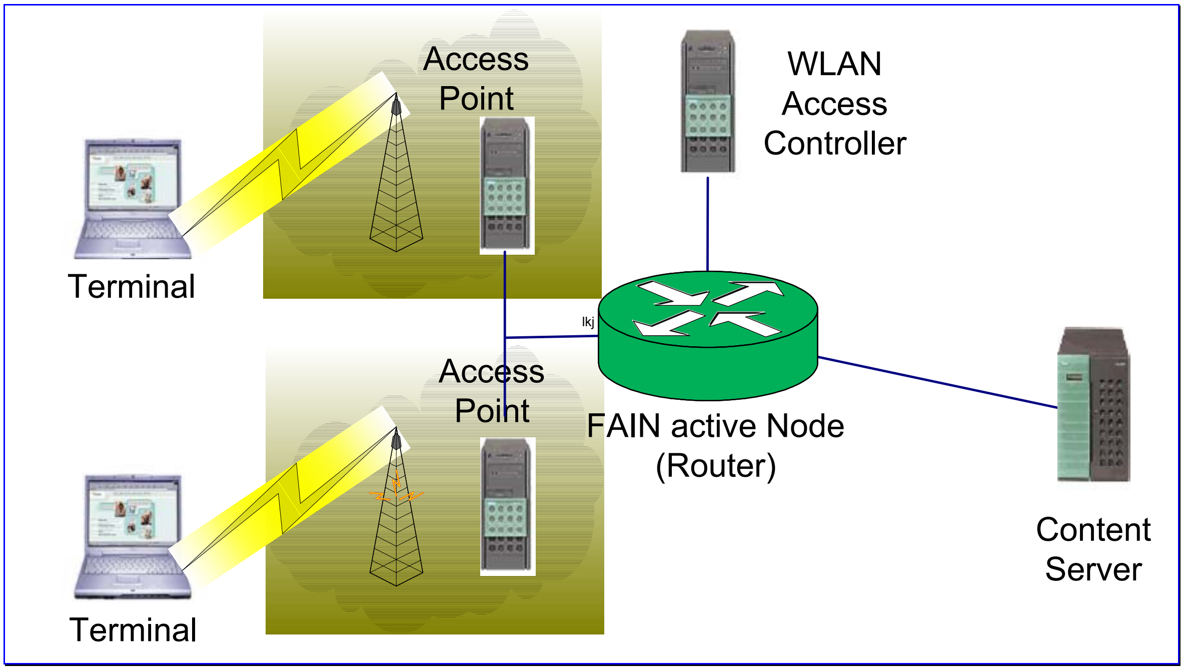


Figure 4-6: General Infrastructure of the Demos

4.3.6 Managed Access Scenario

A host joining a network is assigned a set of network services according to some SLA, e.g. all hosts are allowed to use DNS but only a subset of them is allowed to use the SNMP services provided by the SP. Currently most access providers only manage their own equipment. They assume that the customer has his own arrangements with the rest of the network, the service providers.
In this generic application scenario active packet technology is used to manage packet filters across networks that belong to different SPs.

Architecture/Setup

There are three road warrior laptops. Each wants a different type of network access from the other. The road warriors access the private IP network using a WaveLAN access point. These access points generate SNMP traps when a new host has been assigned an IP address. The IP address for a host is allocated by a DHCP server.
The DHCP server operates on a network controller host. This host also supports an SNMP trap collector and an injector. The injector is a process that can be used to inject active packets into the network. The router of the access network performs packet filtering and conditioning. It has an interface to the backbone network which has two routers for egress and ingress. The former performs network address translation of packets with private IP source addresses to the address of the public interface of the egress router. The ingress router performs NAT for packets from elsewhere to services hosted in the private network.

The active packet is injected and reconfigures all the routers to accept packets for different services from the road warriors on the access network. The active packet could reconfigure all of the routers if need be.
The value added by using active packets for this scenario is that the network elements can be controlled on demand. Normally, network administrators would put a static configuration in place.

4.3.7 Security Scenario

Showing a pure security scenario immediately results in quite artificial settings, especially since security is an integral part of the FAIN architecture. These are the reasons why the security scenario has been hold very general. As such it may easily become a component of any of the above presented generic application scenarios.

The security ‘scenario’ shows how an active packet is passed through a node and how it triggers security concepts. The scenario is applicable to any active packet approach in transport, control or management plain. The scenario consists of the following steps, which are repeated on every network node that the packet traverses:

1. The DeMUX intercepts the packet and invokes security receive check function with the ANEP packet, UDP protocol information data and with the local information (service), where the packet is headed to.
2. The ANEP packet is parsed.
3. The hop integrity option is evaluated:
    a. The correct Security Association (SA) is chosen.
    b. The packet hop replay information is validated.
    c. The integrity token is verified.
4. If the packet contains one or more credential options:
    a. The principal credentials are looked up in local cache.
    b. If they are not already there, they are fetched from the previous node.
    c. The credential path is validated.
    d. The digital signature of the static part of the packet is verified with the trusted public key of the principal.
    e. The credential option time frame is checked.
5. If the packet contains active code and verification is required (e.g. due to a policy), the code is verified.
6. The packet security context(s) is/are build from the principal credentials and results of the packet code verification.
7. The security context of the packet is compared to the security context of the packet destination. If the packet destination channel has defined a policy the security context(s) is/are used to authorize the access to the service.
8. If the access is authorized the packet is returned to the DeMUX,

9. The DeMUX passes the packet to the service.
10. The packet data (variable and payload, code or data) is evaluated in the service.
11. If the evaluation results in an action regarding the node, this action is authorized and the policy, if one exists, is enforced.
12. The packet is returned to the DeMUX and the security check function is invoked.
13. The active packet is build.
14. The next hop integrity option is built:
    a. Regarding packet next hop destination the right SA is chosen.
    b. Replay protection value is added.
    c. An integrity token is built and the hop integrity option is added to the packet.
15. If every thing went successfully, the packet is returned to the DeMUX and in order to be sent to the next hop.

Architecture/Setup

Any network topology with three or more active nodes will be sufficient. Requirements are the installation of the basic FAIN node services, i.e. of management, DeMUX and security facilities. Credentials and related key pairs have to be generated, SAs between the nodes has to be established before the scenario. If the policies are used they have to be supplied during the service setup or set by the network management framework.

4.4 Application Scenarios

The application scenarios are instantiations of generic application scenarios. Since they are used to present the achievements of FAIN they are sometimes referred to as demonstration scenarios. The terms are used interchangeable in the following sections. Note that not all generic application scenarios will be instantiated to application scenarios.

4.4.1 DiffServ Scenario

The detailed version of the DiffServ Scenario can be found at [8]. In this DiffServ demonstration scenario, a service provider (SP-1) tries to make a priority transmission network by renting network resources from an operator (ANSP). The SP-1 makes a contract to rent three levels for the priority transmission with the ANSP. If one assumes that those three levels are DSCP-1 (Differentiated Service Code Point-1), DSCP-32 and DSCP-224. The SP-1 connects a branch office-A and a head office through HANN-1 (Hybrid Active Network Node) and HANN-2 as shown in the Figure 4-4. In addition, it connects a branch office-B and the head office through the HANN-2. Then SP-1 assigns the DSCP-1 and the DSCP-224 transmission qualities between the branch office, A and the head office. In addition, it assigns the DSCP-32 transmission quality between the branch office-B and the head office. Initially, the user, A in the branch office, sends video data to a user, C in the head office, through the network with a DSCP-1 transmission quality. Then user B, in the branch office, sends "jamming" traffic to another user C with a transmission quality of DSCP-32. The priority of the DSCP-32 is higher than that of the DSCP-1. If the amount of video data and jam traffic is above the output bandwidth of the network node (HANN-2), the video data transmission will be impaired, since the priority of the video is less than that of the jam traffic. Then user A changes the priority of the video data from the DSCP-1 to DSCP-224 by an active packet (a SNAP program). The active packet is sent from user C to the user A. The authenticity and authority of the active packet is checked at each HANN. After changing the priority of the video data, it will no longer be impaired.

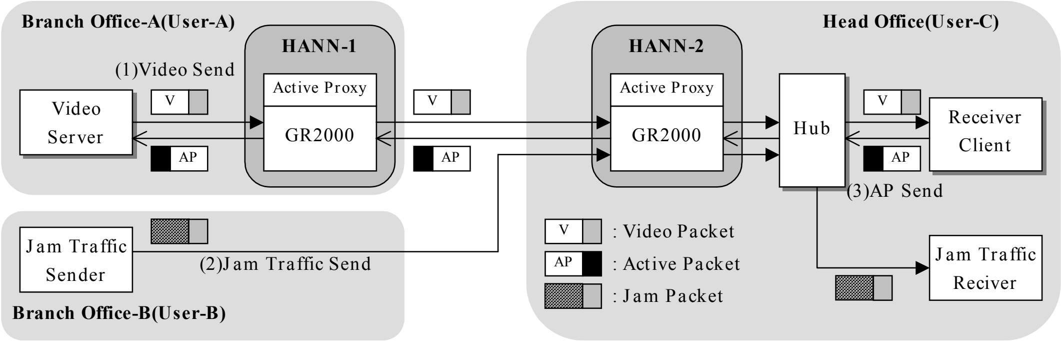


Figure 4-7: DiffServ Demonstration Scenario

Mapping onto FAIN Testbed

As shown in Figure 4-8 the SP connects the HEL site and the FHG site by a GR2000 hardware router and Active Proxy (HANN-1) and the TC-100 hardware router and Active Proxy (HANN-2). The SP assigns DSCP-1[2] to the video flow from HEL and DSCP-32 to the jam flow from the jam sender at FHG. The GR2000/TC-100 hardware router transmits the video flow and the jam flow with the associated priorities. In the shown case the jam traffic has high priority. If the jam traffic consumes the available bandwidth for an output port the video flow will experience packet losses.

If this occurs a SNAP sender at FHG injects a SNAP packet with the objective to change the DSCP value of the video flow at HEL. In fact the DSCP value of the video flow changes from DSCP-1 to DSCP-224, which results in a higher transmission priority for the video stream compared to the jam traffic.

[2] In the implementation of the scenario, the 6 representation of DSCP are mapped to the 8 Bit representation of the TOS (Type of Service) that is used by the GR2000 routers.

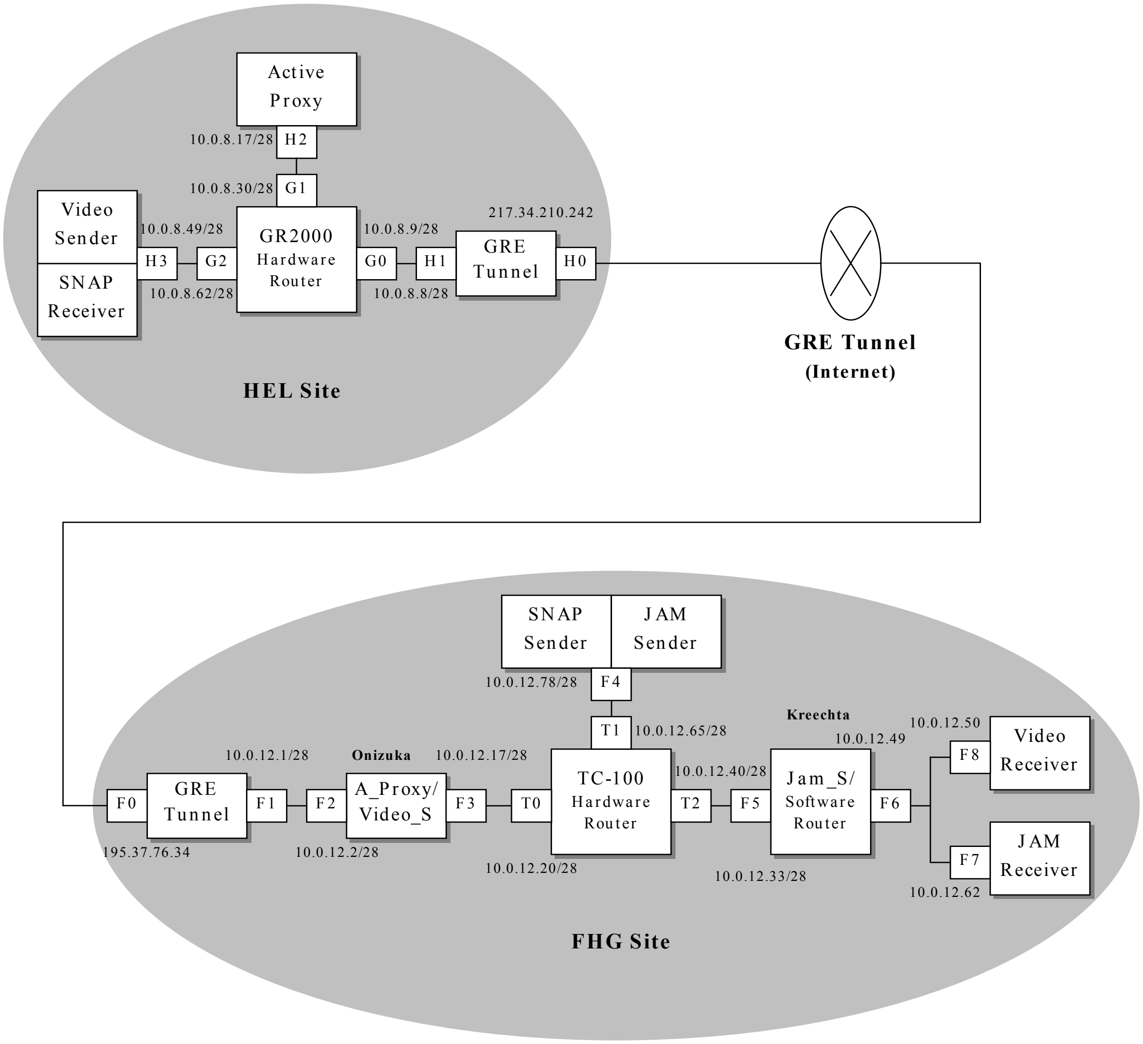


Figure 4-8: DiffServ Network Configuration

Demonstration Steps

- **DiffServ demonstration with GR2000 will be shown as follows:**

1. The privileged VE instantiates the basic service components on the Active Proxies. (VE function)
2. The privileged VE creates a new VE with some resources and components on the Active Proxies. When SNAP-EE is created, VEM requests DeMUX for registering of SNAP-EE by the I/F(1). (VE function)
3. A video sender in the HEL test-bed starts sending video data to the receiver in the FHG test-bed.
4. The jam traffic sender in the FHG test-bed starts sending jam traffic from the port F4 to the receiver's port F7. It intentionally creates network congestion at the port T2.
5. The monitoring component (DiffServ) detects packet discards by receiving a SNMP trap from TC-100 hardware router in FHG test-bed. (Monitoring function)

6. The SNAP sender injects a SNAP active packet, which is encapsulated as an ANEP packet, to video sender. SNAP packet code is stored in variable option and its fingerprint in ANEP payload is attached. (SNAP/SEC function)
7. DeMUX in Active Proxy intercepts the ANEP-SNAP packet and retransmits it to Security component through the I/F (2). (DeMUX function)
8. Security component guarantees integrity of the ANEP-SNAP packet, authenticates the data origin and sending principal, verifies the SNAP packet fingerprint, builds a security context of the packet and returns a verdict to DeMUX by the I/F(2). (SEC function)
9. DeMUX accepts or discards the ANEP-SNAP packet depending on the verdict from Security. (DeMUX function)
10. DeMUX retransmits the ANEP-SNAP packet to SNAP-EE by the I/F(3). (DeMUX function)
11. SNAP-EE gets the hardware router's configuration from the ANEP-SNAP packet and sends a request to the DiffServ controller by the I/F (5). (SNAP function)
12. Request is authorized by SEC. (SEC function)
13. DiffServ controller receives configuration data from SNAP-EE and configures the hardware router for setting a DSCP (TOS) value for a flow by the I/F (6). (RCF function)
14. The video and jam data pass through the hardware router with their assigned DSCP (TOS) codes.
15. SNAP-EE sends the previously received ANEP-SNAP packet to DeMUX by the I/F (4). (SNAP function)
16. DeMUX sends the ANEP-SNAP packet to Security component by the I/F (2). (DeMUX function)
17. Security component builds the packet external representation as ANEP header of the SNAP and returns it to DeMUX. (SEC function)
18. The DeMUX component sends the ANEP-SNAP packet to the next active node. (DeMUX function)

At the next active node, the procedures from 7) to 18) are repeated.

- **DiffServ demonstration with Linux-based router will be shown as follows:**

19. Another jam traffic sender starts sending jam traffic from the port F6 at the Linux based router and congestion is occurred at the out put port.
20. The SNAP sender injects a SNAP active packet to the Linux based router. SNAP packet code is stored in variable option and its fingerprint in ANEP payload is attached. (SNAP/SEC function)
21. The same procedures from Step 7 to 10 above are executed.
22. The SNAP-EE gets a Linux router's configuration and sends a request to Traffic controller by the I/F (5). (SNAP function)
23. Request is authorized by SEC. (SEC function)
24. Traffic controller configures the Linux router by the I/F (6), assigns specified bandwidth to a flow with specific DSCP (TOS) value. (RCF function)
25. The video and jam data pass through the Linux software router with their assigned bandwidth.

Demonstration Results

In this demonstration the video flow has been assigned bandwidth resources and a high priority transmission dynamically. In order to provide these facilities the VEM creates a new VE and basic components such as the DeMUX, the SEC and the RCF components including the DiffServ controller and the traffic controller. In addition a SNAP-EE has also been created. The SNAP-EE injects a SNAP packet, which is encapsulated in ANEP, and DeMUX intercepts it for the configuration of the active nodes. The integrity of the ANEP-SNAP packet is guaranteed by SEC functionality. The demonstration shows that FAIN active networks and nodes provide secure and dynamic network configuration.

4.4.2 Security Scenario

The security scenario will be shown in the same context as the DiffServ scenario. It will therefore run with the same configuration and topology as shown in Figure 4-8.

Demonstration Objectives

The security architecture is present on every active node in the scenario, namely on the hybrid nodes and in video/SNAP sender. The security architecture offers two system level mechanisms to protect active packets in the network, i.e. per hop protection and end-to-end protection. Per hop protection requires that neighbor nodes share security associations. End-to-end protection is based on digital signature mechanisms. It requires trusted certificates of entities on every node in order to issue users credentials. For a service as described in this scenario, suitable credentials will be issued to the service users. The credentials will specify the service, the corresponding VE, the time of validity and the possible user role in the network (i.e. if it is a common user, a manager, or an observer). The credentials are signed by trusted entities. They associate the public key of the user with her authorization data.
For internode communication per hop protection provides authentication of the neighbor node. This suffices the requirements of protocols that need e.g. to fetch user related credentials from the previous node.

There are two types of objects (i.e. targets) that can be protected in this scenario with the mechanisms designed and implemented in FAIN, namely *input/output channels* and the *traps that the SNAP code raises in the DiffServ trap receiver*. For these targets the security policy has to be set in the target security context.

For the target of the channels, the VE and service (EEId) have to match explicitly. It is possible to set a policy for a channel that only a certain user group can access.
For the second target, the DiffServ trap receiver context has to fit into the security policy, which governs user access to the environment API. Every access from the SNAP daemon to the environment is authorized according to the security policy and user supplied authorization data carried in the active packet.

On system level, the security architecture is integrated within the component framework of the node management system. Credentials and security policies are provided for pVE and VEs hosted at the node. The decisions on who is allowed to manage, to monitor or to observe components or their ports on the node, are taken by referring to policies.

Special attention has been devoted to protect SNAP packet in the network. With SNAP, as in-band approach, the packet content can be legally changed in the network. Due to this fact, the requirements of end-to-end protection are not fulfilled by only applying a digital signature from the originator of the packet. In order to protect security relevant data in the SNAP packet the verification framework is used therefore.

When the user originates a SNAP packet the packet is encapsulated in ANEP and fingerprinted for security relevant commands, which are stored in the ANEP packet payload. The SNAP packet is stored in the variable *ANEP option*. The ANEP payload is covered by the digital signature. While passing the nodes in the network, the SNAP packet is fingerprinted again and the fingerprint is compared to the payload. This verification allows to detect modifications on the payload, like reordering or multiplications of the security relevant commands.

Architecture/Setup

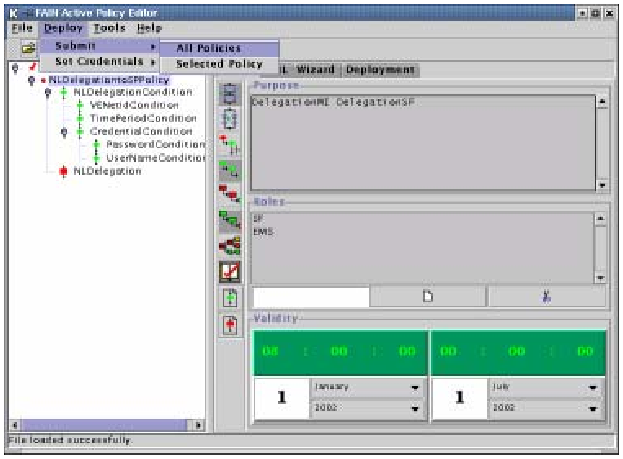
The security scenario will be shown complementary to the DiffServ Scenario. The security scenario will show operations on the security architecture. In particular how, the nodes and node resources can be protected from malicious or unauthorized usage, how active packets can be protected in the network from unauthorized modification, spoofing or replay attacks.

As complementary part of the DiffServ scenario, those operations of the security architecture will be shown as the output to local terminal. The external representation of the packet will be shown together with the processes of building and parsing it. Furthermore the processes concerned with credentials, the access control checks, operations on credentials cache, the verification of the SNAP code and other general operation of the security architecture will be shown in the context of the FAIN component model.

4.4.3 WebTV Scenario

An SP, called WebTV-SP, offers a WebTV service to its customers by broadcasting the video program in the Internet. In the scenario, one customer uses a terminal that is not capable of displaying correctly the video stream, e.g. it uses a handheld device with low processing power and a low access bandwidth. The WebTV-SP pre-processes the video stream for this customer by transcoding into the format understood by the handheld.

As a result of an SLA agreed between the ANSP and the SP, policies are sent to the ANSP MI. These policies are edited using the GUI of the Policy Editor as depicted in Figure 4-9. Consequently, the ANSP PBNM receives a QoS policy and enforces it on both the NMS and the EMS. This results in invoking the active node management framework to create a new Virtual Environment (VE) for the WebTV-SP. If the VE creation is done successfully, then the ANSP PBNM enforces a delegation policy through the NMS and the EMS. This enforcement consequently requests the active node management system to activate the newly created VE. The ANSP then creates a Management Instance (MI) in all the appropriate EMS stations for this WebTV-SP and assigns the access rights to the active nodes interfaces.



DelegationMI DelegationSF

SP
EMS

08 : 00 : 00 1 January 2002

00 : 00 : 00 1 July 2002

File loaded successfully.

Figure 4-9: GUI of Policy Editor

The WebTV-SP is now ready to configure his AVPN by sending policies that are customer specific. The SP also installs service components into the active nodes (a transcoder component in our scenario). The service components are deployed by the ASP system based on the service descriptors.

In addition, the SP deploys service-specific policies in the QoS PDP of his MI after the deployment of the transcoder service component in the active node. In this way the SP can define its own service-specific policies that will be enforced in the active node.

Finally, the monitoring system is used for the reconfiguration of the transcoder at runtime, when for instance the access bandwidth changes dramatically and the end-user needs a different transcoding format on the video stream.

Mapping onto FAIN Testbed

Figure 4-10 shows how the WebTV scenario is mapped to the FAIN testbed.

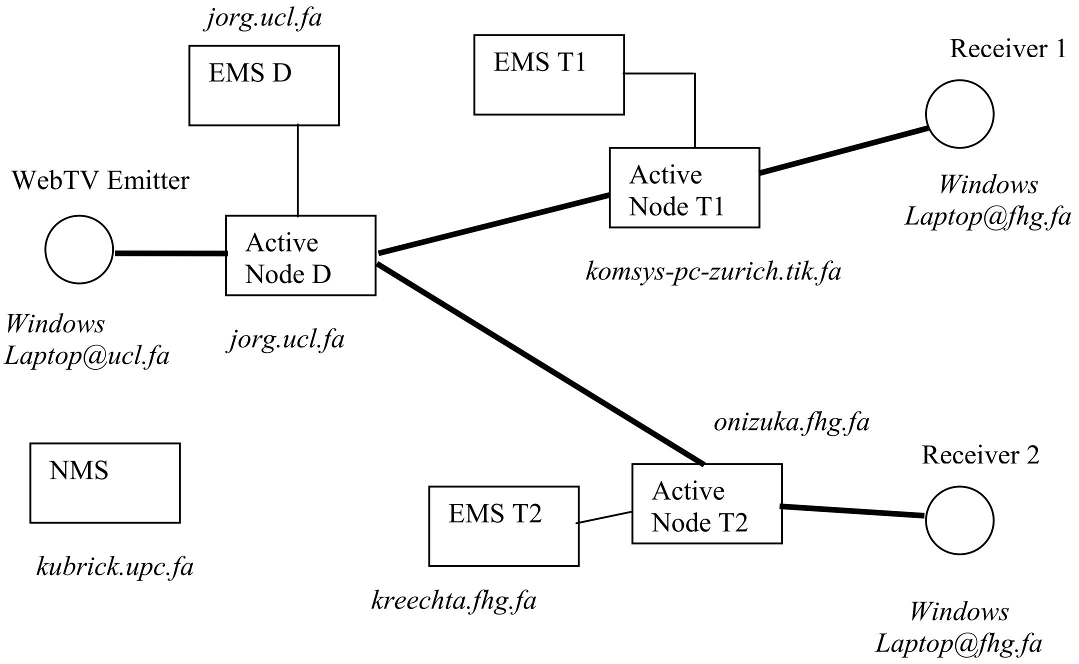


Figure 4-10: WEB TV mapped to Testbed

The components involved in the scenario are:

1. Windows/Jmstudio – emitter (windows laptop at UCL)
2. Windows/SIP Video Client – receiver 1 (windows laptop provided by FT)
3. Windows/SIP Video Client – receiver 2 (windows laptop provided by FOKUS)
4. Active node D – duplicator (jorg.uc.fa)
5. Active node T1 – transcoder 1 (komsys-pc-zurich)
6. Active node T2 – transcoder 2 (onizuka.fhg.fa)
7. Network Management Station (kubrick.upc.fa)
8. Element Management Station D (a Debian machine at UCL)
9. Element Management Station T1 (komsys-pc-bern)
10. Element Management Station T2 (kreechta.fhg.fa)
11. Naming Service (fhg.fhg.fa)
12. Service Registry (fhg.fhg.fa)
13. Service Repository (www.ucl.fa)
14. Network ASP (fhg.fhg.fa)
15. Extended SIP proxy (tik.tik.fa)
16. SIP Server and SIP Components, namely registrar, location, and feature servers (tik.tik.fa)

4.4.4 Web Service Distribution

In the Web Service Distribution Scenario, Web (HTTP) traffic is distributed and redirected within the network among several distributed servers in order to provide reliability, performance and scalability for web services.[3]

The full and exhaustive documentation of the Web Service Distribution Scenario is given in [2].

The application scenario can be run standalone or in a network. The scenario is subdivided into following independent demos:

- **Demo 1:** shows a simple redirection of TCP data. All packets, which are sent by a specified client and are addressed to a specified server, are re-routed to another computer.
- **Demo 2:** Additionally to the functionality of Demo1 the throughput is monitored and plotted. There is however no possibility to change the configuration of the module using this user interface.
- **Demo 3:** This demo is an extension of Demo2 using several client-server pairs. Additionally a simple graphical configuration option is offered.
- **Demo 4:** This demo is a simple load distribution example. Packets, addressed to a specified server are re-routed to different computers dependent on the current utilization condition of these computers.
- **Demo 5:** Adds a HTTP Parser to Demo4. The parser examines all headers and collects the necessary data for sessions. The results of the parser are not used in this demo.
- **Demo 6:** This demo adds support for sessions using the parser introduced in Demo5

[3] The term „Web Service” is often also used to refer to approaches for describing, finding and invoking objects and their services with web-based languages and protocols, e.g. Microsoft’s .NET. We use the term “Web Service” to refer to an application service which is offered to an end-user, as it has been invented in [1].

Architecture/Setup

The generic network topology of our demo is shown in Figure 4-11.

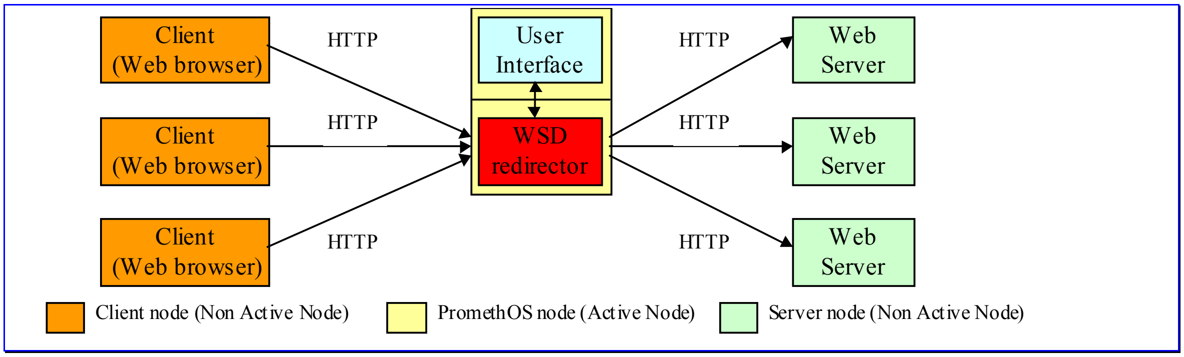


Figure 4-11: Network topology used by the demos

The PromethOS node is a component of a network, which in addition contains several web servers and web clients. Web clients may be ordinary web browser, or traffic generators, which are used in this demo for the easier creation of HTTP traffic. This is shown in Figure 4-11

Mapping onto FAIN Testbed

This application scenario is mapped as follows to the FAIN testbed:

- Web servers are installed at kreechta.fhg.fa, sagfs.sag.fa, sagan.sag.fa and ems.tik.fa.
- A FAIN Active Node is installed at ems.tik.fa.
- Client nodes must be installed at appropriate nodes depending on the location the demo is run. A possible mapping is shown in Figure 4-12.

<u>Possible set-up for demonstration taking place at FOKUS :</u>

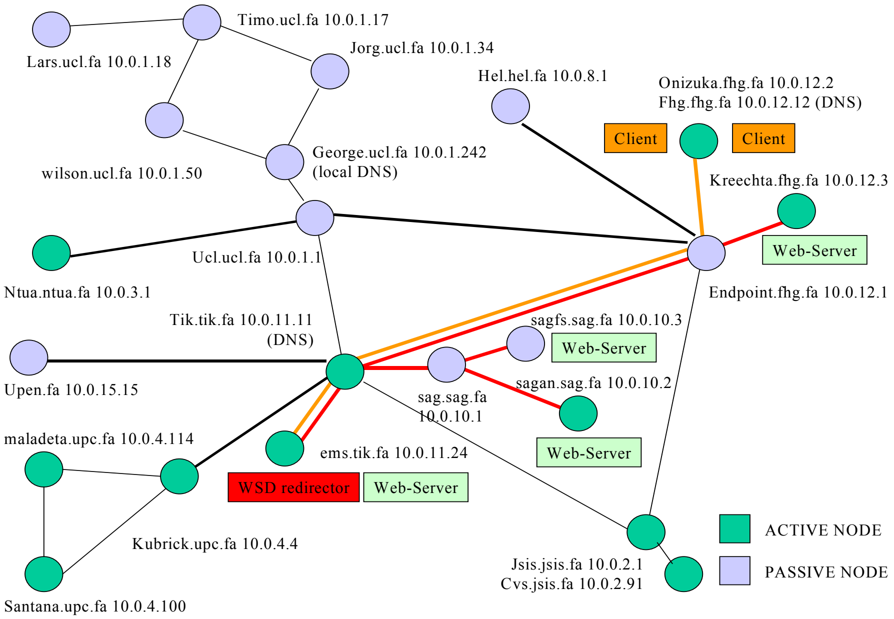


Figure 4-12: Network set-up for demonstration at FHG

Demo 1

This demo shows a simple redirector for TCP data. All packets, which are sent by a specified client and addressed to a specified server (IP-address and port), are re-routed to another computer. This demo is to a large extent outdated. It serves only to show the operability of the concept.

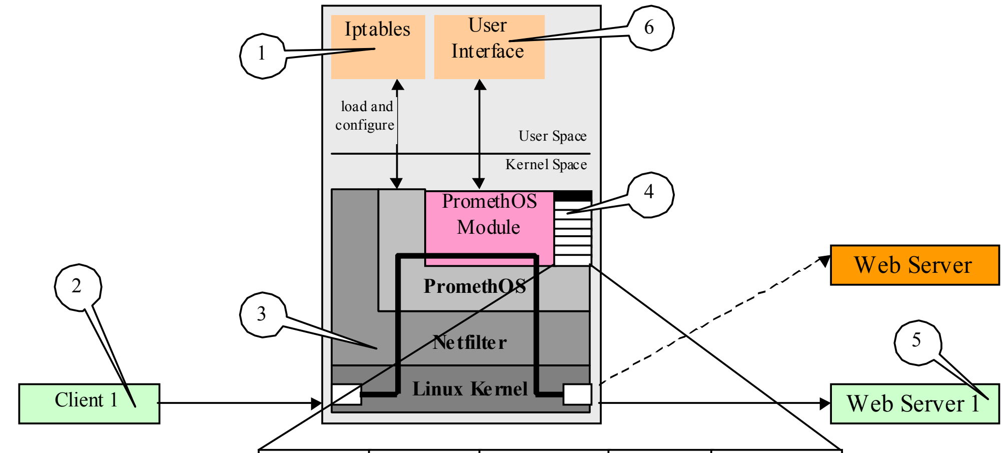
This demo allows to exactly specify one pair of client and server address which can be re-routed to another computer. Iptables is used to load the module and insert the necessary configuration data. The configured data can't be changed at run-time. The demo uses only one client and one web server.

Note: The same result could have been achieved using only Netfilter.

Demo 2

This demo is an extension of Demo1. It adds a user interface which monitors the throughput of the PromethOS module and shows it graphically. There is however no possibility to change the configuration of the module by using this user interface. The configuration takes place again using Iptables. The demo uses only one client and one web server.

Demo2 consists of 2 parts. On the one hand a PromethOS module is used which is responsible for the redirection of packets. On the other hand a user space program is used, which queries and plots the current throughput. For monitoring the PromethOS module a file in the /proc file system is used (/proc/promethos/net/management). The structure of Demo2 is shown in Figure 4-13.



Sender-IP	Receiver-IP	Receiver-Port	Forward-IP	Forward-Port
1.2.3.1	1.2.3.2	81	1.2.3.3	80

1: Iptables is used to load and configure the PromethOS module

2: A client sends requests to „Web Server“

3: Netfilter forwards the traffic to the PromethOS module

4: The PromethOS module redirects the traffic to it's final destination „Web Server 1“

5: „Web Server 1“ processes the requests and returns the results

6: The user interface is used to monitor the PromethOS module using the file /proc/promethos/net/management.

Figure 4-13: Structure of Demo2

Demo 3

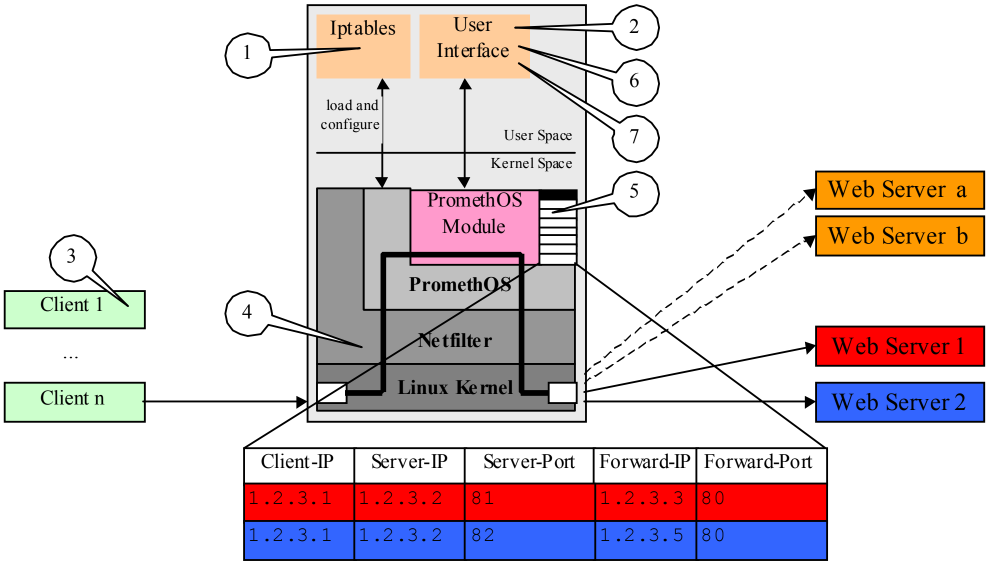
This demo is an extension of Demo2 using several client-server pairs. Additionally a simple graphical configuration option is offered. The user interface can be used to change the current configuration at runtime. This demo can use more than one client and more than one web server but there is no load balancing.

Demo3 consists of 2 parts. A kernel resident PromethOS module is responsible for the actual forwarding of the packets. This module is loaded using Iptables as done with the two preceding demos. A user space program is responsible for configuring the module and for monitoring the throughput. 3 files in the /proc file system are used to change the configuration data at runtime and to query the throughput data. /proc/promethos/net/management is used to send the configuration data to the PromethOS module. The file /proc/promethos_demo3/tables is used to query the current throughput data. The file /proc/slabinfo is used to query and display the current size of the contract tables.

The PromethOS module uses a forwarding table which contains the following information:

- A triple containing client-IP, server-IP, server-port
- A pair containing final destination-IP and final destination-port

Each of the table entries must contain a unique triple client-IP, server-IP and server-port. If there are multiple entries containing the same triple only the first entry will be used. All further entries are ignored. Each of these triples can be assigned to one and only one final destination (see Figure 4-14 for an example).



Client-IP	Server-IP	Server-Port	Forward-IP	Forward-Port
1.2.3.1	1.2.3.2	81	1.2.3.3	80
1.2.3.1	1.2.3.2	82	1.2.3.5	80

1: Iptables is used to load the PromethOS module

2: The user interface is used to configure the PromethOS module

3: One or more clients send requests to „Web Server a“ and „Web Server b“.

4: Netfilter forwards the traffic to the PromethOS module

5: The PromethOS module redirects the traffic to it’s final destination. Requests directed towards „Web Server a“ are redirected to „Web Server 1“ and requests for „Web Server b“ are redirected to „Web Server 2“.

6: The user interface can be used to change the configuration of the PromethOS module

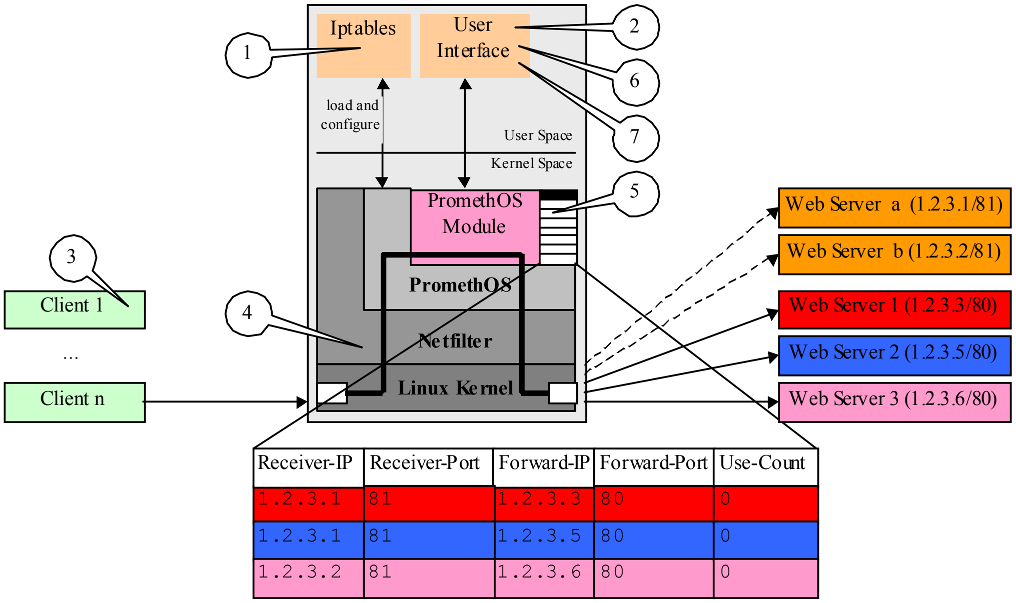
7: The user interface monitors the module using the files /proc/promethos/net/management, /proc/promethos_demo3/tables and /proc/slabinfo.

Figure 4-14: Structure of Demo3

Demo 4

This demo is a simple load distribution example. Packets, addressed to a specified server are re-routed to different computers dependent on the current utilization condition of these computers. The utilization of a computer is defined by the number of open TCP connections to this computer. The only connections considered are the connections running through the PromethOS module.

For each receiver of a request a number of alternate receivers can be specified. The load sent to one of the receivers is distributed over all the specified alternate receivers (see Figure 4-15 and Example 1 below).



Receiver-IP	Receiver-Port	Forward-IP	Forward-Port	Use-Count
1.2.3.1	81	1.2.3.3	80	0
1.2.3.1	81	1.2.3.5	80	0
1.2.3.2	81	1.2.3.6	80	0

1: Iptables is used to load the PromethOS module
2: The user interface is used to configure the PromethOS module
3: One or more clients send requests to „Web Server a“ and „Web Server b“.
4: Netfilter forwards the traffic to the PromethOS module
5: The PromethOS module redirects the traffic to it's final destination. Requests directed towards „Web Server a“ are redirected to „Web Server 1“ or „Web Server 2“ and requests for „Web Server b“ are redirected to „Web Server 3“.
6: The user interface can be used to change the configuration of the PromethOS module
7: The user interface monitors the module using the files /proc/promethos/net/management, /proc/promethos_demo4/tables and /proc/slabinfo.

Figure 4-15: Structure of Demo4

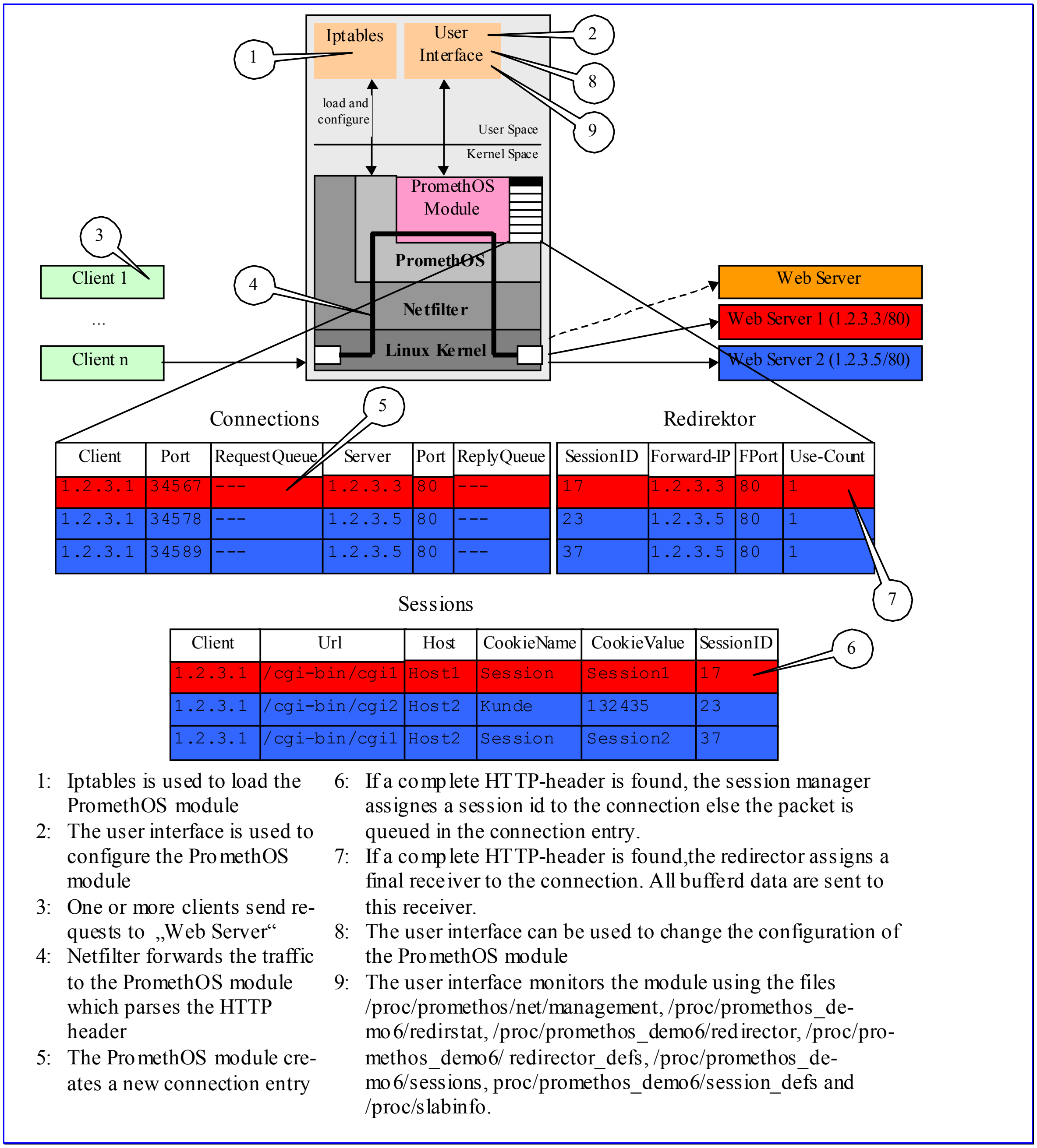
Demo 5

Demo5 is derived from Demo4. Demo5 adds a HTTP Parser to Demo4. The parser is used internally to examine all HTTP headers and to collect the data contained in the header. The collected information is not used in this demo however. A goal of this demo it to determine which load such a parser represents. The demo works as described for Demo4.

Demo 6

This demo shows likewise a simple load distribution. Packets, addressed to a specified server are re-routed to different computers dependent on the current utilization condition of these computers. The utilization of a computer is defined by the number of open TCP connections to this computer. The only connections considered are the connections running through the PromethOS module. Additionally to the two demos Demo4 and Demo5 this demo takes sessions into consideration i.e. all packets belonging to the same session are forwarded to the same computer independent of its current utilization.

In order to determine the packets belonging to a session, the HTTP headers of all requests and replies are examined. In order to guarantee that headers, which span several packets, can be processed correctly too, the packets are redirected to a local socket in the kernel. This socket receives the data, forwards them to the parser and, if necessary, buffers the data, if still no complete header is present (see Figure 4-16.).



Connections

Client	Port	RequestQueue	Server	Port	ReplyQueue
1.2.3.1	34567	---	1.2.3.3	80	---
1.2.3.1	34578	---	1.2.3.5	80	---
1.2.3.1	34589	---	1.2.3.5	80	---

Redirektor

SessionID	Forward-IP	FPort	Use-Count
17	1.2.3.3	80	1
23	1.2.3.5	80	1
37	1.2.3.5	80	1

Sessions

Client	Url	Host	CookieName	CookieValue	SessionID
1.2.3.1	/cgi-bin/cgi1	Host1	Session	Session1	17
1.2.3.1	/cgi-bin/cgi2	Host2	Kunde	132435	23
1.2.3.1	/cgi-bin/cgi1	Host2	Session	Session2	37

1: Iptables is used to load the PromethOS module
2: The user interface is used to configure the PromethOS module
3: One or more clients send requests to „Web Server“
4: Netfilter forwards the traffic to the PromethOS module which parses the HTTP header
5: The PromethOS module creates a new connection entry
6: If a complete HTTP-header is found, the session manager assignes a session id to the connection else the packet is queued in the connection entry.
7: If a complete HTTP-header is found,the redirector assigns a final receiver to the connection. All bufferd data are sent to this receiver.
8: The user interface can be used to change the configuration of the PromethOS module
9: The user interface monitors the module using the files /proc/promethos/net/management, /proc/promethos_demo6/redirstat, /proc/promethos_demo6/redirector, /proc/promethos_demo6/ redirector_defs, /proc/promethos_demo6/sessions, proc/promethos_demo6/session_defs and /proc/slabinfo.

Figure 4-16: Structure of Demo6

4.4.5 Video on Demand Scenario

PromethOS is a Linux kernel-space NodeOS for active nodes. It is managed from Execution Environments (EE) in user space. Since the Virtual Environment Manager (VEM), which is an EE in user space, is also concerned with node management issues, an integration of both, VEM and PromethOS, becomes indispensable for the interoperability of different EE types. The video on demand scenario demonstrates this integration of VEM and PromethOS. It shows how the VEM management interface of PromethOS' user space library has been enhanced in order to control PromethOS.

The ANNs have been designed to support multiple VEs, EEs and EE instances. The scenario demonstrates therefore also how PromethOS is supporting multiple VEs and how it is able to differentiate among the customers by using those VEs. For the scenario, one EE per VE is instantiated in kernel space. The EE runs a Wave Video plugin, which scales its functionality according to the different requirements of the customer.

Architecture/Setup

The source for the video stream is located at ems.tik.fa and flows via tik.tik.fa. The video receiver is installed at onizuka.fhg.fa. Between the source and the active node, a high-bandwidth link provides sufficient bandwidth. The link models a high-bandwidth backbone. The first ANN is statically configured.

The active network node and the video receiver are connected by two links with different bandwidth. The capacities of the links reflect different Service Level Agreements. The active network node is supposed to adapt the high bandwidth requiring input stream according to the pre-set output capacities of the output streams.

At boot time, the ANN is running the VEM and the management components of PromethOS. As a customer request arrives at the ANN, the VEM orders the deployment of a VE, i.e. it assigns resources to the customer, and it orders the deployment of an EE where the Wave Video scaling plugin is installed.

The request to initiate the service deployment is implemented by the FAIN-WP4 service specification. The service specification is parsed and resolved by the Service Creation Engine. The code required for the Wave Video plugin is fetched from code server and deployed on the ANN.

As a second request form a different customer arrives, the ANN creates a second instance with the same configuration but with different resources assigned to the VE. No additional code fetching is required anymore, since the code is available from the nodes local cache.

The creation process is fully implemented and controlled by the VEM. As the process completes, the video source gets a signal to begin with the transmission of the video flows.

Mapping onto FAIN Testbed

As depicted in Figure 4-17 the video source is located at ems.tik.fa. The video flow is transmitted via tik.tik.fa to onizuka.fhg.fa and from there to the laptop. The first ANN, i.e. tik.tik.fa, is statically configured. The main ANN, i.e. the dynamic ANN, is located at FHG's onizuka.fhg.fa. The video sink is a 24 bit capable Linux box. The results of the demonstration are shown by the setup process via VEM at onizuka.fhg.fa. The video flow will constantly be repeated.

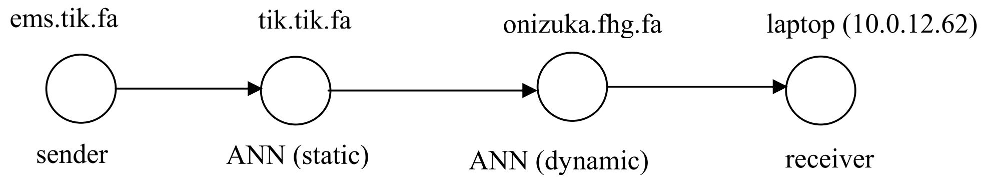


Figure 4-17: Network Topology for Video on Demand Scenario.

4.4.6 FAIN Mobility Demonstrator

The FAIN Mobility Demonstrator shows load balancing in a mobile environment, in particular in WLAN networks. Additionally to state-of-art concepts employed in WLAN networks on regular basis (e.g. hand-over) FAIN specific load balancing concepts are implemented that improve the usability of WLAN networks.

The following concepts are demonstrated by this demo:

- Use of PromethOS kernel modules for monitoring, routing and bridging in WLAN networks.
- Implementation of a load balancing mechanism for WLAN adopted from cellular networks.
    - On camp-on (when connecting to the system, a client is rejected and redirected to another AP if the tried one is overloaded.)
    - Pre-emptive load distribution (is load on a specific AP is getting to high, selected terminals are redirected to other APs which have capacity available.)
- Interaction of PromethOS modules by interaction with user space applications. This includes data exchange, configuration and installation of modules by user space applications.
- Application dependent load balancing mechanism.

Architecture/Setup

The generic network topology of our demo is shown in Figure 4-18. The active node used in the scenarios is a PromethOS Node. The node is responsible for the redirection of WLAN traffic to different Access Points. In addition it runs the user interface for configuring and monitoring the PromethOS modules. As Non-Active nodes the scenario distinguishes sender and receiver nodes. The receiver nodes run common web browsers and video streaming clients or some other web traffic generator. They are mobile clients like notebooks or PDAs. The sender nodes run a normal web server and provide video streaming. The scenario uses two nodes of this type. They play the role of content servers (e.g. static web pages or Web TV).

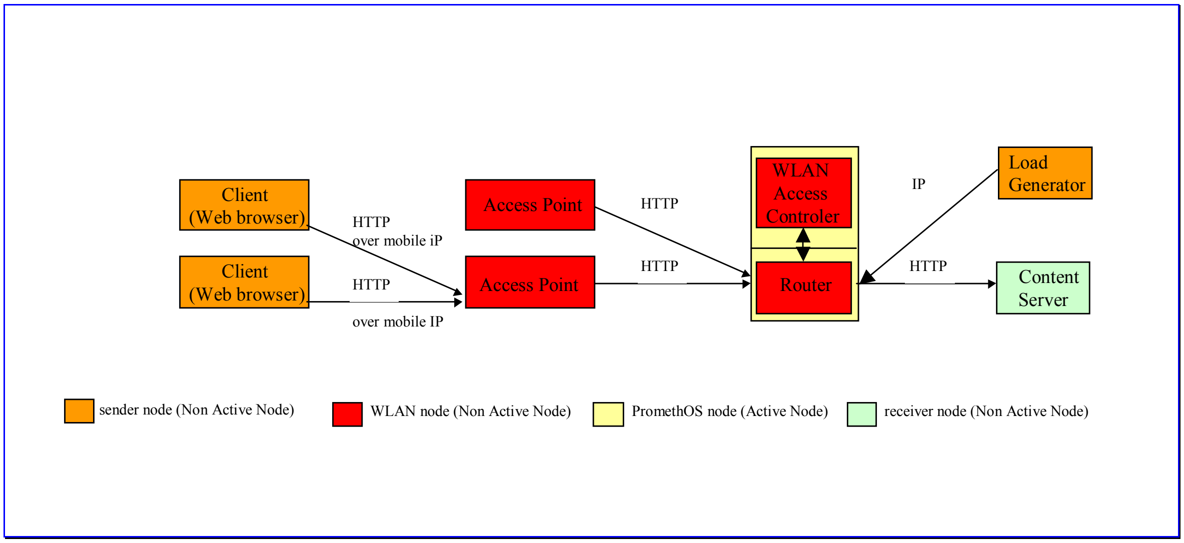


Figure 4-18: Network topology used by the demos

Mapping onto FAIN Testbed

The Web servers used in the scenario are located at Munich and at the ETH (ems.tik.fa). In order to use the preinstalled demo using the FAIN testbed, a FAIN Active Node must be installed at kreechta.fhg.fa. The demos can be shown independent of each other, but it is suggested to show them in the described order as some demos are based on earlier ones.

Demo 1

This demo shows a load-dependent camp-on. The initial connection to an Access Point is made load dependent. The mobile client scans for best Access points, sends a probe to the one selected, which is unfortunately overloaded. This is transmitted to the WLAN control unit and depending on the load information the client is rejected and redirected to another access point. By this procedure a load balancing is affected.

The WLAN Access Controller initializes the PromethOS modules as required by the used functionality of the Access Controller. It inserts the necessary configuration data. The configured data can be changed at run-time. The demo uses only two clients and one FAIN active node with WLAN Access Control Unit. The load on the Access Point 1 may be simulated by setting the load parameters in the WLAN Access Controller directly. In this case a content server is necessary. A second more elaborated version of the demo uses real data transport over Access Point 1 to trigger the redirection of the connection attempt of the client. In this version, a content server is required to generate the load. .

For each Access Point the current load is shown separately in a User Interface. The User Interface is used to manipulate the internal load parameters in the WLAN controller, in order to simulate high load in a simple fashion without having to change to actual load in the network. If a load generator is used generating network load then the user interface does only show the current load on the different APs. The load itself is adjusted by a user interface connected with the load generator that allows to regulate the load required.

Demo 2

This demo starts after the clients have connected to appropriate access points. It shows pre-emptive load distribution: if load on a specific access point is getting to high, selected mobile clients are redirected to other access points which have capacity available. The controlling of the access points' load and the redirection is done by the WLAN Control unit. The demo uses only one client, one FAIN active Node with WLAN Access Control Unit and one content server. The content server provides video streaming. Requesting video streaming generates the load at one access point.

The structure of Demo2 is shown in Figure 4-19.

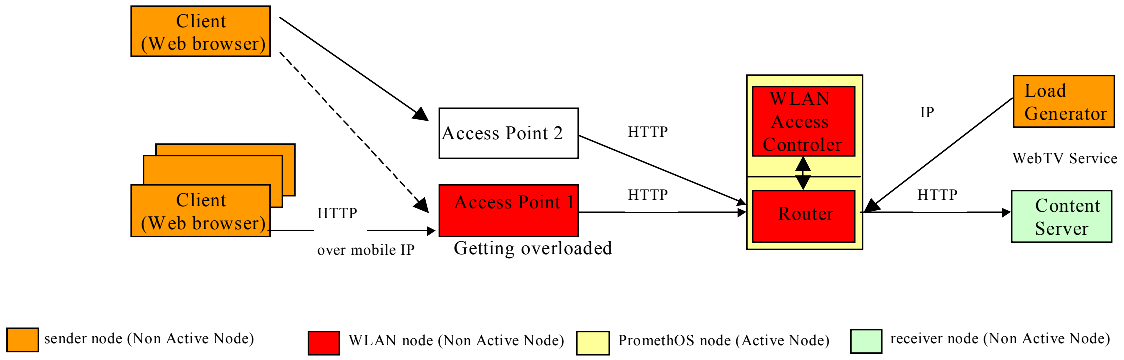


Figure 4-19: Structure of Demo2

Increase of the load on access point 1 is generated by the load generator. This is done either by an automatic script controlling the load generator or manually by controlling the load generator via its user interface. The user interface connected with the WLAN Access Controller is used to change criteria used for decision making. Application specific criteria, e.g. thresholds, and priorities may be set by the administrator of the WLAN access controller.

5 EVALUATION OF THE ARCHITECTURE AND IMPLEMENTATION

5.1 Evaluation Methodology

The overall goal of the evaluation is to give insights on the 'level of programmability' of FAIN. The programmability is expressed by properties. The evaluation consists in identifying if these properties are owned by FAIN and to what expense and extent. In order to avoid exhaustive measurements for the evaluation and in order to get a fine-grained evaluation, the properties are broken down to sets of features and performance metrics.

The features and performance metrics apply to distinct operational planes and at different level/location layers. Figure 5-1 sketches the relation between ownership and cost of a property's feature, depending on the level/location and operational plane where it occurs.

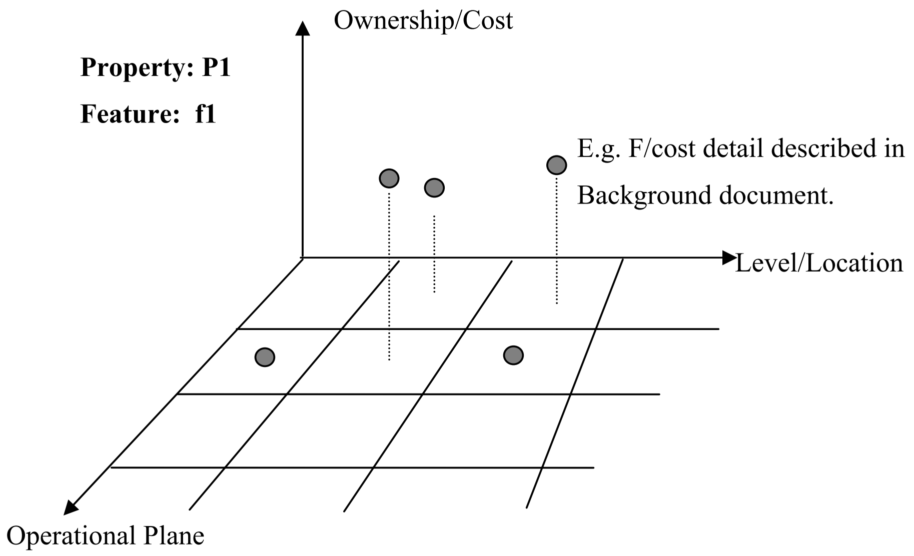


Figure 5-1: Evaluation Model for Features and Properties.

Figure 5-2 represents a reference model that describes the relation between the operational planes and the level/location layers. Every feature will be evaluated against all locations of the given reference model. For those locations we distinguish between Management, Control, and Transport planes and between Network, Node, and Technical Layers.

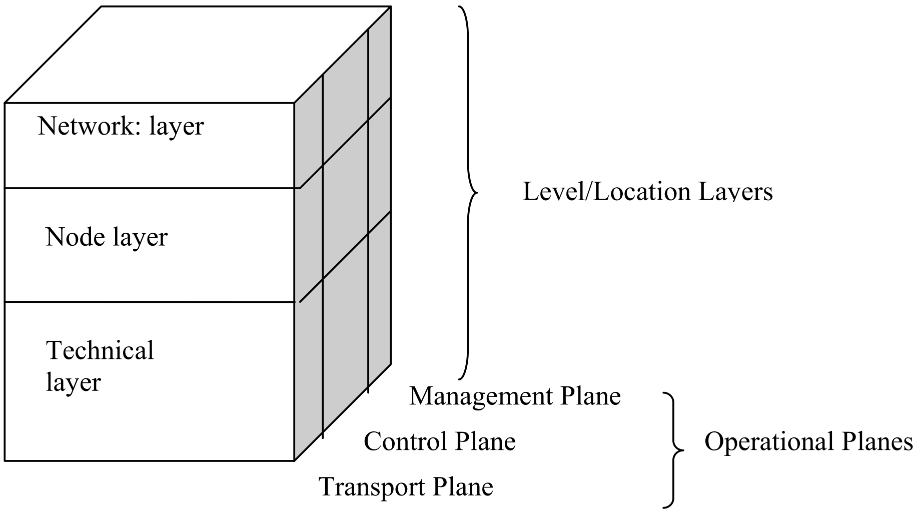


Figure 5-2: Reference Model for Operational Planes and Level/Location Layers.

Note that the evaluation is not going to compare FAIN networks with other AN networks anymore, as has been suggested in the D2 document.

5.1.1 Templates and Representation

For the representation of the evaluation (as suggested in Figure 5-2) there is need for meaningful representation. The objective is to have a uniform and expressive representation of the evaluation results.

The templates, one for each property taken into consideration, provide two abstraction levels. The first level allows getting a quick overview of the evaluation results, whereas the second is going into more detail. The expressiveness of the first level is achieved by deploying a tagging syntax that already includes rating information.

The tags that are available for filling in the table are:

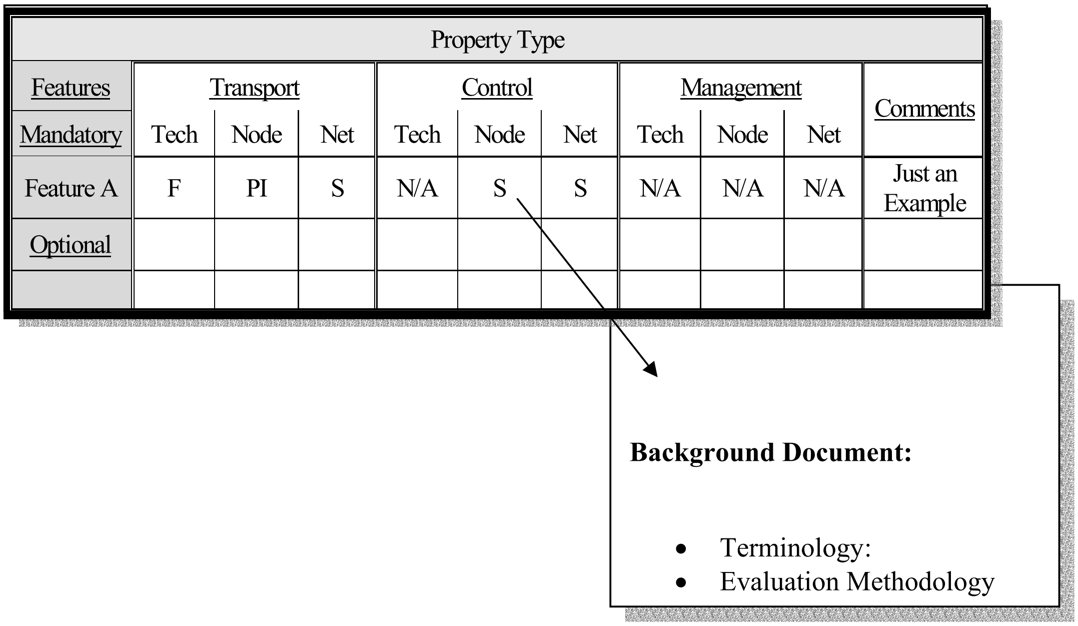
- F : Fully Implemented (specified and fully implemented)
- PI: Partially Implemented (fully specified but partially implemented)
- S: Specified (only specified)
- N/S: Not Specified
- N/A: Not Applicable

In Figure 5-3 the template form is shown: the first level of abstraction is a table containing for each property several features and evaluating for each of them their level of “goodness” in FAIN. For each entry in the table exists a background document that holds the details of the rating. This background document represents the second level of abstraction. It contains the following sections:

Terminology/Context: clarifies the meaning of a feature.

Evaluation Methodology: describes briefly how the evaluation has been carried out. This may contain a short description of performed tests, measurement methods etc.

Evaluation Results: Holds a short description of the results. It’s the most important part of the document.



Property Type										
Features	Transport			Control			Management			Comments
Mandatory	Tech	Node	Net	Tech	Node	Net	Tech	Node	Net	
Feature A	F	PI	S	N/A	S	S	N/A	N/A	N/A	Just an Example
Optional										

Background Document:

- Terminology:
- Evaluation Methodology

Figure 5-3: Two Level Evaluation Template for Property Types.

5.1.2 Classification of the FAIN Components

In Table 5-1we propose a classification of the FAIN components according to the scheme proposed in Figure 5-2. We will refer to this during the evaluation process.

	TRANSPORT PLANE	**CONTROL PLANE**	**MANAGEMENT PLANE**
TECHNICAL LEVEL (router, node OS)	PromethOS GR2000 Java	PromethOS GR2000 Java	PromethOS Java CORBA XMLService Descriptors XML Mgmt policies
NODE LEVEL	FAIN java EE PromethOS EE Transcoder/Duplicator Video scaling	FAIN java EE PromethOS EE RCF Security FW	FAIN java EE SNAP EE EMS Node ASP VEM
NETWORK LEVEL	Web TV Service Video Scaling Web Cache	ANEP+FAIN opt. DiffServ Service SIP Web Cache RTP	NMS Net ASP SNAP activator

Table 5-1: Classification of the FAIN components

5.2 EVALUATION RESULTS

5.2.1 Flexibility

Level 1 Representation

Flexibility is a quite generic property. By this we mean the ability of a system to change dynamically its behavior, adapt to new requirements, cope with increases in information volumes and functionality, and reuse or synthesize its existing services.

Table 5-2: Table for Flexibility Property Type

Property: Flexibility										
Features	*Transport*			*Control*			*Management*			*Comments*
	Tech	*Node*	*Net*	*Tech*	*Node*	*Net*	*Tech*	*Node*	*Net*	
Composability	F	F	NA	F	F	NA	F	F	F	
Extensibility	F	F	NA	F	F	S	F	F	F	
Scalability	F	NS	NA	F	NS	NA	F	S	S	
Code Loading	F	F	NA	F	F	NA	F	F	S	
Virtualization	NS	NA	NA	NS	F	NA	F	F	F	

Background Document (level 2)

Composability:

Terminology: Composability allows the system to reuse and recombine its functional components into forming new services and functionality.

Evaluation Methodology: we propose to evaluate the "flexibility" of FAIN referring to each of its components as grouped in Table 2. For each feature we will look in the subset of components related to each column of the Flexibility table and see whether at least one of them satisfies it (or was designed to). In the following section we will explain and motivate all our statements. If we say that the component X satisfies property Y, we will write here why and how.

Evaluation Results:

Transport plane:

- **Technical level:** The feature is fully implemented at this level. Our Operating System PromethOS satisfies it via its plug-ins [17].
- **Node level:** The PromethOS/java EEs are composable.
- **Network level:** The N/A, i.e. Not Applicability, is due to the fact that in this group are Services and Applications, and whether those services are composable or not doesn't help evaluating the level of composability of the FAIN architecture.

Control plane:

- **Technical level:** cf. Transport plane.

- **Node level:** The PromethOS/java EEs are composable; all components that run on a VEM are composable.
- **Network level:** The N/A, i.e. Not Applicability, is due to the fact that in this group are Services and Applications, and whether those services are composable or not doesn't help evaluating the level of composability of the FAIN architecture.

Management plane:

- **Technical level:** cf. Transport plane. Moreover CORBA and XML add to the system a further level of composability.
- **Node level:** at this level Composability is satisfied by the EMS. [18]
- **Network level:** at this level Composability is satisfied by the NMS [19]that, like the EMS has a composable structure.

Comments: None.

Extensibility:

Terminology: Extensibility allows the system to evolve as new requirements and services are needed while these can be introduced and incorporated in the existing system in a seamless way.

Evaluation Methodology: Same as for Composability.

Evaluation Results:

Transport plane:

- **Technical level:** The feature is fully implemented at this level. Our Operating System PromethOS satisfies it via its plug-ins [17]
- **Node level:** The PromethOS/java EEs are extensible; all components that run on a VEM are extensible [VEM]. The functions can be extended by inserting new rules.
- **Network level:** The not Applicability is due to the fact that in this group are Services and Applications, and whether those services are composable or not doesn't help evaluating the level of composability of the FAIN architecture.

Control plane:

- **Technical level:** cf. Transport plane.
- **Node level:** The PromethOS/java EEs are extensible; all components that run on a VEM are extensible.
- **Network level:** The not Applicability is due to the fact that in this group are Services and Applications, and whether those services are composable or not doesn't help evaluating the level of composability of the FAIN architecture

Management plane:

- **Technical level:** cf. Transport plane. Moreover CORBA and XML add a further level of extensibility to the whole system.
- **Node level:** extensibility is satisfied by the EMS by extending PDPs or adding new ones and PEPs (a deeper explanation is given in [21]). Regarding the Node ASP a deeper explanation of its extensibility is given in [22].
- **Network level:** the NMS is extensible; this is achieved by introducing new policies and new functional domains (The NMS use the same extensibility mechanism as the EMS). The ASP is also extensible (although this is mainly specified and not fully implemented): you may extend services and how you deploy them.

Virtualization:

Terminology: Virtualization allows for the partitioning of network resources among different user communities. This results in supporting more liberal business models and customized usage of resources.

Evaluation Methodology: Same as for Composability.

Evaluation Results:

Transport plane:

- **Technical level:** Virtualization for PromethOS was not specified. Java provides it.
- **Node level:** At this level the feature "Virtualization" is not applicable.
- **Network level:** This feature is not applicable at this level (composed of Services and applications).

Control plane:

- **Technical level:** see Transport plane.
- **Node level:** At this level virtualization is provided (and fully implemented) by the RCF. Its task is in fact to virtualize the resource.
- **Network level:** This feature is not applicable at this level (composed of Services and applications).

Management plane:

- **Technical level:** see Transport plane. Moreover CORBA and XML offer to the system virtualization capabilities.
- **Node level:** EMS achieves virtualization by creating new management instances.
- **Network level:** the NMS achieves virtualization by creating new management instances.

Scalability:

Terminology: Scalability refers to the network architecture design and the distribution of the network functionality in such a way that the network can account for increasing volumes of user requests.

Evaluation Methodology: Same as for Composability.

Evaluation Results:

Transport plane:

- **Technical level:** here we refer to the same scalability properties that Linux and Java have.
- **Node level:** Scalability was not taken into consideration when designing this part of the architecture.
- **Network level:** The N/A, i.e. Not Applicability, is due to the fact that this group consists of Services and Applications, and that whether those services are composable or not doesn't help evaluating the level of composability of the FAIN architecture.

Control plane:

- **Technical level:** same as for Transport plane.
- **Node level:** As far as RCF is concerned, resources can be installed up to a certain limit, we ignore what that limit is.

- **Network level:** The N/A, i.e. Not Applicability, is due to the fact that this group consists of Services and Applications, and that whether those services are scalable or not doesn't help evaluating the level of composability of the FAIN architecture, e.g. DiffServ scales not well**.**

Management plane:

- **Technical level:** here we refer to the same scalability properties that Linux, Java, CORBA and XML have.
- **Node level:** scalability is satisfied by the node ASP as it was designed as fully decentralized. The EMS satisfies scalability via its extensibility and modularity.
- **Network level:** the NMS is scalable [19]. The Network ASP is designed as decentralized and is therefore, scalable.

Code Loading:

Terminology: Code Loading

Evaluation Methodology: Same as for Composability.

Evaluation Results:

Transport plane:

- **Technical level:** Code can be loaded on PromethOS via either the PromethOS Control Daemon or the ASP.
- **Node level:** This property is fully implemented in the EEs.
- **Network level:** This particular property doesn't make sense at this level: here are the services that are usually loaded!

Control plane:

- **Technical level:** see Transport plane.
- **Node level:** This property is fully implemented in the EEs.
- **Network level:** This particular property doesn't make sense at this level: here are the services that are usually loaded!

Management plane:

- **Technical level:** same as for Transport plane.
- **Node level:** ASP is used for it, so for it this feature would be NA. In the case of EMS anyway the possibility to download code via the ASP has been specified. As far as the WP3 code is concerned, this feature has been fully implemented.
- **Network level:** In the case of NMS code loading is a little bit like extensibility.

5.2.2 Security

Level 1 Representation

Evaluation of FAIN security architecture covers issues like

- what security features are provided in FAIN ANN?
- how and where are they provided?
- What "level" of security do these features provide?

- how does FAIN security architecture compare against other existing approaches?
- how does FAIN security architecture perform (in an experimental test-bed environment)?

The evaluation is qualitative and quantitative. Comparing different security architectures directly, i.e. in a quantitative manner, in order to say the least is difficult. The same applies even for quantitative evaluation of a single security architecture, since it is hard to define sensible criteria for the "measurement" of security in a system. Thus, the FAIN security architecture is mostly evaluated in a qualitative way, although we strive to give some measurement results mainly on performance overhead aspects.

Qualitative evaluation covers the first four questions posed. It is based on the analysis of FAIN security architecture with regards to a set of security features/requirements restricted to high-priority security requirements as defined in D2, ([10] on page 70). This should give an overall sense of what "level" of security is provided within FAIN.

The table then gives the reader a comprehensive view of the level of security provided by FAIN (or other AN) based on four sets of information for every security feature:

1. presence of the feature in the transport, control, and management planes
2. operation of the feature, i.e. whether it operates locally on the active node or it demonstrates network wide behavior
3. which technology is the feature based on
4. nature of the feature, describing the level of security this feature provides by specifying what it protects against and how does it implement the protections

As mentioned, it is hard if not impossible to define security evaluation criteria, which can be measured in an experimental set-up. However, even though we can not measure the level of security, it may be interesting to measure the sheer performance aspects of security, i.e. he performance overhead incurred when security mechanisms are activated.

Table 5-3: Table for Security Property Type

Property: Security										
Features	*Transport*			*Control*			*Management*			*Comments*
	Tech	*Node*	*Net*	*Tech*	*Node*	*Net*	*Tech*	*Node*	*Net*	
Authentication	PI	F	F	PI	F	F	PI	F	F	
Authorization	PI	F	F	PI	F	F	PI	F	F	
Enforcement	PI	F	F	PI	F	F	F	F	F	
Integrity	PI	F	F	PI	F	F	F	F	F	
Audit	S	S	S	S	S	S	S	S	S	
Verification	PI	PI	PI	PI	PI	PI	PI	PI	PI	

Background Document (level 2)

Authentication:

Terminology: Authentication allows the system to securely verify the identity of a principal.

Evaluation Methodology: This feature has been evaluated in two respects. Firstly, we estimate to what extent this feature has been provided within the FAIN active node (prototype). This is depicted in Table-2. Secondly, based on the experimental measurements with the FAIN security architecture prototype, we have tried to estimate the performance overhead imposed by this feature. The later part can be found in the performance section on page 54.

Evaluation Results:

Transport plane, Control plane, and Management plane:

- **Technical level:** This feature has been partially implemented; in transport and control plane of PromethOS and GR2000 authentication is not supported.
- **Node level:** Strong authentication has been provided based on digital signatures, symmetric cryptography, and SSL protocol. Digital signatures provide end-to-end authentication for active packets which can be authenticated on every node that the packets traverse. Symmetric cryptography provides per hop authentication between peer nodes exchanging packets and can be used for inter node communication. SSL based authentication is used with CORBA to authenticate management sessions to the node.
- **Network level:** Strong authentication is based on the supporting PKI infrastructure and the protocol for credentials exchange between neighbor nodes and credentials cache on the nodes.

Authorization:

Terminology: Authorization decides whether requested action by a principal shall be allowed or denied.

Evaluation Methodology: We estimate to what extent this feature has been provided within the FAIN active node (prototype).

Evaluation Results:

Transport plane, Control plane, and Management plane:

- **Technical level:** This feature has been partially implemented: in transport and control plane of PromethOS and GR2000 authorization is not supported
- **Node level:** A dedicated, central SBB has been developed, which is responsible for making authorization decisions for every critical operation within the FAIN node. Authorization engine, example policy engine, set of credentials with authorization data and related keystores with private keys and example security policies were provided
- **Network level:** Authorization is based on the supporting infrastructure, such as Attribute Authorities, which are responsible for granting credentials to users. The management system ARC (Access Rights Check) Component has support for it. It validates an incoming request, a policy, against a particular schema. Each principal has associated one. Each schema contains what the principal can do. This schema is in fact an XML Schema.

Enforcement:

Terminology: Enforcement acts upon the authorization decision, i.e. it either allows or denies the execution of principal's request.

Evaluation Methodology: We estimate to what extent this feature has been provided within the FAIN active node (prototype).

Evaluation Results:

Transport plane, Control plane, and Management plane:

- **Technical level:** Each FAIN subsystem implements its own enforcement engine. This feature has been partially implemented for the security subsystem: in transport and control plane of PromethOS and GR2000 enforcement is not supported.
- **Node level:** Enforcement was integrated with the node management system which implementation of components include the enforcement engines and mechanism implemented with CORBA interceptors that pass the necessary information about accessing subject to the object being accessed. Access to every component interface can be controlled by accessing Security manager authorization interface. This interface invokes authorization engine and if necessary policy engine. At EMS the ARC component checks if the incoming request is valid and denies execution of the unauthorized requests
- **Network level:** Mechanisms have been specified and partially implemented, which control the network-wide use of resources by the user. As in the case of the EMS there is one ARC component that checks if the principal is authorized to do what it's described inside his request, the policy.

Integrity:

Terminology: Integrity enables the system to detect any modifications of the information in transit over the network by unauthorized adversaries.

Evaluation Methodology: This feature has been evaluated in two respects. Firstly, we estimate to what extent this feature has been provided within the FAIN active node (prototype). Secondly, based on the experimental measurements with the FAIN security architecture prototype, we have tried to estimate the performance overhead imposed by this feature. The later part can again be found in the performance section on page 54.

Evaluation Results:

Transport plane, Control plane, and Management plane:

- **Technical level:** This feature has been partially implemented: in transport and control plane of PromethOS and GR2000 integrity is not supported.
- **Node level:** Hop-by-hop integrity is provided based on a keyed hash function, when packets need to be modified at FAIN ANNs en route
- **Network level:** End-to-end integrity is either provided with digital signature (when packets are not modified en route) or can be incurred from per-hop protections, when packets are processed at ANNs.

Audit:

Terminology: Logging allows the system to keep a trail record of security relevant (and other) events within the system and enables later analysis and assessment of security critical events.

Evaluation Methodology: We estimate to what extent this feature has been provided within the FAIN active node (prototype).

Evaluation Results:

Transport plane, Control plane, and Management plane:

- **Technical level:** This feature has been specified.
- **Node level:** This feature has been specified.
- **Network level:** This feature has been specified.

Verification:

Terminology: Verification allows the system to dynamically assess the safety of the active code before executing it.

Evaluation Methodology: We estimate to what extent this feature has been provided within the FAIN active node (prototype). This is depicted in Table-2.

Evaluation Results:

Transport plane, Control plane, and Management plane:

- **Technical level:** This feature has been partially implemented:
- **Node level:** Code verification is based on the digital signature of the trusted third party, which performs the code safety assessment. Verification of the in-band code with the use of digital signature mechanism and program fingerprint for security critical data in program has been implemented in Active SNMP system.
- **Network level:** Code verification is based on support infrastructure, such as trusted code servers.

5.2.3 Interoperability

Level 1 Representation

The interoperability property is assessed (evaluated) by checking the interoperability between several system components of a FAIN node or network infrastructure. For instance, we envision to check interoperability between de-multiplexers, resource control software, security software and virtual environment software.

Table 5-4: Table for Interoperability Property Type

Property: Interoperability										
Features	*Transport*			*Control*			*Management*			*Comments*
	Tech	*Node*	*Net*	*Tech*	*Node*	*Net*	*Tech*	*Node*	*Net*	
Security Interop	NS	F	F	F	F	F	F	F	F	
VE Interop	NA	F	NA	NA	F	PI	NA	F	F	
EE Interop	NA	S	NA	NA	S	NA	NA	S	NA	
Mgmt interop	NA	NA	NA	NA	NA	NA	NA	NS	PI	

Background Document (level 2)

EE Interop:

Terminology: Allows to evaluate the interoperability of two execution environments, basically means for an EE to call services offered by another EE.

Evaluation Methodology: Collect means offered by an EE to call its services from another EE.

Evaluation Results: This also depends on the willingness (by design) to provides such functions. For specialized EEs (for example, management environment), this could be voluntarily limited to avoid problems. Therefore, different shades of results are possible.

Transport plane:

- **Technical level:** Not Applicable.
- **Node level:** SNAP EE to Java EE and Java EE to PromethOS EE interoperability (i.e. FAIN EEs interoperability) on the same Node may exist due to the fact that all the EEs use the ANEP encapsulation protocol.
- **Network level:** Not Applicable

Control plane:

- **Technical level:** Not Applicable
- **Node level:** SNAP EE to Java EE and Java EE to PromethOS EE interoperability (i.e. FAIN EEs interoperability) on the same Node may exist due to the fact that all the EEs use the ANEP encapsulation protocol.

  SNAP to Java interoperability on data path may also be extended to the control path
- **Network level:** Not Applicable

Management plane:

- **Technical level:** Not Applicable
- **Node level:** cf. transport plane. The EE interoperability is ensured by common node level management APIs.
- **Network level:** Not Applicable

Comments: Note that the results of this evaluation (as many others) is not for quantification (can be hardly done, and not necessarily useful), but to assess its use (i.e. the type of applications it is better suited for).

Security Interop:

Terminology: Allows to evaluate the interoperability of two security services, especially those running on different nodes.

Evaluation Methodology (Hint): Check the results of the processing of security measures carried by packets. For example, the result of authentication triggered by a packet on security service A should be the same as if the authentication was performed by security service B.

Evaluation Results:

Transport plane:

- **Technical level:** holds for Java only
- **Node level:** applies for all entries of table 2
- **Network level: --**

Control plane:

- **Technical level:** applies for Java and weakly for PromethOS and GR 2000
- **Node level:** should apply for all
- **Network level:** applies for the ANEP +FAIN opt and the DiffServ Services

Management plane:

- **Technical level:** holds for the XML Service Descriptors and the XML Management policies

- **Node level:** applies for all entries at this point except for EMS and ASP
- **Network level:** applies for the SNAP activator only

VE Interop:

Terminology: Relates the interoperability between two virtual environment (i.e. the possibility for components running on these VEs to interact), whether they run on the same node or on different nodes.

Evaluation Methodology (Hint): Collect means offered by FAIN VE interfaces to invocate services from another VE.

Evaluation Results (Note): This can give different shades of results ranging from "null" (impossible to call services from another VE) to a subset of services. But it seems hard to allow all services to be called from the outside, for confidentiality reasons. A major goal of VEs is to isolate stakeholders …

Transport plane:

- **Technical level:** Not Applicable
- **Node level:** only privileged VE to VE interoperation, privileged VE inherits CORBA interoperability, VEs interaction is possible based on the DeMUX.
  Same like EE properties. C.f. EE interoperability.
- **Network level:** Not Applicable

Control plane:

- **Technical level:** Not Applicable
- **Node level:** cf. EE interoperability
- **Network level:** CORBA is used for VE control messages

Management plane:

- **Technical level:** XML service descriptors are the base technology used in different components in FAIN. The description of a service may provide interoperability between components understanding and using the same description.
- **Node level:** ASP via the same service descriptions [24]. EMS is used to manage VEs on the node.

  Cf. EE interoperability, same as EE properties.
- **Network level:** Net-ASP for service deployment via XML service descriptors [25]. NMS is used to manage VEs across the network, Virtual Network ID is used to define VEs to interact

Management System Interoperability:

Terminology: interoperability between different management systems.

Evaluation Results:

Transport plane:

- **Technical level:** Not Applicable
- **Node level:** Not Applicable
- **Network level:** Not Applicable

Control plane:

- **Technical level:** Not Applicable

- **Node level:** Not Applicable
- **Network level:** Not Applicable

Management plane:

- **Technical level:** Not Applicable
- **Node level:** Not Specified.
- **Network level:** FAIN management systems in different domains communicate using RMI. A protocol describing the service agreement has been partially specified [23].

5.2.4 Openness

Level 1 Representation

Openness refers in a large sense to several criteria, typically the availability of application programming interfaces and facilities to use different specifications and implementations of same service functionality, e.g. format for management policies.

Table 5-5: Table for Openness Property Type

Property: Openness										
Features	*Transport*			*Control*			Management			*Comments*
	Tech	*Node*	*Net*	*Tech*	*Node*	*Net*	*Tech*	*Node*	*Net*	
Application Programming Interfaces (API)	FI	FI	FI	FI	FI	FI	FI	FI	FI	Applies for PromethOS and WP4
Open System Services (OSS)	NA	NS	NS	NA	NS	NS	NA	FI	FI	WP4

Background Document (level 2)

APIs:

Terminology: Relates the availability of APIs offered by the system. APIs can be defined at the application level (for example, a service provider could offer APIs to its clients), at the network level, and at the node level (VE, EE, system services interfaces).

Evaluation Methodology (Hint): Assess system design choices and implementations.

Evaluation Results:

Transport plane:

- **Technical level:** PromethOS is open source and can be downloaded at http://www.PromethOS.org. The APIs are all clearly defined and well-documented and open to modification.
- **Node level:** API are specified for the most of the components at this level
- **Network level:** API are specified for the most of the components at this level

Control plane:

- **Technical level:** cf. Transport plane.
- **Node level:** API are specified for the most of the components at this level
- **Network level:** API are specified for the most of the components at this level

Management plane:

- **Technical level:** cf. Transport plane
- **Node level:** API are specified for the most of the components at this level
- **Network level:** API are specified for the most of the components at this level

OSSs:

Terminology: Possibility to use different specifications and implementations of the same functionality.

Evaluation Methodology: Assess system design choices and implementations.

Evaluation Results:

Transport plane:

- **Technical level:** Not Applicable
- **Node level:** Not Specified
- **Network level:** Not Specified

Control plane:

- **Technical level:** Not Applicable
- **Node level:** Not Specified
- **Network level:** Not Specified

Management plane:

- **Technical level:** Not Applicable
- **Node level:** Policies are a kind of open interfaces given in different formats, e.g. COPS (in FAIN we make use also of different formats, the concept here expressed remains anyway the same).

  Both public and extensible interfaces apply for the ASP, e.g. XML service descriptor and IDL (both open standards). Chameleon, experiences with modifications and adaptations.
- **Network level:** cf. Node Level.

5.2.5 Portability

Level 1 Representation

Portability is a key system property for the implementation of active services: it determines if it is feasible to dynamically run the same program on several nodes or node service environments. We can distinguish three key feature of this property: the possibility to run an active code over different VEs within a node (called context switching), the possibility to run an active code over different EEs within the same node (run-time facility), and the possibility to run a same active code on different nodes (program migration).

Table 5-6: Table for Portability Property Type

Property: Portability										
Features	*Transport*			*Control*			*Management*			*Comments*
	Tech	*Node*	*Net*	*Tech*	*Node*	*Net*	*Tech*	*Node*	*Net*	
Run-time Facilities	NS	F	NS	NS	F	**S**	NS	F	PI	
Context independency	NA	NA	NA	NA	NA	NA	NA	F	F	

Background Document (level 2)

Run Time Facilities:

Terminology: Is code built for a specific run-time platform able to run on another run-time platform without any modification.

Evaluation Methodology: Assess system design and implementation choices.

Evaluation Results:

Transport plane:

- **Technical level:** PromethOS plugins run on the PromethOS platform only.
- **Node level:** every FAIN node (either type A or C) offers the same API to the code running on them. In this way the same code may run on the different node types without need for modification (in this case the portability is limited to the same EE).
- **Network level:** NS

Control plane:

- **Technical level:** PromethOS plugins run on the PromethOS platform only.
- **Node level:** every FAIN node (either type A or C) offers the same API to the code running on them. In this way the same code may run on the different node types without need for modification (in this case the portability is limited to the same EE).
- **Network level:** packets may be executed in other networks, the java code can be executed on other virtual machines on different platforms and the ANEP header is known.

Management plane:

- **Technical level:** PromethOS plugins run on the PromethOS platform only.
- **Node level:** every FAIN node (either type A or C) offers the same API to the code running on them. In this way the same code may run on the different node types without need for modification (in this case the portability is limited to the same EE). PDPs are portable to different PEPs, because PEPs offer the same interface. PEPs are portable across different FAIN nodes (type A or C).

  The ASP uses Java, OpenORB, and other dependencies (Node Management

Framework). Management Components are not compatible with the Node Management Framework cannot be deployed directly with the ASP yet. But the ASP is used to transport the code from the Service Repository and left it in the local service repository, the one located in each management station. For this particular case the installation and instantiation is done by the Management Framework.

- **Network level:** For the network ASP what we said regarding the ASP remains valid. The ASP uses Java, OpenORB, and other dependencies (Node Management Framework). Management Components are not compatible with the Node Management Framework cannot be deployed directly with the ASP yet. But the ASP is used to transport the code from the Service Repository and left it in the local code repository, the one located in each management station. For this particular case the installation and instantiation is done by the Management Framework.

Context independency:

Terminology: Is a specification done for a specific virtual environment able to run within the context of another virtual environment without any modification.

Evaluation Methodology: Assess system design and implementation choices.

Evaluation Results:

Transport plane:

- **Technical level:** Not Applicable
- **Node level:** Not Applicable
- **Network level:** Not Applicable

Control plane:

- **Technical level:** Not Applicable
- **Node level:** Not Applicable
- **Network level:** Not Applicable

Management plane:

- **Technical level:** Not Applicable
- **Node level** Service or policy descriptions are understood in different context. The universal service descriptor can be applied to services supposed to run in different EEs. The high level description can be translated into different low level implementations and downloaded by the ASP to the actual EE. Through the description, the service becomes portable across different EEs.

  ASP deploys different services throughout EEs independently on node type and EE type.
- **Network level:** Service or policy descriptions are understood in different context.

  Network ASP deploys different services throughout EEs independently on node type and EE type.

5.2.6 Performance

Unlike the other properties listed in the previous sections, the performance property has been introduced to enable quantitative-measurements. . The evaluation results cannot be interpreted without the context of their measurement, therefore the abstraction of a *level 1 representation* and a *background document*, as proposed in the evaluation framework, has not been applied to the evaluation of the performance. The results of the performance evaluation are directly given in the extensive background document style.

Throughput in User Space (DeMUX System):

Terminology: Evaluates the throughput for packet handling at the node and network level, which (for active networks) does not only depend from bandwidth and packet forwarding performance at nodes, but also from de-multiplexing, packet processing, performance (?), VE management function and control functions related to the execution of multiple concurrent EEs, etc.

Evaluation Methodology:

Figure 5-4 depicts the test system used to evaluate the DeMUX performance. The test system is composed of two nodes. One is for sending flows and the other is an active node where is installed the demultiplexing function. The connection between the two nodes is a 100Mbps Ethernet. As shown in the Figure 5-4, multiple sender programs can be instantiated in the sender node. On the other hand, multiple receiver programs can be instantiated in the active node. In addition, to evaluate the performance of the DeMUX, especially data transmission performance from the DeMUX program to service program, a shaper program is instantiated for each flow that is sent by each sender program.

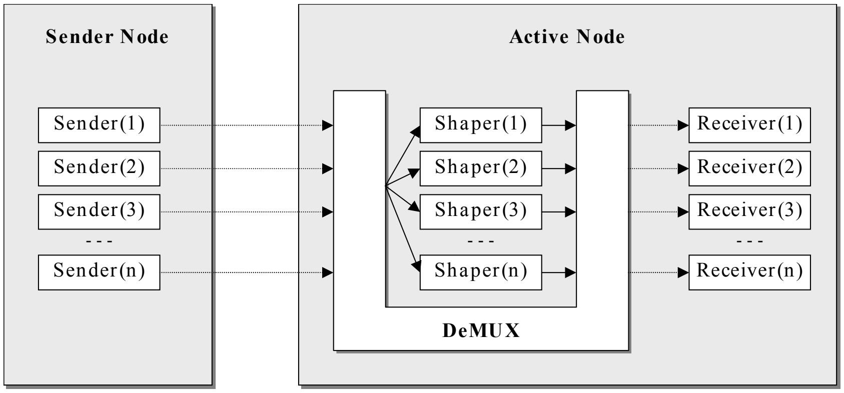


Figure 5-4: DeMUX Test System

Table 5-7 shows a specification of the packet sender. The sender is able to send bit rates from 0 to 2 Mbps. The packet size can be set from 0 to 5kByte. In addition, Table 5-8 shows specifications of the active node. The DeMUX program is implemented in a Linux box with 750MHz CPU, 128Mbyte RAM and a 100Mbps LAN card. One of the important functions is the Iptables, which is the component to transmit data from kernel to user space.

Table 5-7: Specification of the Packet Sender

Sender Program	Bit Rate	Packet Size
Specification	0 - 2Mbps [250kByte/sec]	0 - 5k Byte

Table 5-8: Specification of the Active Node

No.	Item	Specification
1	CPU	Intel Pentium III, 750MHz
2	Memory	128Mbyte
3	Operating System	Linux, Kernel 2.4.2
4	Network Card	100Mbps Ethernet
5	Iptables	1.2.6a

The evaluation of the DeMUX was performed on a dynamic Shaper component as illustrated in the Figure 5-5. The Shaper component requests the Channel Manager to create a new data channel. The Channel Manager creates the new Data Channel and configures the Netfilter. Furthermore, it dynamically connects the Data Channel to the Shaper component. Then the Netfilter intercepts a packet data that matches one of the conditions, and transmits it to the Channel Manager. Upon receipt the Channel Manager retransmits the packet data to the Shaper through the Data Channel. The Shaper can change the data rate and send back the data to the Data Channel. After receiving packet data from the Shaper, the Data Channel sends it back to the outside network through the Netfilter.

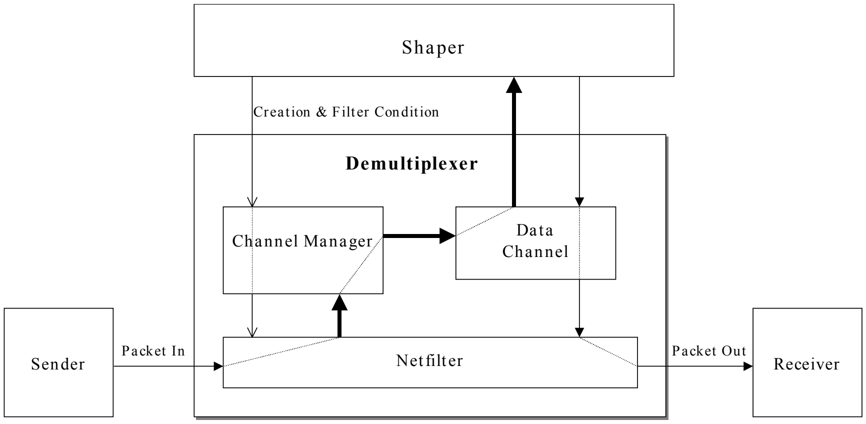


Figure 5-5: System Diagram for Demultiplexing Evaluation

Before starting with the evaluation of the DeMUX system the performance of the default system as depicted in Figure 5-6 has been evaluated. The specification of the active node is shown in the Figure 5-5 and the settings of the sender node are shown in the Figure 5-6.The default performance of the data transmission between the sender node and the active node was evaluated by increasing the bandwidth of the sender flows. Since the sender program could only send up to 2Mbps by one flow, multiple flows were used to obtain bandwidths higher than 2Mbps. The sender node managed to send about 20Mbps data and the active node managed to receive them. The 20Mbps bandwidth was composed by ten-2Mbps data flows. At that time, 500 IP datagram packets were sent per second. Further measurements showed that 20Mbps was a performance limit of the sender node.

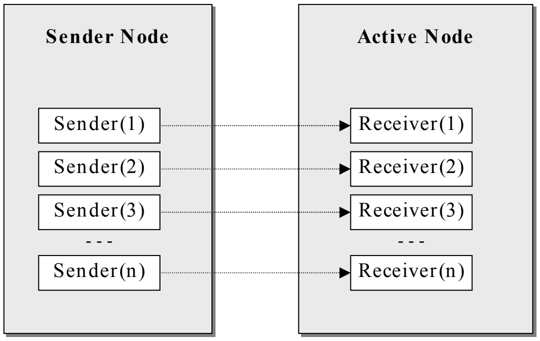


Figure 5-6: Block Diagram of the Default Data Transmission

Table 5-9: Specification of the Sender Node

No.	Item	Specification
1	CPU	Intel Pentium III, 850MHz
2	Memory	256Mbyte
3	Operating System	Linux, Kernel 2.4.2
4	Network Card	100Mbps Ethernet

Bit Rate Performance Evaluation of the DeMUX

The performance of the DeMUX was evaluated by increasing the bit-rate of the flow from the sender node. A specification of the flow that was used in the evaluation is shown in the Table 5-10. When the DeMUX tried to receive over 6Mbps data the receiver experienced packet loss. The DeMUX can therefore handle about 6 Mbps data flow. The localization of the exact performance was difficult since the measurements lacked of an accurate sender.

Table 5-10: Specification of the Flow that is 5kByte long data

Flow	Bit Rate	Packet Size	IP Packet Interval	Remark
Specification	0 - 2Mbps	5kByte	0 - 50 packet/sec	Fragmented

IP Datagram Packet Rate Performance Evaluation of the DeMUX

In addition to the previous experiment, the performance of receiving IP datagram packet was evaluated. A specification of the flow that was used in the evaluation is shown in the Table 5-11. The length of the IP datagrams was 1kByte long. In this case, maximum bit-rate of one flow resulted with 400kbps. When the DeMUX tried to receive over 1.2Mbps data, packet loss occurred even if the bit rate was under 6Mbps. The condition of 1.2Mbps is realized by three 400kbps flows. In this case, 150 IP datagram of the size of 1kByte were send per second. The DeMUX can therefore handle about 150 IP datagram packets per second.

Table 5-11: Specification of the Flow that is 1kByte long data

Flow	Bit Rate	Packet Size	IP Packet Interval	Remark
Specification	0 - 400kbps	1kByte	0 - 50 packet/sec	Not Fragmented

IP Packet Rate Performance Evaluation of the DeMUX

As a final experiment the performance of receiving an IP packet, which is not an IP datagram packet, has been evaluated. This was done to investigate IP fragmentation. The IP datagrams had the size of 2.5kByte as shown in Table 5-12. The maximum bit-rate of one flow results in 1Mbps. When the DeMUX tried to receiver over 3Mbps data, the receiver program detected packet loss. The condition of 3Mbs is realized by three 1Mbps flows. In this case, 150 IP datagram packets per second were sent. In other words, about 250 IP packets of the length of 1.5kByte were sent per second. According to these three experiments, the DeMUX can handle about 150 IP datagrams and it seams that the influence of fragmentation can be neglected.

Table 5-12: Specification of the Flow that is 2.5kByte long data

Flow	Bit Rate	Packet Size	IP Packet Interval	Remark
Specification	0 - 1Mbps	2.5kByte	0 - 50 packet/sec	Fragmented

Results:

According to these three experiments, the DeMUX can handle about 150 IP datagram packets per second or put it in another metric, the DeMUX can handle about 6Mbps data at least. In these evaluations, even if the receiver program detected the packet loss, an exact program that discarded packet was not investigated. Therefore, to obtain a more detailed and accurate evaluation of the DEMUX performance, further tests may have to be run.

Response Time:

Terminology: Evaluates the network and note response time for different control and management operations. For example, how long it takes to install a new service (VE, EE at involved nodes, initial code needed, etc.).

Evaluation Methodology: In order to perform the evaluation we set up a VAN for deploying a particular service. We are going to measure:

1. Bootstrap time of NMS, and EMS
2. Time required for generating the appropriate NL policies for setting up a VAN.
3. Time required for creating a VAN
    a. Time for deploying functional domain (NMS - QoS PDP/PDP)
    b. Times required for enforcing the corresponding element level policies through its appropriate EMS. (Time for creating a VE by mean of policies).
        i. Time for deploying functional domain (EMS - QoS PDP / AN-Service PEP)
4. Time required for activating a VAN
    c. Time for deploying functional domain (NMS- Delegation PDP/PEP)
    d. Time required for enforcing the corresponding element level policies through its appropriate EMS. (Time for activating a VE by mean of policies).
        i. Time for deploying functional domain (Delegation PDP)
        ii. Time for activating VE.

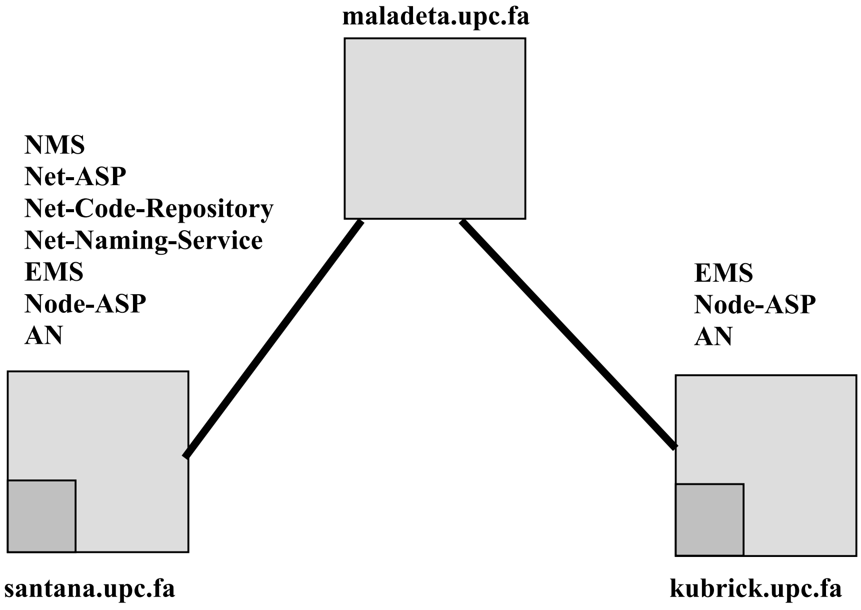


Figure 5-7: Topology for Evaluation Measurement

Figure 5-7 illustrates the topology used and where the systems used are located. All the measurements taken might vary depending on:

- How the network topology used is? (Bandwidth, delay of links
- Where the systems are located?
- How much loaded is the system?

santana.upc.fa is a Pentium IV 1.5 GHz with 500 MB RAM.

kubrick.upc.fa is a Pentium III 666 MHz with 378 MB RAM.

maladeta.upc.fa is Pentium 166 MHz with (it acts as a legacy router)

<u>Evaluation Results</u>:

Bootstrapping		
NMS	**EMS-santana**	**EMS-kubrick**
4911	4908	3285
6059	6011	3273

Table 5-13: Bootstrapping Measurements

<u>Comments</u>:

As can be seen in Figure 5-7 santana.upc.fa has more load than kubrick. This explains why the same process of bootstrapping the EMS is bigger in santana than in kubrick.

NMS:

SM Generate Policies	VAN Creation	VAN Activation	Deploy Functional Domain At NMS	
			QoS	Dlg
1481	41607	56619	1012	175
1072	36121	51127	946	336

Table 5-14: NMS Measurements

Comments:

The measurements done have been taken under a different situations. For the 1st trial we decided to remove all data from the local code repository (i.e. Service Descriptors and packages of services). This is the reason why in the 2nd trial, the required time for activating a VAN is not affected by the delay associated to retrieve the Service Descriptors and the corresponding java packages from the net service registry and the network code repository.

EMS- santana

Deploy Policy		Deploy Functional Domain		
QoS	Dlg	QoS PDP[4]	Service PEP[5]	Dlg PDP
19697	10776	6187	4594	167
18007	10375	5015	3567	149
1410	5097	0	0	0

Table 5-15: EMS-santana Measurements

EMS – kubrick

Deploy Policy		Deploy Functional Domain		
QoS	Dlg	QoS PDP[4]	Service PEP[5]	Dlg PDP
10922	7734	6069	5293	200
9862	7906	5089	4338	185
1741	7326	0	0	0

Table 5-16: EMS-kubrick measurements

Comments:

[4] The given time includes the time required for deploying the Service PEP.

[5] The Service PEP consists of two components the QoS and Delegation of Access Rights PEP, which will be deployed of the PVE and bound together.

This evaluation has been done by applying a 3rd trial. The 1st and 2nd were done just after the bootstrap of the management system. Under these circumstances only the ANSP Proxy component and the PDP Manager component of the ANSP instance were running. Another thing to consider was that as soon as the deployment of the QoS functional domain is triggered, the QoS PDP will be deployed on the management station and the Service PEP will be deployed inside the Privileged Virtual Environment located in the AN.

The time to deploy an element level QoS Policy for creating a VE for the 1st or 2nd trial are bigger than for the 3rd trial. The reason is that in the 3rd trial there is no delay associated to the extension of the management system, i.e. the QoS Domain and the Service PEP are already running on the system so there is no need for triggering their deployment.

The same does not apply for the deployment of the Delegation of Access Rights. Due to that the delay associated to deploy a Delegation of Access Rights is lower than for the QoS PDP, since the delegation of access rights PEP was already deployed during the deployment of the service PEP.

The time for deploying an element level Delegation of Access Rights for activating a VE is bigger that the time required for deploying a QoS policy for creating a VE. This is due to the fact that during the creation of the VE we only allocate the resources required but during the activation of the VE the already allocated resources must be created, which is an operation that requires more time.

PromethOS Performance Evaluation on Wave Video Plugin

We evaluate the performance of our Wave Video plugin on our active node, which is a Pentium III PC running at 800 MHz. In order to allow for very accurate measurements, we use the Pentium's processor clock register TSC which is incremented on every CPU cycle. Measuring CPU cycles provides a certain degree of independence of the CPU speed but depends on the architecture and release of the CPU, i.e. it is obvious that a CPU running at double the frequency can spend much more time to handle packets; however, executing an assembler instruction like a register-to-register move instruction requires only one CPU cycle on a specific CPU release, for example. However, measuring CPU cycles and providing a meaningful explanation is extremely difficult in a commonly used operating system since every operating system internal states (like memory management issues) may have a significant influence on the measured value.

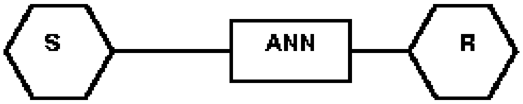


Figure 5-8: Network for performance measurements

Of interest to our measurements are the CPU cycles spent in the PromethOS framework when a Wave Video packet flows along IP network stack at the active node R (Figure 5-8). In our experimental setup, we configured PromethOS and the Wave Video plugin such that the plugin is attached to the PRE-ROUTING hook of the Netfilter framework. The relevant objects for these measurements are depicted inFigure 5-9: Evaluated Components

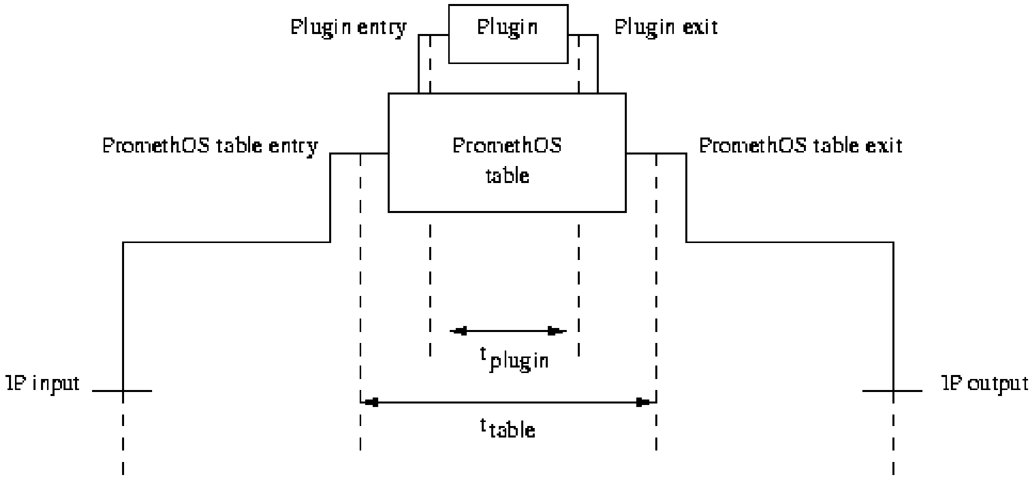


Figure 5-9: Evaluated Components

We measured the number of CPU cycles spent in the PromethOS table, in the Wave Video plugin, in a null plugin and without plugin. Transmitting the Foreman-video leads to approximately 17000 packets that need to be processed by the Wave Video plugin.

Components measured	**Cycles consumed**
T plugin	
Always	795
Regular	508
T table	
Plugin-always	1460
Plugin-regular	1158
Null	474
Empty	156

Table 5-17: PromethOS Measurements

Table 5-17 provides an overview of measured CPU cycles. The time spent in the Wave Video plugin is referred to as t plugin; the time spent in the PromethOS table as t framework. For testing purposes, we also measured a Null-plugin, a plugin without functionality, of which we refer to the cycles used by section null in the t table list of figures; the section "empty" refers to the configuration where no plugin was installed in the PromethOS framework.

For the Wave Video plugin two numbers of CPU cycles are provided depending on whether adjustments to the Wave Video filter tables are required or not. Note that the filter table is recomputed at fixed intervals (currently each 100 ms). We refer to the configuration where every arriving Wave Video plugin leads to a re-computation of the Wave Video filter tables with the index (extension) "always". The regular configuration of the Wave Video plugin, in which a filter table adjustment takes place only every tenth of a second, is referred to by the index (extension) "regular".

In the regular case, we achieve a minimal requirement of 508 CPU cycles for the Wave Video plugin; on our active node this is equivalent to approximately 635 ns per packet. The PromethOS table requires additional 650 CPU cycles.

To estimate the overhead created by the PromethOS table itself, the PromethOS framework was run "empty" and with the null plugin. Calling the empty PromethOS table consumed 156 CPU cycles. This figure indicates the cycles consumed to run through the empty list at the PRE-ROUTING hook.

Calling the null plugin requires 474 cycles. This figure leads to the indication that the overhead created by the PromethOS table is mainly due to the design of Netfilter and, since PromethOS seamlessly fits into the Netfilter framework, of PromethOS itself which make use of several indirect function calls.

Referring to the measurement results in Table 5-17, PromethOS proved to create little overhead.

Authentication (Performance):

Terminology: Authentication allows the system to securely verify the identity of a principal.

Evaluation Methodology: This feature has been evaluated in two respects. Firstly, we estimate to what extent this feature has been provided within the FAIN active node (prototype). This is depicted in Table-2. Secondly, based on the experimental measurements with the FAIN security architecture prototype, we have tried to estimate the performance overhead imposed by this feature. The later are presented below.

Evaluation Results:

Transport plane, Control plane, and Management plane:

- **Node level:** Initial performance measurements shows, that the node can process over 7000 active packets in the case of per hop authentication. Using digital signatures based authentication the node can authenticate data origin of 570 active packets, including the overhead of packet decoding and validating a certification path. With credentials caching 1700 authentications can be performed.
- Measurements were done on commodity PC, with Intel P4 2.2 GHz processor , 512 MBit RAM, Red Hat Linux 8.0, kernel 2.4.18-14, Java SDK 1.3.1_3, Bouncy Castle cryptolibrary version 1.17 and network node related FAIN code. Digital signature algorithm was RSA encryption with SHA-1 hash, key size 768 bits, X.509 certificates were signed with RSA encryption with MD5 hash, key size 1024 bits, certification path length was 1. In the case of per hop authentication we have used HMAC -SHA-1 as keyed hash.

Integrity (Performance):

Terminology: Integrity enables the system to detect any modifications of the information in transit over the network by unauthorized adversaries.

Evaluation Methodology: This feature has been evaluated in two respects. Firstly, we estimate to what extent this feature has been provided within the FAIN active node (prototype). Secondly, based on the experimental measurements with the FAIN security architecture prototype, we have tried to estimate the performance overhead imposed by this feature.

Evaluation Results:

Transport plane, Control plane, and Management plane:

- **Node level:** Hop-by-hop integrity is provided based on a keyed hash function, when packets need to be modified at FAIN ANNs en route. Experimental measurements indicate that validating integrity represents 12% of the total packet processing cost on the FAIN node. Testing environment was the same as specified in authentication description.

- **Network level:** End-to-end integrity is either provided with digital signature (when packets are not modified en route) or can be incurred from per-hop protections, when packets are processed at ANNs. The cost of integrity provisioning for the static part of the packet that doesn't change in the network the same results apply as in the case of authentication. In the case when the cached credentials are used integrity validation represents 52% of the packet processing costs.
-

6 CONCLUSIONS

The presentation in this document serves to assess and justify the main claims that the FAIN project has in the area of research and development in active network technology.

Claim 1: FAIN has produced and demonstrated a novel architecture for an active network node, which implements the concept of a virtual environment and the simultaneous use of multiple execution environments of different types to enable a very flexible, dynamic creation and deployment of services.

The justification of this claim has been shown by successful demonstration of a series of complex application scenarios, each involving on-demand deployment of a service and the execution of an application using such services.

Claim 2: FAIN has produced an architecture which is capable of supporting active services on three different planes, where each plane has different requirements in terms of flexibility and performance. The three planes are the transport plane, the control plane and the management plane.

Claim 2 has been shown to be justified by the fact that FAIN enables different EE types to interoperate and jointly participate in the provisioning of a complex distributed service. High performance transport plane functions are supported by the PromethOS execution environment, while medium speed, but highly flexible control functions may be realized in a Java- or CORBA-based execution environment. Active functions in the management plane and their interaction with the other planes have been shown to be possible with a policy-based management approach, where executable policies fulfill the requirement of flexibility in this plane.

Claim 3: FAIN has developed a service description and deployment approach, which is independent of the specific type of platform on which service components are executed.

Claim 3 has been fulfilled by a using a platform-independent service specification, which determines the service to be deployed in an active node. Services may be complex aggregates consisting of several components that interoperate among each other and across execution environment boundaries.

Claim 4: The FAIN architecture, design and implementation fulfils the evaluation criteria of flexibility, security, interoperability, openness, portability and performance to a high degree.

Claim 4 has been shown to be justified by the extensive discussion of why these criteria are fulfilled contained in chapter 5 of this report.

Recommendation

While we feel that the work done in FAIN may have closed the book on important work in the basic structure and functionality of an active network node and the instantiation of execution environments, we feel that the project opened another book, which is not written as yet: The problem of dynamic service provisioning, especially in a network-wide scope, has only been scratched on the surface. We feel that a future phase of research in programmable networks should concentrate on this problem, and should incorporate and integrate methods and technologies for service creation, deployment and management that have been considered in the areas of active networks, peer-to-peer-networks, ad-hoc networks, leading towards truly self-organizing networks and services.

7 ACRONYMS

AN	Active Network
ANEP	Active Network Encapsulation Protocol
ANN	Active Network Node
ANSP	Active Network Service Provider
API	Application Programming Interface
ASP	Active Service Provision
CORBA	Common Object Broker Architecture
CS	Core Scenario
DNS	Domain Name System
DNSEC	DNS Security Extensions
DSCP	DiffServ Code Point
EE	Execution Environment
EMS	Element Management Station
GAS	Generic Application Scenario
IPSec	IP Security Protocol
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MI	Management Instance
NMS	Network Management Station
OS	Operation System
PDP	Policy Definition Point
PEP	Policy Enforcement Point
pVE	privileged Virtual Environment
QoS	Quality of Service
RM	Resource Manager
SA	Secure Association
SBB	Scenario Building Block
SCE	Service Creation Engine
SID	Secure Identifier
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SNAP	Safe and Nimble Active Packets
SNMP	Simple Network Management Protocol
SP	Service Provider
TA	Technical Annex
TOS	Type of Service

VAN	Virtual Active Network
VE	Virtual Environment
VEM	Virtual Environment Manager
VN	Virtual Network

8 REFERENCES

[1] Requirements Analysis & Overall AN Architecture. FAIN Deliverable D1, WP2-DT-004-D01-Int. FAIN consortium, May 2001.

[2] C. Klein, L. Mandl, “Web Service Distribution: A FAIN application scenario with PromethOS – V1.0.doc”

[3] E. Pfeuffer, R. Schmid, C. Meyer, C. Niedermeier. WP3-SAG-029-MobileScenarioPromethOS-INT-V03.doc, April 2003.

[4] http://www.videolan.org/ , 3 April 2003.

[5] http://www.videolan.org/pub/vlms/0.2.3/rpm/vlms-0.2.3-1.i586.rpm , 3 April 2003

[6] http://www.videolan.org/pub/vlc/0.5.0/vlc-0.5.0.tar.gz 3 April 2003

[7] http://www.videolan.org/pub/libdvdcss/1.2.5/libdvdcss-1.2.5.tar.gz 3 April 2003

[8] WP5-HEL-053-DiffServExe-intv.doc

[9] Set in Properties/Summary Subject (F9 to update), FAIN Deliverable D1

[10] Initial Active Network and Active Node Architecture, FAIN Deliverable D2

[11] Initial Specification of Case Study Systems, FAIN Deliverable D3

[12] D. Scott Alexander, Bob Braden, Carl A. Gunter, Alden W. Jackson, Angelos D. Keromytis, Gary J. Minden, David Wetherall, "Active Networks Encapsulation Protocol", Draft RFC, July 1997.

[13] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden, “A Survey of Active Network Research”, IEEE Communications Magazine, Vol. 35, No. 1, pp80-86. January 1997.

[14] 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, Vol. 29, No. 2, pp. 7-24, April 1999.

[15] Next Generation Networks Initiative, http://www.ngni.org/overview.htm

[16] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, “SIP: Session Initiation Protocol”, RFC

[17] Ralph Keller, Lukas Ruf, Amir Guindehi, Bernhard Plattner, PromethOS: A Dynamically Extensible Router Architecture Supporting Explicit Routing , Proceedings of the Fourth Annual International Working Conference on Active Networks (IWAN 2002), Springer Verlag, Lecture Notes in Computer Science, 2546, December 4.-6. December 2002, Zurich, Switzerland.

[18] EMS, FAIN Deliverables n.5 and n.8

[19] NMS, FAIN Deliverables n.5 and n.8

[20] VEM, FAIN Deliverables n.5 and n.8

[21] C.Kitahara, S.Denazis, C. Tsaurochis, J. Vivero, E. Salamanca, E. Magaña, A. Galis J.L.Mañas, Y.Carlinet, B.Mathieu, O. Koufopavlou "A Policy-Based Management Architecture for Active and Programmable Networks", Network Magazine special issue on "Network Management of Multi-service, Multimedia, IP-based Networks" to be published in May/June issue 2003

[22] Marcin Solarski, Matthias Bossardt, Thomas Becker: Component-based Deployment and Management of Services in Active Networks. In Proceedings Fourth Annual International Working Conference on Active Networks (IWAN 2002), Zürich, Switzerland, Lecture Notes in Computer Science 2546, Springer Verlag, Berlin Heidelberg New York, December, 2002.

[23] Service Level Agreement, Inter Domain Manager (IDM). FAIN Deliverable n.8

[24] ASP, FAIN Deliverable n. 8

[25] Network ASP, FAIN Deliverable n. 8