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

D40

CEC Deliverable Nr:	**D40**
Deliverable Type:	**PUB**
Dissemination:	Int
Deliverable Nature:	
Contractual date:	
Actual date:	

Editor:	**Placi Flury, ETH**
Workpackage(s):	**WP3, WP4, WP5**
Abstract:	
Keyword List:	**Scenarios, Testbed, Evaluation Framework**

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

D40

Editor:	**Placi Flury**
Document No:	**D40**
File Name	**D40.doc**
Contributors:	**c.f. Authors Section below**
Version:	**1.0**
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
FhG/Fokus	Partner	Germany
IKV	Associate Partner to FhG/Fokus	Germany
INT	Associate Partner to FhG/Fokus	Spain
UPEN	Partner	USA

The FAIN Consortium

University College London	(UCL)
Josef Stefan Institute	(JSIS)
National Technical University of Athens	(NTUA)
Universitat 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) 207 679 5738

Fax +44 (0) 207 388 9325

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

Authors

Arso Savanovic (JSIS)	Bernhard Plattner (ETH)
Chiho Kitahara(HIT)	Cornel Klein (SAG)
Drissa Houatra (FT)	Dusan Gabrijelcic (JSIS)
Elisa Boschi (FhG)	Epi Salamanca (UPC)
Evelyn Pfeuffer (SAG)	Juan Luis Manas (INT)
Lukas Ruf (ETH)	Marcin Solarski (FhG)
Matthias Bossardt (ETH)	Placi Flury (ETH) - Editor
Reiner Schmid (SAG)	Richard Lewis (UCL)
Spyros Denazis (HEL)	Thomas Becker (FhG)
Toshiaki Suzuki (HEL)	Walter Eaves (UCL)
Yannick Carlinet (FT)	Carsten Meyer (SAG)
Evelyn Pfeuffer (SAG)	Reiner Schmid (SAG)

Table of Contents

Table of Figures

Table of Tables

1 INTRODUCTION

This document presents the effort of the FAIN project in defining application scenarios that reflect the novel functional concepts of FAIN and that support the evaluation of the work done in FAIN. It is a refined version of the deliverable D9, providing detailed information about the FAIN evaluation effort, while D9 has been written to convey a reasonably concise presentation of the same material.

Our approach is the following: First, we elaborate on the concepts introduced and used in FAIN. Second, we define application scenarios that reveal and demonstrate these concepts; these scenarios are thereafter used to qualitatively validate the concepts, matching them to the facilities offered by FAIN.

The document is isomorphic to this approach. The first chapter is dedicated to the FAIN concepts. In order to deduce scenarios from the concepts, a scenario definition framework has been designed and is described in chapter 3. The framework permits reusability of functionality by following a component-based approach. The components, called Scenario Building Blocks, allow for the specification of arbitrary scenarios. The scenario definition framework provides the abstraction needed to define scenarios focusing on the concepts they embody rather than a specific implementation in a specific network topology. This method of abstraction is comparable to the one found in programming languages, where one differentiates between declarations and instantiations. After defining the abstract scenarios (called generic application scenarios) they are mapped onto the infrastructure provided by FAIN: The Fain testbed, consisting of the FAIN active network nodes, management stations and supporting servers and repositories. An executable scenario (showing all the details of the implementation, as needed for actually executing a demonstration of such a scenario) is called an application scenario. To validate the relevance and usefulness of the FAIN concepts, an extensive evaluation framework has been defined. The purpose of the evaluation framework is to assess whether the key properties identified in the FAIN architecture documents (flexibility, security, interoperability, openness, portability and performance) are actually fulfilled. This chapter 8 will allow the reader to assess the quality of the design and implementation of the FAIN technology. Appendix 1 contains detailed evaluation of the FAIN Mobility demonstrator

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.

Creation of a new VN Management Domain as Part of the VN Creation

New VN management domains are created as a pre-requisite for VN creation. This concept actually concludes the cycle of VN creation.

Use of new VN Management Domain to manage Services and Resources

Simultaneous operation of two different management domains to manage their own services, possibly using different policies that are only meaningful by the corresponding domain. This concept also includes requirements such as customized monitoring for each domain, and a variety of Resource Monitoring systems (RMs) that may be installed and used to achieve specific objectives.

ASP Specification and Deployment

A service suitable for the scenarios must be described here that demonstrates the ASP functionality and combined with concepts 3 and 4.

Tuning the Active Network for maximizing Performance

Maximal performance can be achieved with an intelligent combination of the different platforms available. According to the type of Active Node (within FAIN, Type A & Type C nodes) the right EEs are deployed to maximize gains through performance and flexibility tradeoffs. By using the EE and service component deployment mechanisms an Active Network substrate that is tuned for achieving the desired gain can be created. This may involve Control and Transport plane EE collaboration as part of the same service. For instance, a service provides a network API that is implemented by means of distributing parts of it in a control EE and a transport EE. The control part of the service running in the control EEs (distributed across different platforms Type A and Type C) configures the transport plane functionality either by controlling the resources of the nodes and/or by introducing additional functionality (service components)

Using Active Networks for Policy Distribution

The use of active packets for distributing policies provides a higher degree of flexibility to the FAIN management system and demonstrates that the system takes advantage of active technology also for management tasks.

Simple fault management functionality

The idea should be to demonstrate some simple fault management functionality. Functionality such as alarm filtering or correlation is clearly unrealistic for the actual deadlines.

Network-level deployment mapping

The deployment process at the network level includes finding a target environment (a set of nodes/EEs which are most suitable for the service deployment) and a mapping: service components to the corresponding EEs.

Concepts (3) and (4) deal with the deployment of multiple service components and their interactions. Finding an optimal target environment is also important when deploying a service. This is the core functionality of the network ASP.

Active Network Upgrades

A deployed service has to be redeployed and replace the service previously deployed.

Active Networks promise extendibility. As the service evolves and new (better) versions are available, the services deployed on active nodes have to be replaced. After an active service is deployed, it may be the case that it needs to be upgraded for some reason (malfunctioning, a better service variant available)

3 SCENARIO DEFINITION FRAMEWORK

In order to define expressive scenarios, a framework, called *scenario definition framework* is introduced. As depicted in Figure 3-1 the framework consists of three layers. The lowest layer holds so called *Scenario Building Blocks* (SBBs), which are used for building the *Generic Application Scenarios* shown on layer two. The last layer represents the *Generic Application Scenarios*. Those are instances of the Generic Application Scenarios.

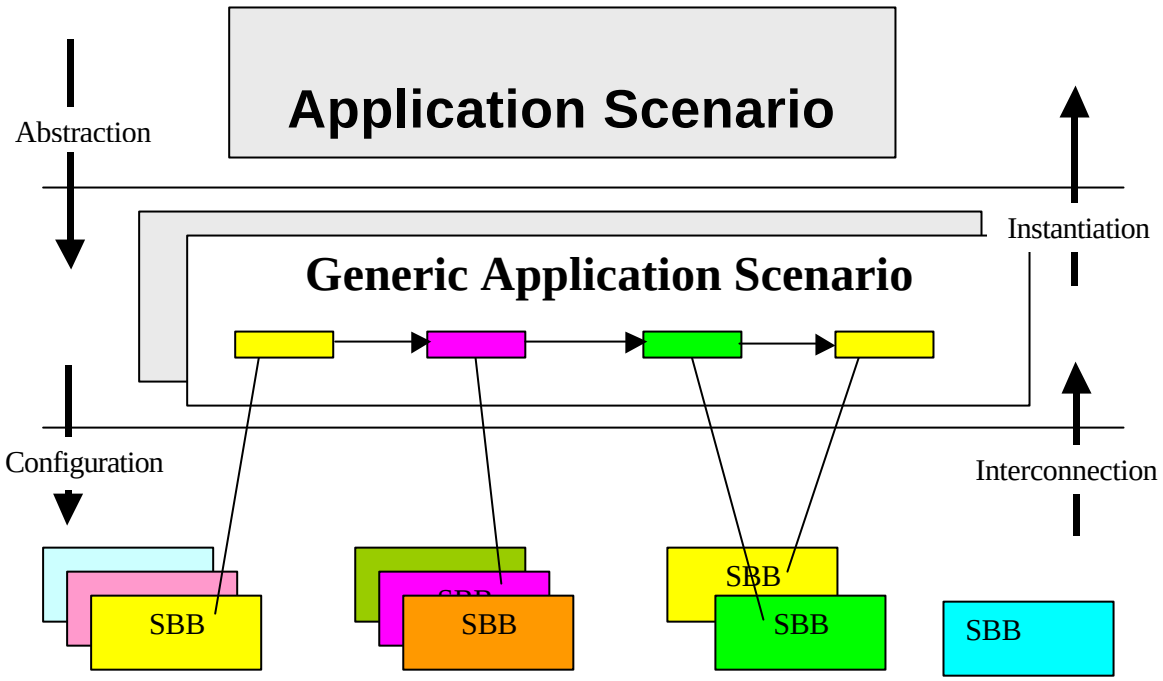


Figure 3-1: Scenario Definition Framework Model.

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 (better Generic Application Scenario) need to be mapped to the corresponsive SBBs only.

3.1.1 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 not overlap.
SBBs make up a Generic Application Scenario by sequential interconnection. An example for the interconnection is given in Figure 3-2. In order to keep the SBB specification simple, nested combinations of SBBs are not allowed.

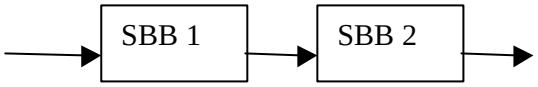


Figure 3-2: Sequentially Connected SBBs.

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 includes those pre- and post conditions.

3.1.2 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). 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.

3.1.3 Application Scenario

As told 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 the testbed, with its different sites and nodes, in FAIN. 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 CORE SCENARIOS

To alleviate the identification of all the necessary SBBs for the intended generic application scenarios we make use of simplified ‘mini’ scenarios called core scenarios. The core scenarios are not part of the scenario definition framework. However, they represent our way to start with braking down functionality to basic and reusable components for scenario definition. Three core scenarios are used for that purpose. Each refers to the concepts and objectives it reflects.

Security aspects of FAIN are handled in an overall way. They have therefore not been associated to a particular core scenario, but are instead, since they apply equally to all core scenarios, discussed separately.

CS1: Virtual Network Creation including Privileged VN Bootstrapping

CS1 is one of the most important of the core scenarios as it requires the collaboration between the different work packages as well as different components within the individual work packages. It is obvious that the generic scenarios rely on the SBBs resulting from this core scenario for the instantiation of the SP’s VN.
SP requirements are submitted to the portal of the ANSP in the form of arguments. This enfolds also the request for the creation of VNs, resulting in the bootstrapping of the Privileged VN (Privileged VEs + ANSP management domain + ASP.
Following stages for the realization of CS1 have been identified:

- Admission Control

During this stage the ANSP checks the availability of the resources in order to admit the SP’s VN.

- VN Activation and SLA Enforcement by means of policies

After successful admission the ANSP registers the new VN across the network nodes that are part of the topology. It enforces the SLA by configuring components like the security component for authentication and authorization enforcement, resource managers for resource enforcement etc. Part of this stage is also the configuration of routing tables so that packets that belong to a certain VN flow within the topology of the VN. Furthermore, we also need to include policers to enforce an SLA and make sure that the SP stays within his contract. Such policing mechanisms belong to the ANSP and keep the different service providers isolated from each other.

- A minimum number of flows
- VN instantiation must include the creation of a number of initial flows that are required for the communication among the management entities of the newly created SP management domain. Additional flow creation may happen at another occasion when services are deployed. Privileged VN Bootstrapping

During the bootstrapping stage the Privileged VE is created and the corresponding interfaces are exported. Although this is the first step in realizing this core scenario it has also to go through all the previous stages. The configuration may be carried out in a hard coded and automated way.

CS 2: Flow and Data Path Creation for Service and User Communication

After the creation of the VN and its corresponding management domain, the SP must be able to use the assigned resources. One core functionality will be flow creation across the VN network by configuring the corresponding virtual nodes, as well as data path creation that is used for processing the flows. According to this scenario the SP management domain must create a policy, based on which a flow is enforced and the VN's resources are reserved. Before that, the corresponding ANSP RMs must check if the request is authorized, it does not violate the SLA and finally carry out the request. Note that part of a flow creation is the association with a number of resources (computational and communication). These facilities are configured so that packets that belong to this flow are directed through the proper EEs and service components. To carry out this important scenario we need a model based on which we will implement the flow and data path creation concepts.

CS 3: Deployment and Instantiation of Services and Service Components

This core scenario involves the service description, deployment, and binding of the service components inside an EE (deployed previously). It has network level (ASP) and node level aspects (VEM). It is also associated with the created flows as it is obvious that these components will process packets that belong to specific flows.

We start by making some assumptions for CS3:

- There is one source of the video stream. It emits video data with a given format. The format does not have to be suitable for the network it is sent through.
- There may be a number of users interested in receiving the signal. The users can access the active network at any of the edge points through a link defined by different parameters than the core active network, like a limited bandwidth, worse communication reliability, etc.
- Format conversion is needed as the users can receive a format different from the one sent by the video source.
- Conversion has to happen in the active network as the users use terminals with limited processing power

Three variants of the scenario are proposed for further investigation:

1. The transcoder service consists of one service component that has to be deployed onto a most suitable node. A group of users are interested in receiving the same video stream in the same format. The location of all the users is known. The objective is to choose a target node so that only one instance of the transcoder needs to be deployed for the group. The node has to provide ample performance to deal with resource consumption of the transcoding process. As the user terminals cannot handle multicasts, point to point connections are needed. To reduce the network traffic, generated by the transcoder resending the data in the format suitable to the receivers, the target node location should be selected so that the node is as close as possible to the group of users.
2. The transcoder service consists of two service components to deploy on two different active nodes. The sender is an end user using limited capabilities of his terminal to send a video stream. The video stream is to be received by a few of other end users connected to the network at different locations. Their terminal processing capabilities are limited as well. For optimal data transfer through a network, the signal needs to be converted into another format. The solution is to install a distributed transcoder. One service component should be installed close to the sender of the video stream and convert a video format that the sender uses into an intermediate format that is most suitable to transmit in the network, i.e. be optimized to match the network traffic characteristics. The other part of the transcoder has to convert this intermediate format into a format most suitable for the video receiver. The latter component of the transcoder has to be installed in a most suitable location as described in scenario a)
3. The transcoder service consists of data path and control service components. The components have to be deployed on two or more active nodes. The data path component(s) are as described given by CS2. The control component is the controller (known from the Barcelona demo) that has to be deployed onto the node where the application and web servers are installed.

4.1 Core Scenario Mapping to SBBs

4.1.1 CS 1: Virtual Network Creation including Privileged Virtual Active Network Bootstrapping

Setting up a virtual active network (VAN) for a service provider (SP) includes the creation of virtual environments (VEs) on the appropriate active network nodes (ANNs) by the active network service provider (ANSP) (see figure). Finding the appropriate ANNs involves the alignment of requested and available resources. This implies that the ANSP has to configure its infrastructure. To carry out the necessary steps the ANSP uses its management system. Once the ANSP knows which ANNs to set up, it will try to make a reservation of requested resources and if they are available on all ANNs the ANSP will then request their activation. Once all needed VEs are activated the ANSP will set up the default routes to complete the setup of the VAN.

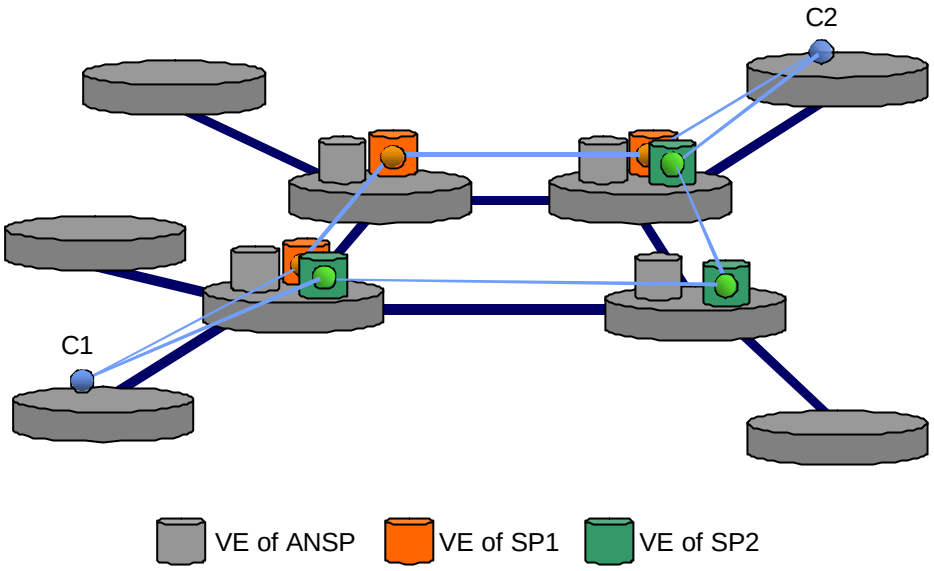


Figure 4-1: Example of Virtual Active Network Setup for two Service Providers.

A special case is the setup of the privileged VAN owned by the ANSP. In this case there is no need to calculate a VAN topology. When the ANNs are booted this includes the (automatic) creation of the privileged VEs forming the privileged VAN.

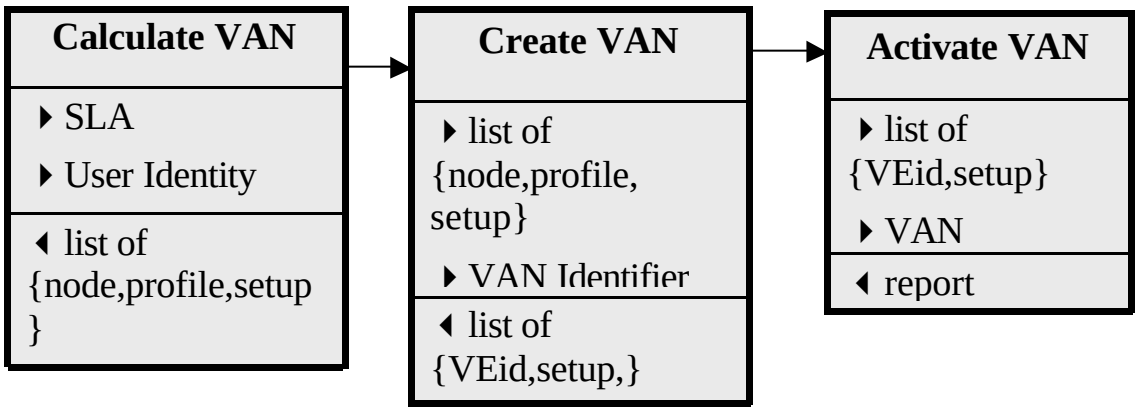


Figure 4-2: SBB for Virtual Active Network Setup.

This chapter starts with presenting the scenario building blocks describing how a VAN is created from the network perspective and then shift the focus to the scenario building blocks related to the creation of a VE on a particular ANN.

SBB Calculate VAN

This building block describes how the ANSP calculates which of all ANNs of the topology will be part of the VAN; it also identifies the corresponding profile and set up parameters of each of VE.

Pre-conditions

The Resource Manager of NMS must be up and running.

All of EMS and ANNs must be up and running

The NMS is also up and running

I assume that we only have one administrative domain, that is, only one NMS.

Post-conditions

The ANSP holds:

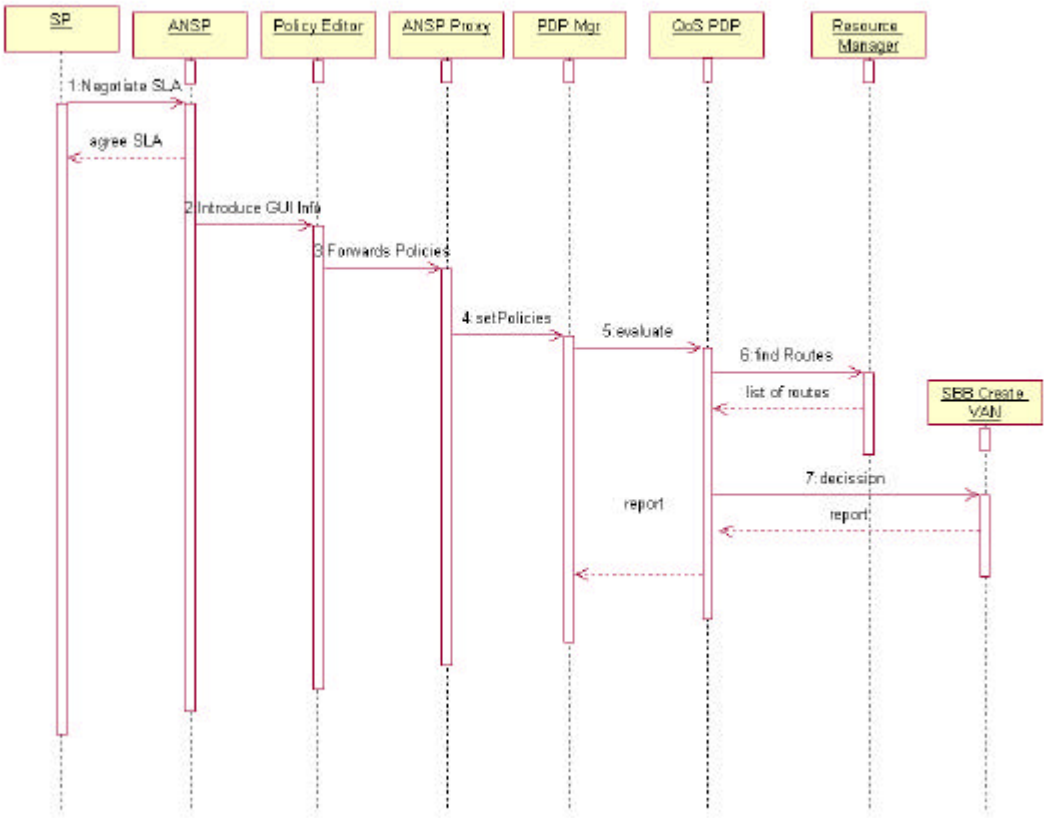
- List of tuples (node, profile, setup)
- VAN Identifier (An unique identifier associated to an unique SP)
- Identity of the owner of the VAN

Dependencies

This building block depends on

- SBB Boot Node
- SBB Boot EMSs
- SBB Boot Resource Manager
- SBB Boot NMS

Sequence Diagram



1. SP negotiates a SLA with ANSP.

2. The ANSP allocates the agreed resources, in the form of a VE to the SP. It introduces the necessary information through the GUI offered by the FAIN Network Management System.
3. The Policy Editor component at the network level station receives this information and translates it into a group of policies that should be enforced atomically. That is, either all policies are enforced or none is. The policies that conform to the group are two: a QoS policy that creates a VE and reserves the resources to it, and a Delegation policy that assigns access rights to that VE, instantiates the VE, as well as creates a new Management Instance for that SP at the element level to allow it to manage these resources.

   The Policy Editor sends the policies from the policy group to the ANSP Proxy element at the network level station, which based on the user information authenticates it and forwards policies to the correct management instance, in this case the ANSP's MI
4. The PDP Manager receives the policy group, stores it in the DB, and starts the processing of the policies that conform to that group. It detects that the policies should be enforced in order and that the delegation should be processed if the result of the processing of the QoS policies is correct. Thus, it starts the processing of the QoS policy requesting to the Access Right Check Component (Not shown) to check whether the sender of the policies (the ANSP) is authorized to introduce such policies.

   Once a positive reply is received, the policy is demultiplexed to the correct PDP component to process that policy.
5. The QoS PDP will retrieve the policy from the DB.
6. Request to the Resource Manager, where in the network the resources requested by the SP can be allocated.
7. With the information about which nodes are involved and since there are no conditions or actions that need to be realized at the network level, the QoS PDP forwards the decision to the PEP. (Here SBB Create VAN starts) QoS PEP will use all these information (nodes, QoS NL Policies, that is information about resources required) and will translate it into the corresponding QoS EL policy and send the policy into the EMS of the established nodes.

   When the enforcement results of the QoS Policy arrive to the QoS PDP, it is forwarded back. The PDP Manager will process it to decide upon the policy group processing.

Implementation Status

Right now the ANSP administrator, who uses the templates offered by the policy editor to generate the NL policies needed for generating the VAN, does the mapping of SLA into a policy set manually. Right now, there are only a few templates defined. Templates required for core scenario will be implemented. There is also a proposal to implement some specific wizard, which will be in charge of creating automatically policies from the SLA.

With respect the resource manager component it needs to be designed and implemented. Right now his functionality has been hard coded.

Also exits a proposal for creating a VAN between two sites separated by more that one administrative domain, which involves that there should exist a component in charge of INTERDOMAIN matters.

Subcomponents:

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
Policy Editor	Helps the ANSP the introduction of policies, offering a set of templates, which will be used by the ANSP in order to generate those policies he wants to	Define default wizards, which automatically generate policies from data inserted by the administrator.	D5

	those policies he wants to enforce. Offers a panel with information related to the actual enforcement status of the policy submitted.	Enhanced the way we show information about current policies already being enforced on the system.	
ANSP Proxy	Authenticate incoming requests Dispatch incoming request to one of the management instance (PDP managers)	Mechanism of authentication must be implemented	D5
PDP Manager			
Domain Manager	Maintain a list of domains already installed on the management station If a new one is required then it dynamically extends itself loading the appropriate class and creating a new instance of it.	Interact with ASP in order to trigger deployment of new code for the new domain.	D5
Forward Controller	Receives policy group and process it according to the forwarding policy defined.		D5
Local Repository	Stores policy and returns an unique id, which will be used by other components to retrieve policy stored.	Uses of a database.	D5
Access Rights Check	Checks if the sender of policies is authorized to introduce such policy	Enhanced it	D5

PDP

Semantic Policy Check	Check if the policy under evaluation is going to enter in conflict with any of the others policies already enforced in the system	The current version is obsolete due to the fact we have modified some field of policies so it must be upgrade!!	D5
Condition Interpreter	The Evaluation engine use it and making use of monitoring system to decide when a policy should be enforced	Basic functionality is done but it needs to be performed and enhanced	D5
Action Interpreter	There is an action interpreter for each policy action. It includes code, which knows what to do in order to collect all information required, which will be sent to the PEP.	If new policy actions are need then new action interpreter should be implemented.	D5
Monitoring System		It is in charge of gathering the state of resources in a given domain nodes in order to provide the other network level components (Delegation PDP, QoSPDP, NL-Resource Manager) with domain wide resources information, refreshed at a time scale to be dimensioned	D5
Resource Manager	Hard coded	Assess the resource utilization information that it has registered to receive from the monitoring system. This evaluation will drive short-term or long-term decisions for admission control, traffic re-routing, resource re-allocation (It would be in charge of find out the appropriate nodes,	D5

		which conform VAN) It must be designed and basic functionality implemented.	

SBB Create VAN

This building block 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.

Pre-conditions

The Service PEP (QoS PEP Instance) has been registered, deployed (in the pVE)

The Service PDP has been registered (QoS PDP instance) (It's going to be deployed into the Management Station)

Post-conditions

The NMS has received a report about the actual policy enforcement

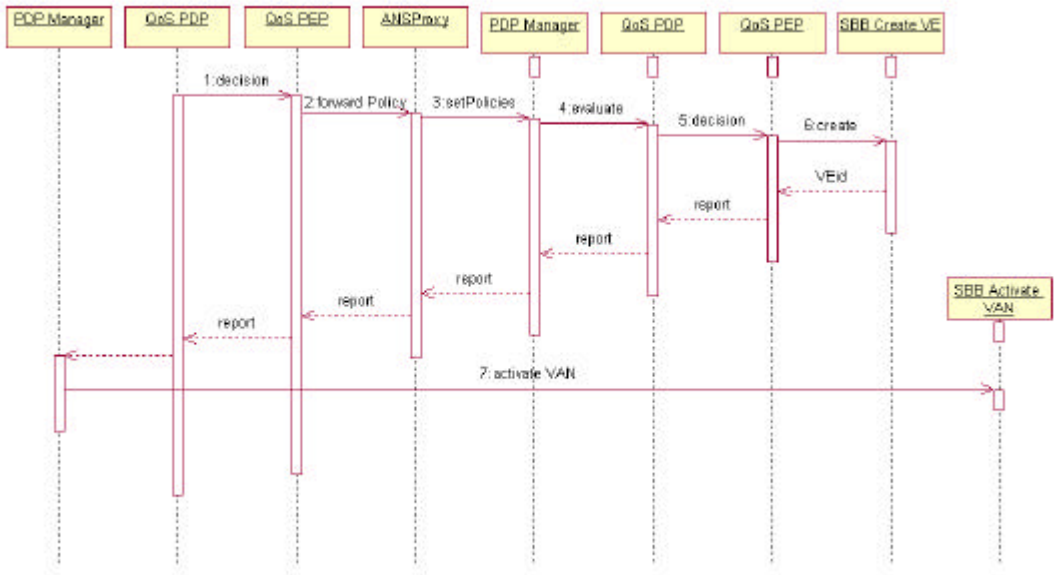
The NMS, more exactly, the resource manager knows the mapping between VAN Identifier and VE Identifier on all ANs of topology.

Dependencies

This building block depends on

- SBB Install Service Component onto a node (from CS3)
- SBB Install Service Component onto management station (to be discussed)
- SBB Boot EMS
- SBB Boot Node
- SBB Create VE

Sequence Diagram



Done at NMS:

1. (See step 7 of the last sequence diagram) with the information about which nodes are involved the QoS PDP forwards the policy to the QoSPEP that will translate it into the corresponding QoS EL policy

Done at EMS:

2. Send this policy into the EMS of the established nodes.
3. The QoS Policy received by the ANSP Proxy bases on the user information and is demultiplexed to the policy adequate to the MI, in this case the ANSPs MI.
4. Inside the MI, the PDP Manager receives the policy, stores it in the DB, and requests the access rights check for that policy and user. In case of positive answer, the PDP Manager demultiplexed the policy to the QoS PDP.
5. The QoS PDP retrieves the policy from the DB, detects that the policy should be enforced immediately, and for that reason forwards the decision about it to the PEP component which is running inside the ANSPs VE (the privileged VE) of the active node.
6. The QoS PEP enforces the decision, creation of a VE, allocating resources required. To do that it use the API offered by the ANN (Use of SBB Create VE). The enforcement result is sent back through involved components to the NMS.
7. Here the SBB Active VAN starts.

Implementation Status

Done.

There is still one open issue to be discussed it's about interaction between ASP and the management system, how is management system going to contact with ASP in order to download new service. A service which can be only a jar package with the code needed for interpret new policies, or a new domain PDP, PEP. May be a first solution could be, only to download this code into the management station in a well known directory, and one of the components used by the Domain Manager will be in charge of, using the Class Loader, loads the class and creates a new instance. This component also will be in charge of lifecycle of this code, I mean, release, delete, uninstall, and withdraw an instance.

Subcomponents

- At NMS:
  - QoS PEP at NL
- At EMS:
  - ANSP Proxy
  - PDP Manager
  - QoS PDP
- At ANN:
  - QoS PEP

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
QoS PEP at NL	Translate QoS NL Policies into QoS EL Policies.		D5
ANSP Proxy	See comments above	See comments above	D5

PDP Manager	See comments above	See comments above	D5
QoS PDP	Evaluate and Make Decision	Subscribe specific events on monitoring system, scheduler Process incoming event from monitoring system, scheduler	
QoS PEP	Enforce Decision using SBB Create VE	Identify correct parameters to create the resource profile (What RCF can do?) Add code in order to support signaling request	D5

SBB Activate VAN

This building block describes how the ANSP activates all appropriate new VE instances, which have just been created. At the time of activation, the ANSP will also assign enough access rights for the SP in the VE created to him. That is, ANSP will delimit the management functionality offered by the Active Network Node to the SP VE in order to guarantee isolation of the management functionality offered to that Service Provider. Therefore, the SP will not be able to manage other SP's functionality, and that others SP's will not be able to manage the functionality owned by this particular SP.

Pre-conditions

All of appropriate VE are created and the ANSP holds a reference of them.

PEP Service is registered on the ASP (Delegation Instance)

PDP Service is registered on the ASP (Delegation PDP)

The Policy Forwarder Controller at NMS knows the status about the creation of VE Network, and it has been successful.

Post-conditions

A VE Network has been activated and from now on the SP is allowed to use it.

A report with the result of VE activation is forwarded back to the NMS station through the involved components.

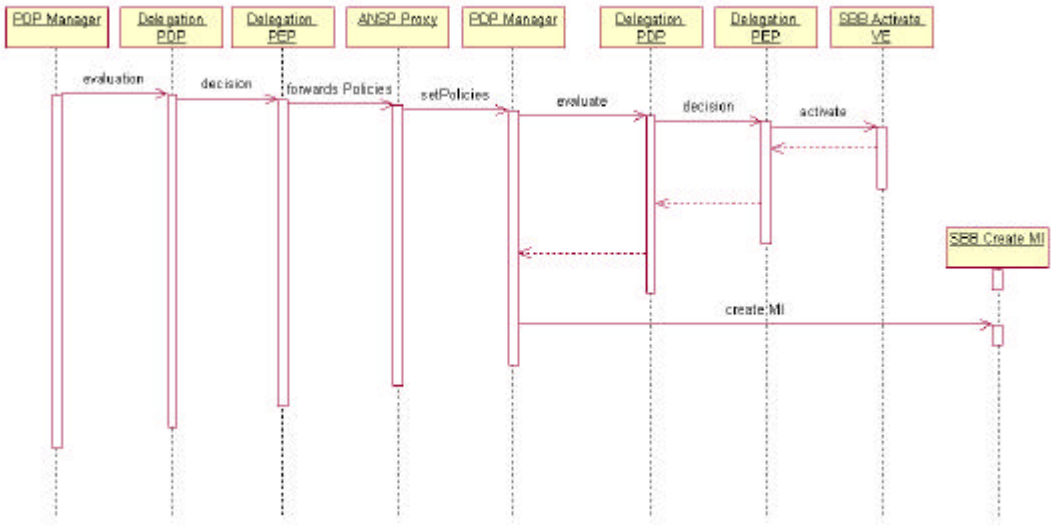
Dependencies

This building block depends on

- SBB Install Service Component unto a node (from CS3)
- SBB Install Service Component unto management station
- SBB Create VAN
- SBB Activate VE

Sequence Diagram

This scenario starts after the enforcement result of the QoS NL Policy arrives to the PDP Manager.



1. Since the enforcement of the QoS Policies has been positive in all nodes, the PDP Manager starts the processing of the delegation policies requesting the check of the access rights for that policy and user. After the positive reply the policy is forwarded to the Delegation PDP to be evaluated.
2. After Delegation PDP retrieves it from the DB and there are no conditions or actions to be processed at the network level, forwards it to the PEP. Before, it contacts with the Resource Manager in order to know which nodes are involved.
3. The Delegation PEP will translate the Network level into two element level delegation policies: one for the assignation of access rights to the VE and activate it (VE activation). And the second for the instantiation of the MI inside the EMS so that the SP can manage its resources. (Create MI).
   A policy set is created with them and submitted to the respective EMSs.
4. The Delegation policy group is received, again by the ANSP Proxy, which forwards it to the PDPManager of the ANSP MI
5. The PDPManager receives the policy group, stores it in the DB, and starts the processing of the policies that conform to that group. It detects that the policies should be enforced in order and that the next delegation policy should only be processed if the processing of the first delegation policy is correct. The first delegation policy in the one that causes the assignation of access rights for the SP in the VE created to him, and at the same time, the activation of this VE.

   After this policy is checked in the Access Rights Check component is dispatched to the Delegation PDP to be evaluated.
6. The Delegation PDP detects it should be enforced immediately and forwards it to the Delegation PEP running inside the privileged VE in the ANN
7. The Delegation PEP enforces the policy activating the VE using the API offered by ANN.: (uses SBB Activate VE to activate the VE).

   The result of the VE activation is forwarded back to the PDP Manager through the involved components.
8. Here SBB Create MI starts.

Implementation Status

Done.

Subcomponents

- At NMS
  - QoS PEP at NL
- At EMS
  - ANSP Proxy
  - PDP Manager
  - Delegation PDP
- At AN
  - Delegation PEP

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
Delegation PDP at NMS	Evaluate and make decision		D5
Delegation PEP	Translate Delegation NL Policies into Delegation EL Policies.		D5
ANSP Proxy	See comments above	See comments above	D5
PDP Manager	See comments above	See comments above	D5
Delegation PDP	Evaluate and Make Decision	Subscribe specific events on monitoring system, scheduler Process incoming event from monitoring system, scheduler	D5
QoS PEP at ANN	Enforce Decision using SBB Activate VE	Identify parameters to set up correctly security framework.	D5

SBB Create MI

This building block describes how the ANSP configures all of element management stations responsible of the involved nodes, those on which the ANSP has already activated a VE for a SP, allowing to the SP to manage its resources.

Pre-conditions

VE Network has already been created and NMS knows that.

QoS PEP component which is located on the EMS is running.

Post-conditions

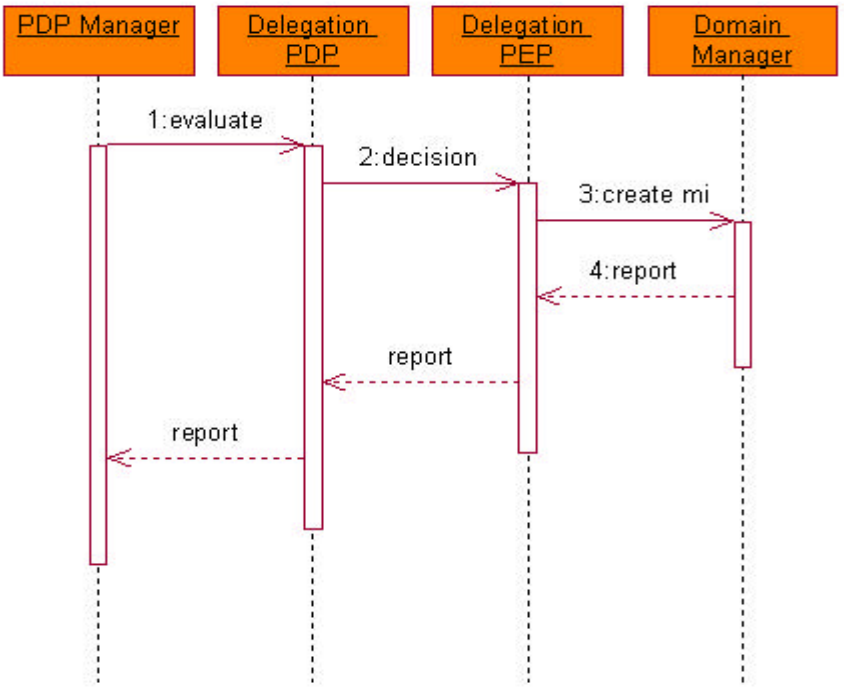
A new "management instance" for this SP is created, where part of the functionality of the ANSP, i.e. the PDP Manager is instantiated there as well as access to the monitoring system services. The PDPMgr will allow the SP to dynamically extend its functionality on runtime in an automatic manner. A report about the enforcement result is forwarded back to the NMS.

Dependencies

This building block depends on

- SBB Activate VAN
- SBB Install Service Component onto management station

Sequence Diagram



1. Since the enforcement of the first delegation policy has been positive, the PDP Manager starts the processing of the second delegation policy that is, the policy is checked in the Access Rights Check component and forwarded to the Delegation PDP.
2. Again, Delegation PDP detects it should be enforced immediately, thus it is forwarded to the correct PEP. This PEP is running inside the management station since the configuration actions should be realized here.
3. The enforcement of the policy causes the creation of a new Management instance, where part of the functionality of the ANSP, i.e. the PDP Manager is instantiated there as well as access to the monitoring services. The PDP Manager will allow the SP to dynamically extend its functionality on runtime in an automatic manner.

The enforcement result is forwarded back to the NMS.

Implementation

Done.

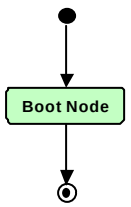
Subcomponents

- At NMS
  - QoS PEP at NL
- At EMS
  - ANSP Proxy
  - PDP Manager
  - Dlg PDP
  - ARC
  - Dlg PEP

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
PDP Manager	See comments above	See comments above	D5
Delegation PDP	Evaluate and Make Decision	Subscribe specific events on monitoring system, scheduler Process incoming event from monitoring system, scheduler	D5
QoS PEP at EMS	Enforces decision setting up EMS. (ARC-Domain Manager) It creates a management instance for SP.		D5

SBB Boot Node

This building block describes how an active node is booted. Bootstrapping the privileged VN (owned by the active network service provider, ANSP) is a special process. Because the VE management infrastructure isn't available at this point one has to rely on operating system support. For example, scripts for starting the privileged VE can be included in the operating system's boot procedure.



Pre-conditions

The software needed for the VE management has to be installed

- JAVA virtual machine
- VE management distribution

The appropriate start script has to be linked to the boot procedure.

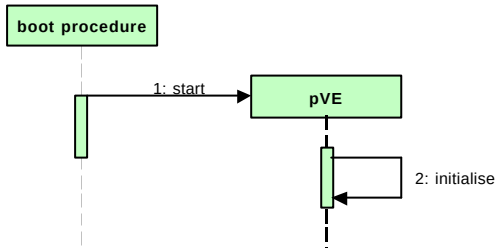
Post-conditions

The privileged VE is running and the reference to its initial port (*iComponentInitial*) can be obtained from a well known TCP port. This building block is parameterized with the node to contact.

Dependencies

None.

Sequence Diagram



1. The boot procedure starts the privileged VE (pVE) via a script.
2. The pVE initializes itself by loading the basic services (VE/EE management, security, demultiplexing, traffic control, etc.). Finally the reference to the pVE's initial port (*iComponentInitial*) is made available at a well known TCP port.

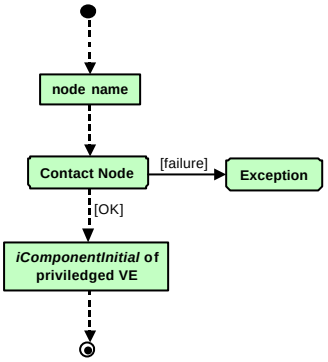
Implementation Status

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
start script for privileged VE	Starting the privileged VE via script works fine.	Start script doesn't automatically run at boot-time.	OS Manual

SBB Contact Node

This building block describes how an active node is contacted by a client. This involves getting the reference to the privileged VE's initial port.

This building block is parameterized with the node's name.



Pre-conditions

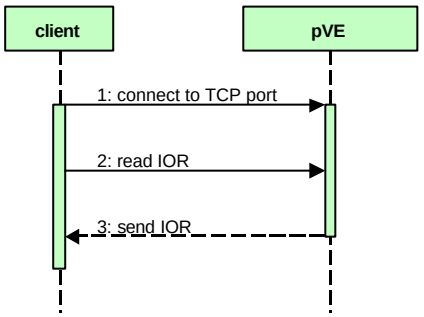
The privileged VE is running.

Post-conditions

The client holds a reference to the initial port of the node's privileged VE.

Dependencies

None.



Sequence Diagram

1. The client connects to a well known TCP port of the privileged VE (pVE).
2. The client reads the reference to the pVE's initial port from the TCP connection.
3. The pVE sends the reference.

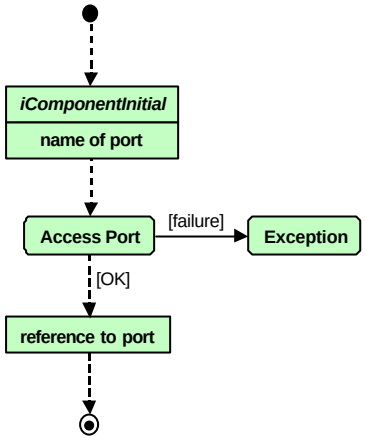
Implementation Status

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
privileged VE	A client can obtain the reference to the initial port of the privileged VE from a well known TCP port.	None.	None.

SBB Access Port

This building block describes how a port of a component is accessed by a client. This involves the authentication of the client.

This building block is parameterized with the component in question and the requested port.



Pre-conditions

The client holds a reference to the component's initial port.

The client uses a valid port name.

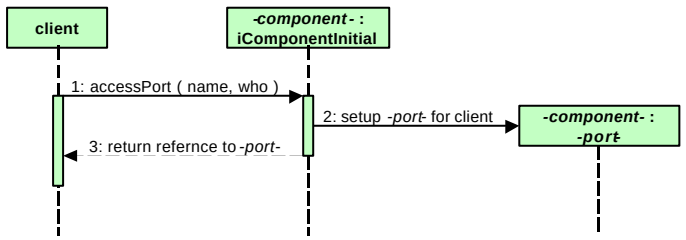
Post-conditions

The client holds a reference to the requested port.

Dependencies

None.

Sequence Diagram



1. The client requests access to a specific port at the component's initial port by specifying the port's name and the client's identity.
2. After checking the access rights of the client with the security context (not shown here) the component sets up the requested port for the client.
3. The component returns a reference to the requested port to the client.

Implementation Status

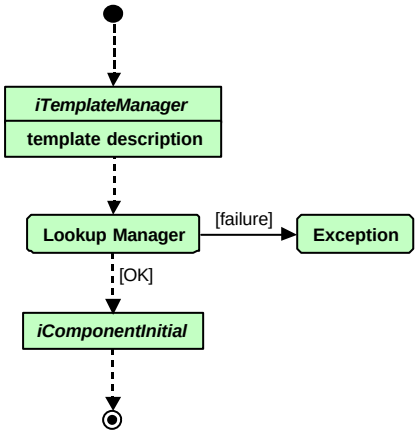
Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
iComponentInitial implementation in Basic Component	A client can request access to a port by specifying the port name and the client's identity	The check for admittance is performed but not yet linked to the security component. The security	D4

	and the client's identity. After checking for admittance the component framework will create a new client-specific reference to the requested port. When a particular reference will later be addressed it will be mapped to the original client. Note that if a client passes a reference to a third party there is no way to distinguish it from the original client. A client can explicitly stop using a port so that the component framework will invalidate the reference. Otherwise the reference stays valid until the component terminates. If a client requests access to the same port multiple times it will always receive the same reference.	component. The security component will have to maintain security policies in order to accept or deny requests for port access.	

SBB Lookup Manager

This building block describes how a manager for a particular service (i.e. resource) is looked up by a client. The client has to provide a description of the service's template. The result will be a list of identifiers of matching managers. The client can then use an identifier to request a particular manager's initial port.

This building block is parameterized with the environment in question (VE/EE) and the template description.



Pre-conditions

The client holds a reference to the *iTemplateManager* port of the appropriate VE or EE, usually the privileged VE.

The client has set up a valid template description.

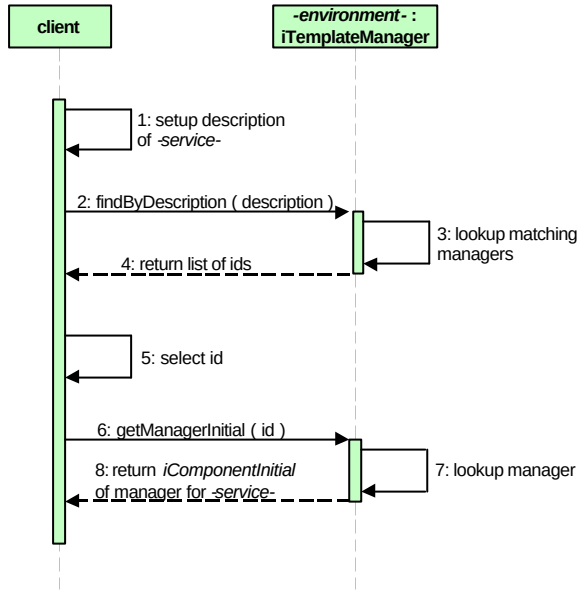
Post-conditions

The client holds a reference to the initial port of a matching manager.

Dependencies

None.

Sequence Diagram



1. The client sets up a description of the desired services. The description includes the service name, version, and other fields. Fields other than the service name can be left empty if not of interest.
2. The client requests to lookup managers for services matching the specified description.
3. In the case of a virtual environment the VE searches its attached EEs, in the case of an execution environment the EE checks its internal tables and additionally looks for a possibly running VE manager to search its VE instances, too. With this recursive approach it is possible to search an entire tree of VEs and EEs starting at the privileged VE which is the tree's root.
4. The environment returns a list of identifiers of matching service managers.
5. The client selects one identifier.
6. The client requests the initial port of a manager specifying its identifier.
7. The environment looks up the appropriate manager. This is again done recursively whereas caching of identifiers and references to managers during step 3 can be used to speed up the process.

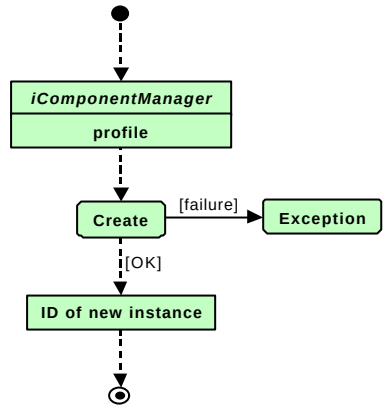
Implementation Status

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
iTemplateManager implementation in Virtual Environment and (JAVA)Execution Environment	The process of searching through a hierarchy of VEs and EEs works in general.	In the case of a failure for a particular VE or EE in the hierarchy the whole process might get spoiled. This has to be made more robust.	D4

SBB Create Instance

This building block describes how a client can create a component instance of a particular service. For the creation the client can specify a resource profile which will be examined by the manager to check the availability of resources.

This building block is parameterized with the manager which is used to create a new instance and the corresponding profile.



Pre-conditions

The client holds a reference to the manager's *iComponentManager* port.

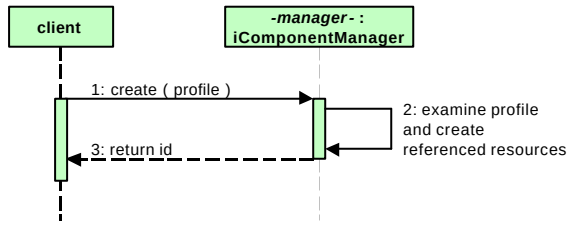
Post-conditions

The client holds the identifier of the new instance.

Dependencies

None.

Sequence Diagram



1. The client requests the creation of a new component instance and passes a resource profile as parameter. The content of the profile is specific to the kind of manager.
2. The manager examines the profile and tries to create (pre-allocate) all requested resources.
3. The manager returns the new instance's identifier to the client.

Implementation Status

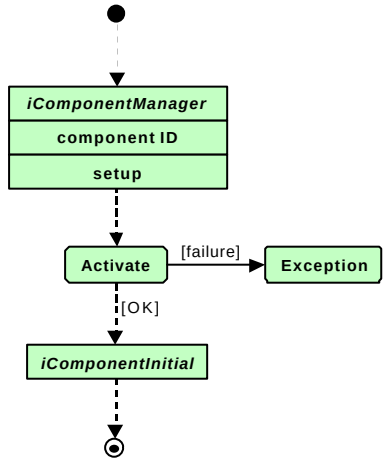
Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
iComponentManager implementation in Component Manager and various sub-classes, i.e. Channel Manager,	Though the component framework provides the management of instances and keeps track of their profiles and their states	Other managers.	D4

Security Manager, Traffic Manager, Virtual Environment Manager, (JAVA) Execution Environment Manager	(created, activated) it is the responsibility of the specific manager to implement the examination of the profile, the resource admission control, and the instantiation and deletion of actual objects. This is done for the basic services: VE manager, EE manager, demultiplexing, and traffic control.		

SBB Activate Instance

This building block describes how a client can activate a component instance of a particular service. For the activation the client can specify an initial setup for the component instance. After the activation the client can obtain a reference to the instance's initial port.

This building block is parameterized with the manager which is used to activate the instance, its identifier, and the initial setup.



Pre-conditions

The client holds a reference to the manager's *iComponentManager* port.

The client knows the instance's identifier.

The client uses valid setup parameters.

Post-conditions

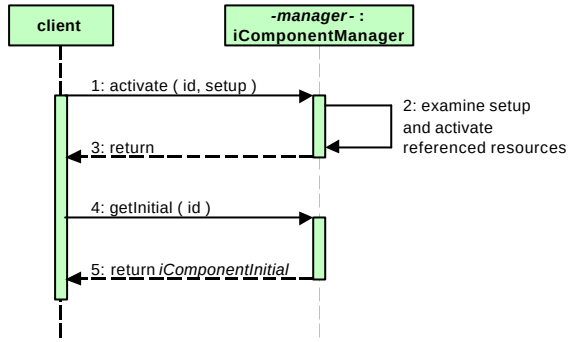
The instance is activated.

The client holds a reference to the instance's *iComponentInitial* port.

Dependencies

None.

Sequence Diagram



1. The client requests to activate the new instance and passes an initial setup as parameter.
2. The manager examines the initial setup and activates the previously created resources. The separation of creation and activation allows for a client first to create a number of placeholder instances and only in the case of a full success to activate them.
3. The activation is done.
4. The client requests a reference to the initial port of the new instance.
5. The manager returns the reference to the initial port.

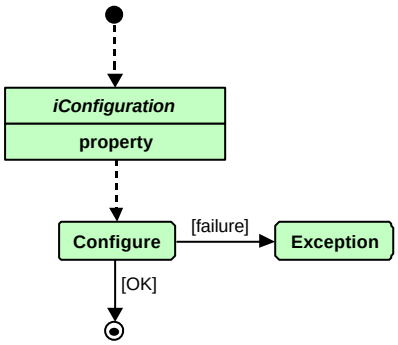
Implementation Status

See implementation comments for "SBB Create".

SBB Configure Instance

This building block describes how a client can configure a component instance. Configuration is done by setting properties of the component where a property is a pair of a name and a value.

This building block is parameterized with the component in question and the property.



Pre-conditions

The client holds a reference to the initial port of a component.

The component supports the *iConfiguration* port.

The client uses a property valid in the context of the component.

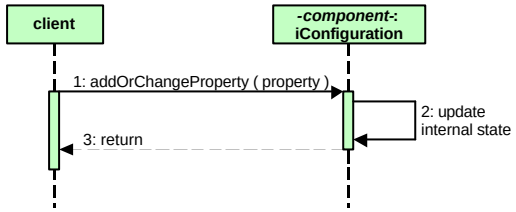
Post-conditions

The component is configured.

Dependencies

None.

Sequence Diagram



1. The client requests to add a new or change an existing property.
2. The component updates its internal state accordingly.
3. The configuration is done.

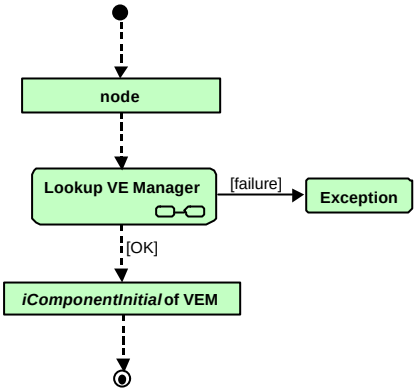
Implementation Status

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
iConfiguration implementation in Configurable Component	The component framework provides support for "active" properties – i.e. properties that fire an event when they change – through the generic configurable component. Subclasses can override property related methods to implement specific behavior.	None.	D4

SBB Lookup VE Manager

This building block describes how a client is able to lookup the VE manager on a node. It is composed by several other SBBs.

This building block is parameterized with the node's name.



Pre-conditions

None.

Post-conditions

The *iComponentManager* port of the VE manager is returned.

Dependencies

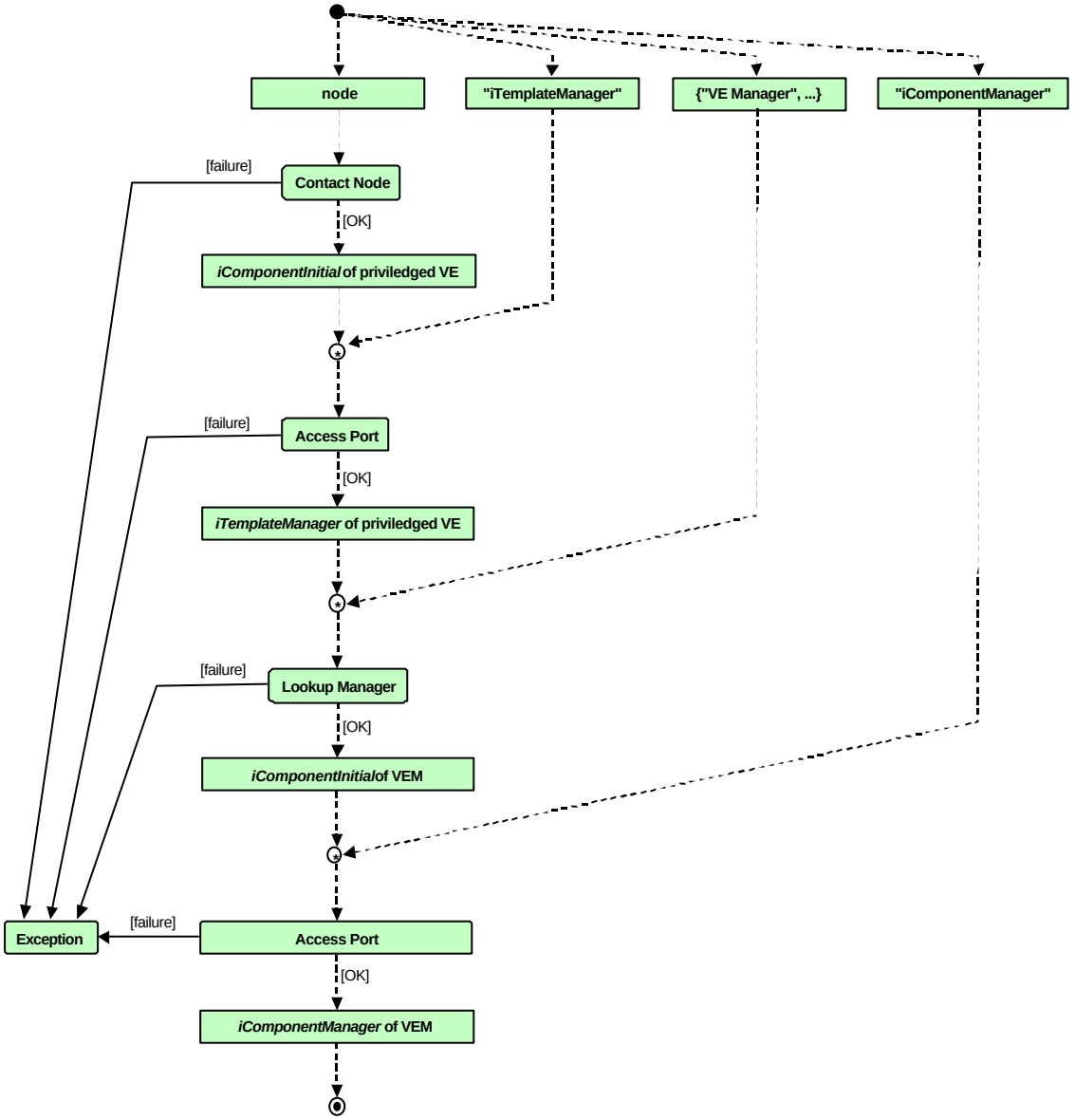
This building block depends on

SBB Contact Node

SBB Access Port

SBB Lookup Manager

Sequence Diagram



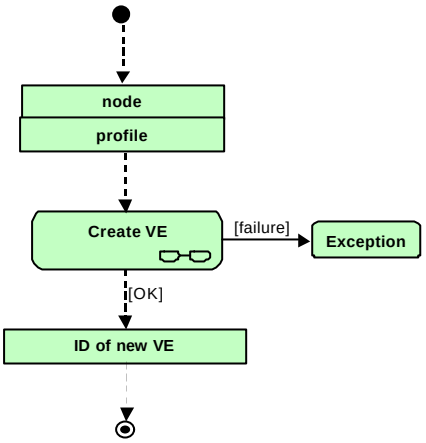
Implementation Status

See implementation comments for SBB Contact Node, SBB Access Port, and SBB Lookup Manager.

SBB Create VE

This building block 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.



Pre-conditions

None.

Post-Conditions

The VE is created.

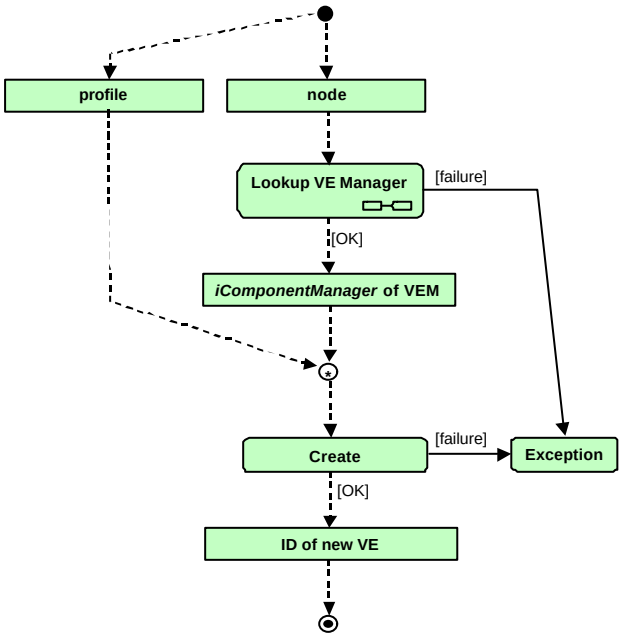
Dependencies

This building block depends on

SBB Lookup VE Manager,

SBB Create.

Sequence Diagram



Implementation Status

See implementation comments for SBB Lookup VE Manager and SBB Create.

SBB Activate VE

This building block describes how a VE is activated on a node. It is composed by several other SBBs.

This building block is parameterized with 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.

Pre-conditions

The VE was created before.

Post-conditions

The VE is activated.

Dependencies

This building block depends on

SBB Lookup VE Manager

SBB Activate

Implementation Status

See implementation comments for SBB Lookup VE Manager and SBB Activate.

4.1.2 CS 2: Flow and Data Path Creation for Service and User Communication

This section introduces the first ideas on the interactions required in order to achieve the data path creation scenario. It is focused on the interactions occurring at the network level mainly involving the management system and the ASP, although eventually lead to node and node OS interactions. The scenario building block approach has been used in order to simplify the descriptions, and reuse the defined blocks in different core scenarios. The interactions have been categorized as pertaining to the configuration, the data path creation or data flow creation phase.

Configuration

A data path establishment requires a previous definition of the restrictions that apply to the resources involved in its creation. Such definition is usually performed as part of a service level agreement carried out between a SP and an ANSP. The translation of these requirements into actual network configuration is a mandatory step that must be fulfilled before any request is accepted.

The configuration process affecting the resources used along a data path involves several nodes and should be therefore supervised and controlled at the network level management system. This process is part of the initial VE configuration and has effect as long as the virtual network exists (unless we consider a dynamic reconfiguration is possible).

Note that although this process does not actually create any data path, is a prerequisite to its creation. The restrictions that apply to any data path should be specified in terms of the bandwidth assigned to each available link and the existent topology paths (that is, the restrictions that apply to connections with other nodes of the network)

Data Path Creation

An analysis of the situations that imply the creation of a data path make us come to the conclusion that either the elements in charge of the service creation and the services themselves may request its establishment.

In both cases, knowledge of the whole network status is required in order to select the path that better meets the network management requirements. The network management system is in the best position to provide this “network wide knowledge” and therefore, any request should be delivered to it. It also makes sense to make the best of the delegation framework in order to provide the required scalability in this approach.

The network level PDPs (in the best located management instance), cooperating with the RM and the network level ASP, would perform the decision on the paths connecting the active nodes. They would send the configuration policies to the network level PEPs, which in turn would be in charge to translate them into data flow creation policies.

Data Flow Creation

The “instantiation” of a data path is supported by the creation of appropriate data flows between adjacent nodes. This is an element level operation that is an element level responsibility. The existence of privileged EE ports allowing the entrance of configuration information is a major requirement.

Although the PDPs at the element level make the decision on the final data flow parameters based on information retrieved from the node, it is the element level PEP who actually performs the method call to the EE (no request to the privileged VE is required since the resources were already assigned in the previous phase).

It is the responsibility of the EE management component to perform the required configuration of internal node components, such as the multiplexer, through the available interfaces.

SBB Data Path Creation

This SBB 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 an *unidirectional stream of packets that have some common attribute(s) (such as source, destination, or protocol* **[1]**). 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.

In order to simplify the description of the SBB, the scenario has been decomposed into the sub-SBBs that are shown in Figure 4-3.

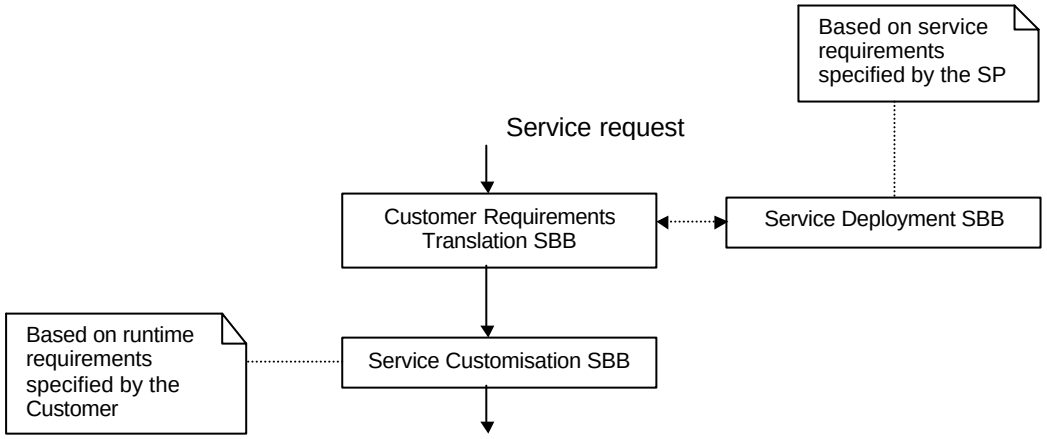


Figure 4-3: Data Path Creation SBB decomposition.

The diagram shows the SBB is initiated by a customer service request, and leads to the establishment of a customized data path in the active network that provides the communication and processing capabilities expected by the user. The contribution of the proposed decomposition is the separation between the service deployment and service customization blocks. Such distinction results in an easier approach to describing service configuration based on different user profiles[1].

Under a TINA perspective, this SBB can be considered as part of the service access phase, preparing the infrastructures required to render the service during the following service usage part.

Pre-conditions

- The Virtual Network associated to the SP is already created and configured.
- Service is realized, i.e. service descriptors and service components are ready to be retrieved from service registry, and service repository, respectively. However, there is no need to actually deploy the service before a user requests it.
- The service entry point is configured and already running. The configuration information includes the location where service requests should be forwarded. In case the service provider possesses his own management instance, such instance would become the recipient of service requests.

[1] In practice we are approaching the problem of rendering the service to several customers, using the same service components with different quality and resource requirements.

- In case the SP manages his own virtual network, appropriate delegation policies have been already deployed.
- A data path establishment requires a previous definition of the restrictions that apply to the resources involved in its creation. Such definition is usually performed as part of a service level agreement carried out between a customer and a service provider. The translation of these requirements into actual network configuration is a mandatory step that must be fulfilled before any request is accepted.

Post-conditions

- Service components have been instantiated on appropriate locations along the active network and have been bound appropriately.
- Appropriate data flows have been set up.
- During the data path lifetime, the information coming on a data flow is identified and assigned to a specific data path, so that it receives de local service and is sent to the new target component.
- The mechanisms to survey the assigned resources have been set up. This surveillance is realized based on the QoS policies previously established. The management system should take over the responsibility of this mission, which might be delegated to the appropriate management instance due to scalability reasons.

Dependency

The data path creation SBB relies mainly in three SBBs that are performed sequentially, namely:

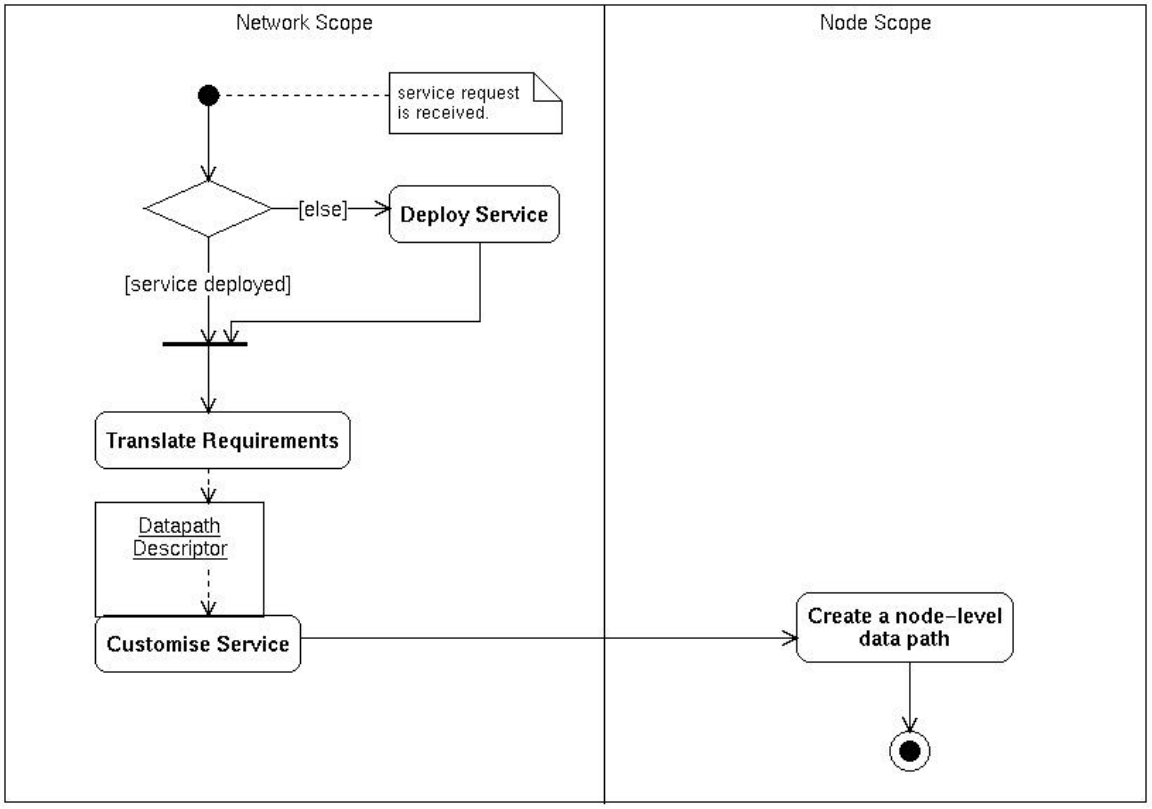
- The service deployment SBB, which is only accessed in case the service, has not been deployed yet. This SBB is skipped when the service already exists in the network, for instance due to a previous service request. The service deployment is performed based on the requirements specified by the service provider in the service descriptor. Its purpose is ensuring the presence of every service related component in the network.
- The customer requirement translation SBB that includes the description of dynamically obtaining the concrete data path QoS requirements from the customer service level expectations.
- The service customization SBB, that comprises the service component deployment SBB and data flow creation SBB, which are nested SBBs, and may also include additional configuration operations.

Required Functionality of Involved Subcomponents

- The PDPs and PEPs should provide a transactional deployment of policies.
- The SCE should be able to provide new graphs dynamically, based on changing requirements.
- The portal should turn into a complete service access point. Further cooperation with the management system is required in order to control service request admission.

Activity Diagram

This section presents a diagram that depicts the dependencies and coordination between the different scenarios building blocks involved in the data path creation. It should be noted that this diagram approaches the out-band data path configuration mechanism.



Sequence Diagram

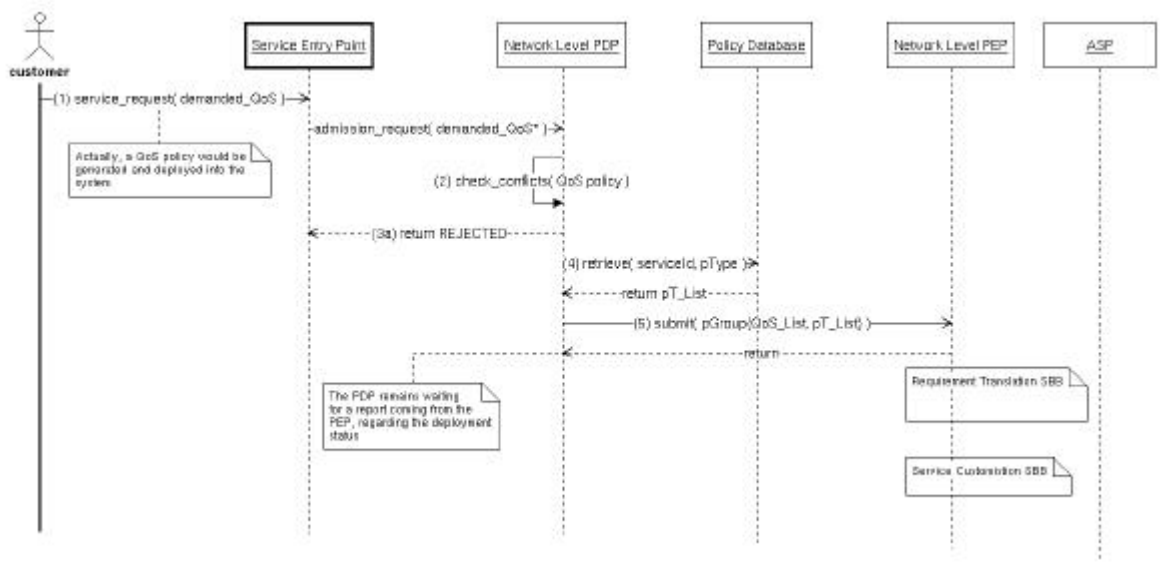
Whenever a service request arrives to a service entry point, it is adapted and forwarded to the specified service provider management instance (1). The QoS PDP[2] at the network level is requested to make a decision on the request admission. It is the PDP responsibility to ensure the existence of an appropriate data path before admitting the request.

As an initial step, the PDP checks for conflicts with the existing QoS policies[3] deployed by the service provider (2). If a conflict is found, the request is rejected (3a). Otherwise, the PDP retrieves all the translation rules available for the specified service (4) and submits them together with the policy to its subordinated PEPs (4), which in turn perform the requirement translation process assisted by the resource manager and network ASP. As a result, the requirements for each of the data path elements are obtained. At this time, such requirements are hold in a data path descriptor.

The data path descriptor is delivered to the service customization SBB as part of its input information (5). According to it, this SBB will locate and enable appropriate resources, being also responsible for initiating the node-level data path creation SBB.

[2] Unless explicitly said, we will use just the term PDP when referring to the QoS PDP.

[3] Originated from the SLA.



Implementation Status of Involved Subcomponents

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
- Network Level PDP		Conflict Checking for request admission.	
- Policy Database	Retrieval of policies.	Query-based searching capabilities.	
- Management System		Appropriate QoS policies for the transcoder service have to be defined. Define the scope of the SLA for the transcoder service.	

SBB Requirement Translation

The purpose of this SBB is to obtain the requirements associated to the data path elements (either service points or connections between them), from the service needs specified by the customer. These requirements will be included in a data path descriptor.

The data path descriptor represents a complex structure that provides all the required information to instantiate and configure the data path elements. Part of the information contained in the structure can be generated from static sources such as the service descriptor, which defines the required service components, the order in which these components should be traversed, and the customer service needs. The rest of the description can only be obtained at runtime, based on the resource availability, and is therefore out of the scope of this SBB[4].

[4] The process of obtaining the runtime information is further described in the Service Customisation SBB.

Therefore, the data path definition is performed in two steps: initially, static information is obtained from the service quality requirements. Then, this information is compared against runtime restrictions to obtain the actual data path definition. This SBB is centered in the first phase, being in charge to adapt or translate the high level service requirements into low-level, component oriented requirements.

The static information that can be directly obtained from the service descriptor includes:

- Data path Graph type.
- List of component types that form the data path.
- In the case of a sequential data path graph, the order in which each type of component should be traversed. For example, each service point would specify the following one.
- In the case of an arbitrary data path, for each service point a set of possible following service points should be specified.

Additionally, we would need to obtain the particular QoS parameters that are applicable to the links between components and the components themselves. These parameters would be translated from the overall QoS expected by the customer, following a list of rules defined for each type of element. The QoS policies defined by the SP are checked to ensure the acceptability of the service request.

The requirement translation is be performed in two phases: in the first phase, the initial requirements are distributed between the different resources according to the service specification. The result is a list of resources which are associated to the considered parameter. Each of the resources is assigned a relevant parameter value[5]. In the second phase, each parameter is translated into actual configuration values. For example, the considered bandwidth may imply different buffer lengths or queue priorities in the node.

The resulting QoS policies are specified at different levels of abstraction, ranging from the element level policies to the parameter-value configuration list sent to the devices.

As it is explained in the sequence diagram section, the generation of QoS policies is a required step to ensure the correctness of the data path descriptor and is closely interlaced with its production.

Pre-conditions

- The policies generated from the *customer-service provider* SLA have been deployed into the management system.
- Appropriate requirement translation rules have been defined for the relevant parameters on the service elements.
- The requested service level fits to the SLA. The service provider management instance performs this initial process by checking for conflicts with the currently deployed QoS policies. Note that this might be not enough to make a final decision on the admission process. Such process is successful only when an appropriate data path has been actually found.

Post conditions

- The QoS and service configuration policies have been generated from the service requirements by the management system.
- The dynamic part of the data path descriptor, containing the resource requirements has been fulfilled.

[5] For example, a thread priority may be influenced by bandwidth demanded by the client.

Required Functionality of Involved Subcomponents

- The Network Level PEPs should provide a translation engine able to use translation rules on a per-service base.
- A concrete protocol for MS-ASP interaction should be designed.
- The ASP should be able to request topological information to the management system, whereas the management system should offer an interface to retrieve this information.

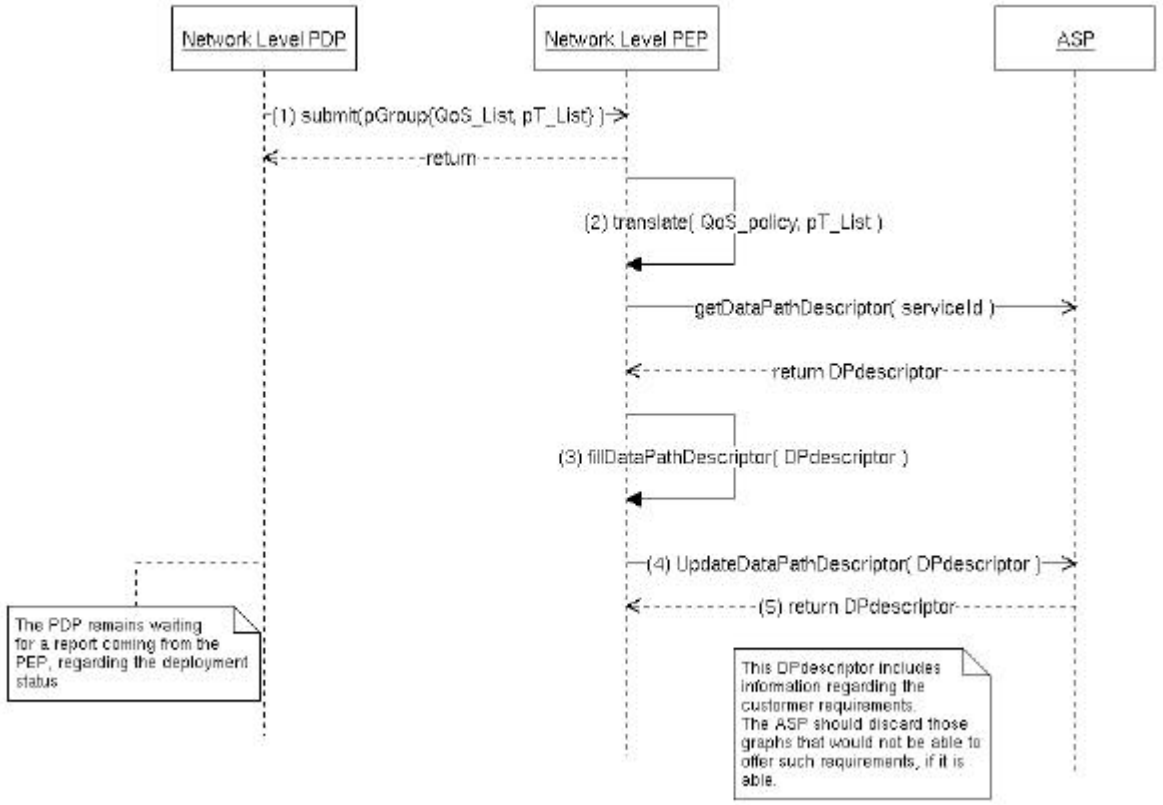
Sequence Diagram

During the requirement translation process, the network level QoS PEP receives the translation rules available for an specified service (1) and uses its translation engine to create the element level QoS policies and the dynamic part of the data path descriptor that refers to the service requirements, based on the requested QoS level (2).

The network level QoS PEP is also in charge of distributing the element level policies to the appropriate locations. In order to find such locations, the network level PEP makes use of the static part of the data path descriptor, which is obtained from the ASP under demand (3). In this process, the PEP includes the previously obtained requirement information into the data path descriptor and returns it to the network level ASP (4), which should be able to fulfill the information regarding the component location within the network[6] (5). Note that this information is not valid yet, since the ASP can not be sure that the currently deployed service components and the links between them can actually offer the requested service level.

The validation of this data path descriptor by the management system is essential before acknowledging the correctness of the data path graph. The status flag related to each service element listed in the data path descriptor should be marked as “in progress”.

[6] The way the ASP obtains the location information is out of the scope of the data path creation SBB. It should be described as part of the service deployment SBB.



Implementation Status of Involved Subcomponents

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
Network level PEP	policy translation	Dynamic policy translation at the network level.	
Network level ASP		Storage of updated service topology. Protocol to request topological information to the management system. Topology description format.	
MS-ASP		Concrete structure of the data path descriptor. interface to exchange data with the MS.	

SBB Service Customization

The purpose of this SBB is to locate and enable appropriate resources according to the data path descriptor. Figure 4-4 shows the SBBs that will be considered as part of this block. The adaptation of the service to the customer needs, received in the form of data path requirements may involve the deployment of service components in new locations of the network and always results in the assignment of resources to the customer.

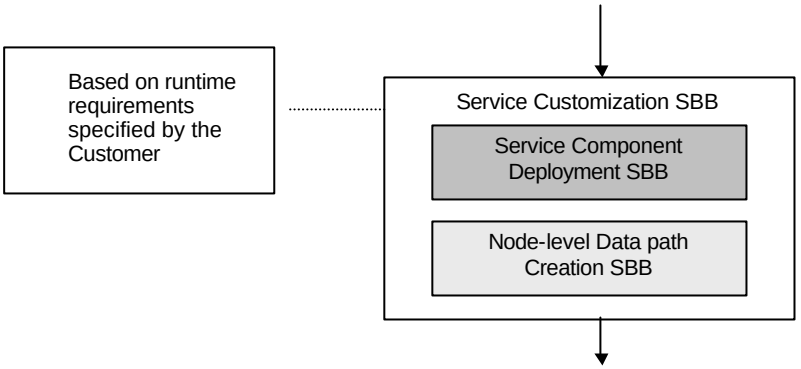


Figure 4-4: Service Customization SBB.

Additionally, the SBB should also define the runtime parameters of the data path descriptor. This process will be performed in different ways depending on the data path graph type. The general case is that in which each service point and the order they are traversed is known in advance and remains fixed during the service lifetime. A particular case occurs when the next service point is decided depending on the result of the data processing. In this situation, a set of available service points might be defined for each of the service points themselves.

The service customization is a progressive process. We can assume that the restrictions (in the form of static information included in the data path descriptor) will guide the process of locating the best service components and routes available. The produced data would be included in the descriptor as runtime fields (for example, the data flow descriptors).

When it was not possible to locate a component that fulfills the QoS requirements, the deployment of specific service components on new locations may be ordered[7]. As part of the deployment decisions, an assessment of the interconnection resources is required, implying knowledge of the network status[8].

As a result of the translation process and the subsequent customization process, a set of QoS policies guaranteeing the acceptable quality levels are also deployed in the management system.

Pre-conditions

- The service is already deployed in the active network.
- The low level requirements for each service element have been defined, based on user needs.

Post-conditions

- There exists a data path that fulfils the customer expected service level.
- Per-customer service-related QoS policies have been deployed into the management system.

Dependencies

- Deployment and instantiation of service components.
- Node-level data path creation SBB

Sequence Diagram

[7] If supported by the ASP, the reconfiguration of the service may be also requested.

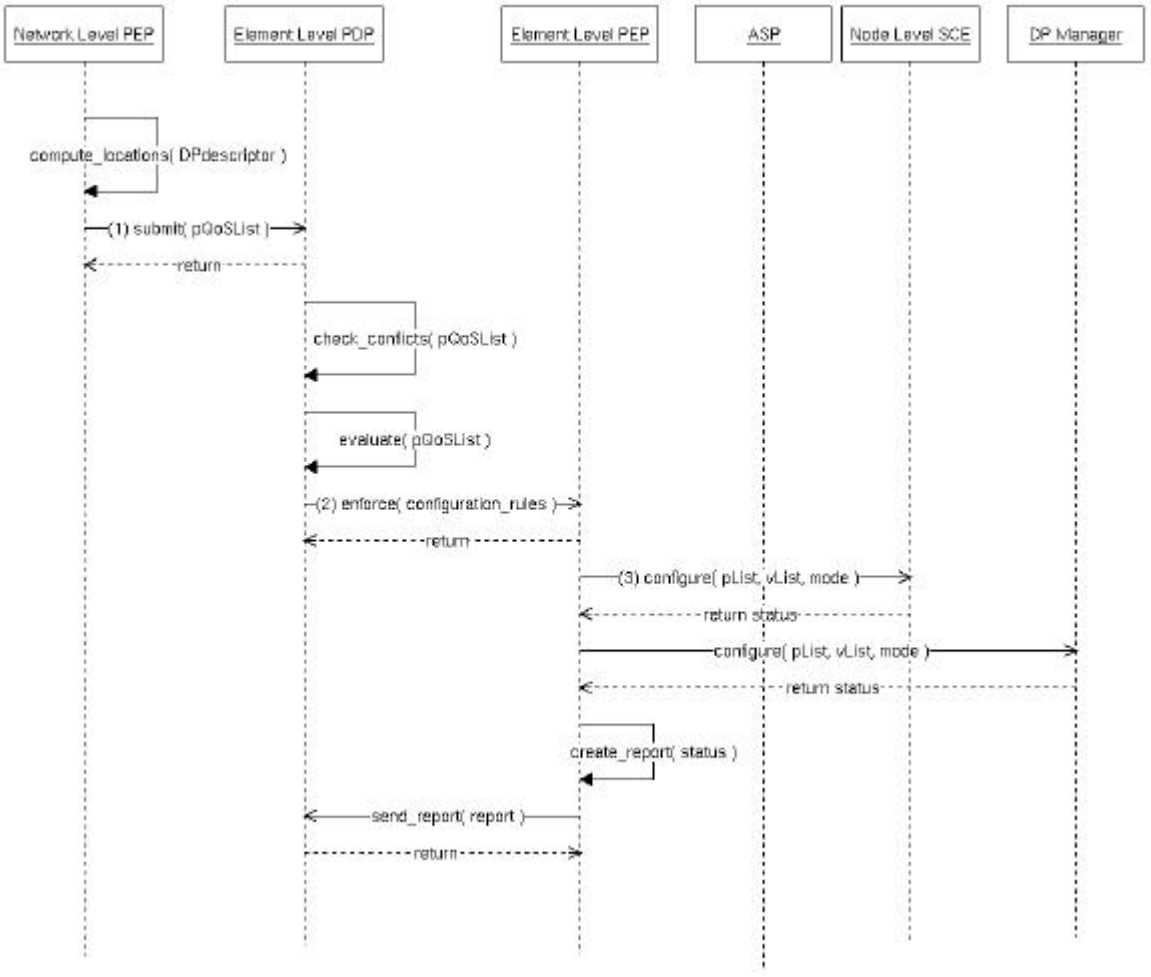
[8] The ASP and NMS would collaborate in order to solve this situation, which is part of the service component deployment SBB.

Based on the information provided by the network level ASP, the QoS PEP at the network level selects the location of the element level PDPs to whom it should deliver the obtained QoS policies. The policies are then delivered to these locations, where the service components are running (7), in a transactional deployment mode.

The transactional deployment of policies is a two-phase mechanism for ensuring policy consistency across a set of remote locations in a distributed environment. In the first phase, the policies are sent to the locations where they should be enforced. In the second phase the policies are activated, but only when there is a commitment that they can be actually enforced in *every* location. Otherwise, a rollback procedure is performed, and the policy deployment fails.

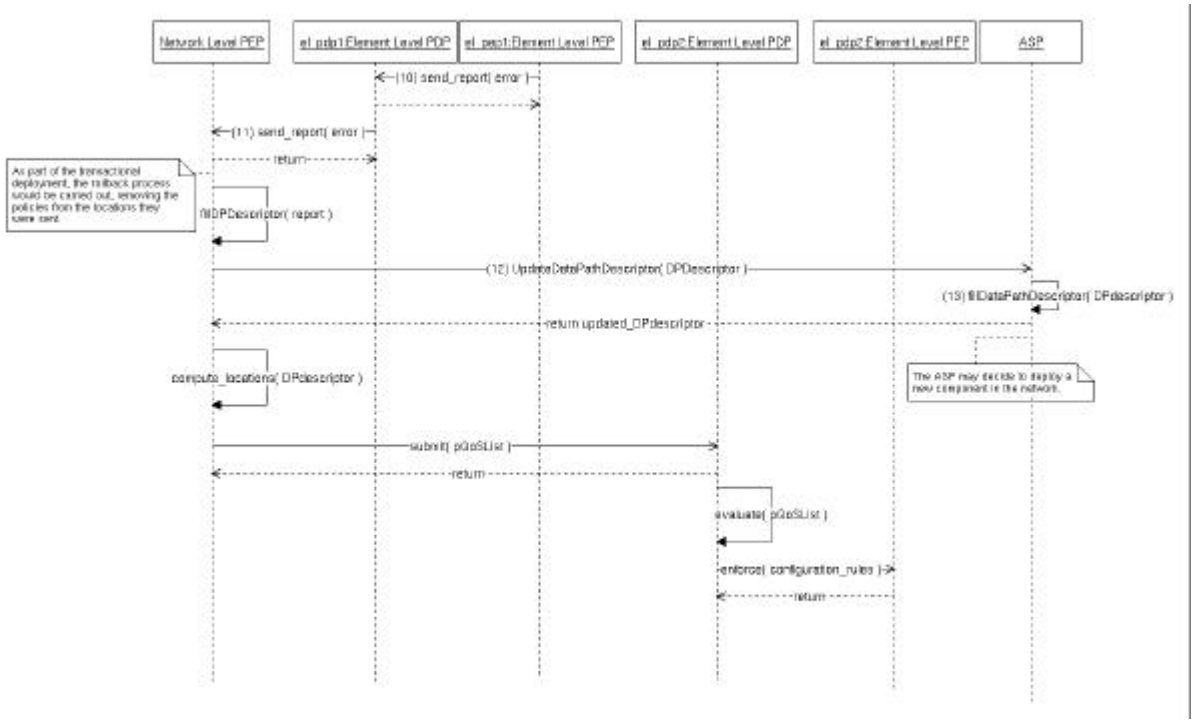
The PDP at the element level makes the necessary translations and delivers the low-level configuration rules to the element level PEP (8), which in turn tries to enforce the policies(9). Based on the acknowledgements (or positive reports) received by the element level PDPs, the network level PEP fills the status field of acknowledged components in the data path descriptor with an "able" flag.

Whenever a policy cannot be enforced due to a lack of resources, a policy deployment error report is sent by the element level PEP to the element level PDP (10), which reports the error to the network level (11). The report includes enough information to identify the offending component within the network. This information is reflected in the data path descriptor by setting the status flag to "unable".



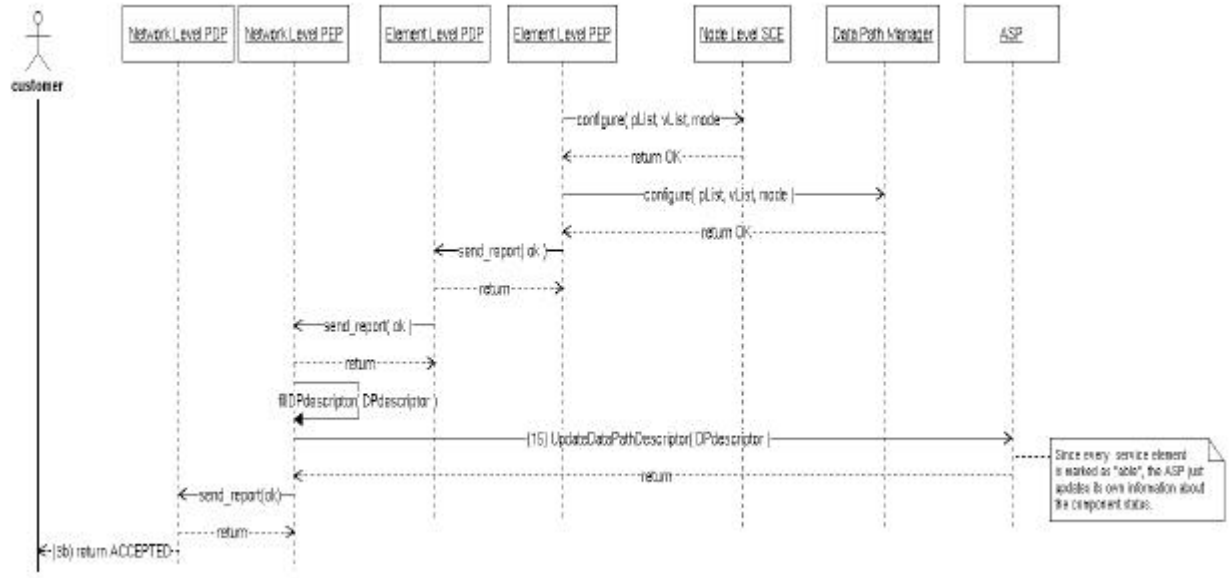
The network level PEP engages a dialogue with the ASP, requesting the deployment of a new component, of the same type that the offending component and in a location fitting the requirements available in the current service descriptor[9] (12). Remember that at this time, the data path descriptor holds information about the components whose locations are suitable, so the ASP knows the available components and can process a new graph for the service. The ASP is responsible for deploying the component, filling the data path descriptor with the new location information and returning it to the management system (13).

The management system then sends the corresponding policies to the new locations in a transactional deployment mode. Normally, at this time the deployment should succeed, but in the case a new error appears, the process would be repeated (14).



Once a positive acknowledgement has been received from every element level entity, the data path descriptor can be considered completed (15). At this point, the element level PEP initiates the node-level data path creation SBB, using the received configuration information to customize the component in every location and actually bind the data flows to the data path.

[9] The service descriptor should include at this time the information regarding which service elements (either components and links) are able to deliver the requested service level.



Implementation Status of Involved Sub-components

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
Network level ASP		Dynamic deployment of new components based on requirements.	
Network level PEP	Forwarding deployment mode.	Transactional policy deployment.	

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.

For this purpose, the data path descriptor will be interpreted as a low level policy which is enforced in the active node by appropriate PEPs. The enforcement implies the modification of the routing tables[10], the configuration of the demultiplexer or the assignment of ports in the EEs.

The descriptor information regarding the service component order may be used to decide the following point to send this low level policy, providing a decentralized means to create the data flows.

Using simple management elements inside the active node we get a flexible way of creating the data path hop by hop.

[10] Based on the data flow descriptors.

SBB Data Path Creation in PromethOS

This SBB describes the download, installation, and instantiation of service components onto a specific active node in the PromethOS/Linux kernel space.

This SBB starts when the VE Manager request the creation of a VE, EE, together with the instantiation of service components from PromethOS.

The code modules are subsequently installed and instantiated.

This SBB stops when all the code modules are instantiated.

Pre-conditions

- VE is operational
- VE is connected to appropriate wrappers
- Kernel is ready for PromethOS, i.e. the running has the required hooks available.
- Service Components are available as code modules on the node.

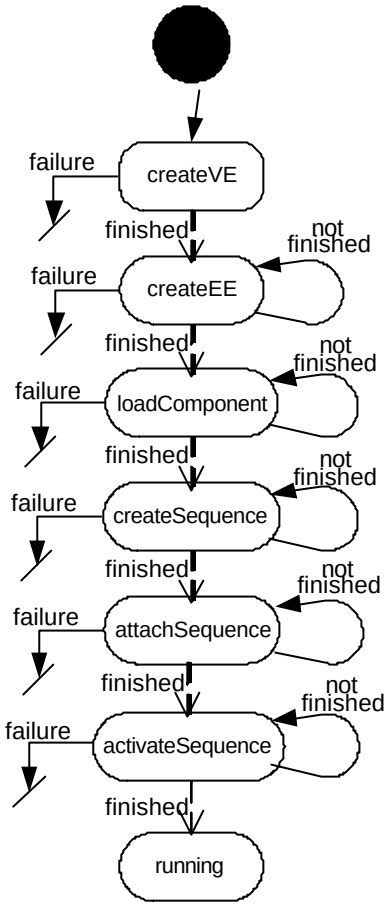
Dependencies

- VE management SBB

Post-conditions

- VE is created in kernel space
- EEs are instantiated in kernel space
- Service components are loaded, instantiated and inter-connected
- Service-part in PromethOS is operational

Sequence Diagram



Implementation Status of Involved Subcomponents

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
PromethOS	Instantiation of 1 EE	Instantiation of N EEs Identification of VE Instantiation of N VEs	D4
PromethOS wrappers		Adaptation to VE Management	

4.1.3 CS 3: Deployment and Instantiation of Services and Service Components.

SBB Service Deployment

This SBB 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.

Pre-conditions

- VE is created.
- Service is released, i.e. service descriptors and service components are ready to be retrieved from service registry, and service repository, respectively.
- Network level mapping of service is determined, i.e. nodes running service components are identified.

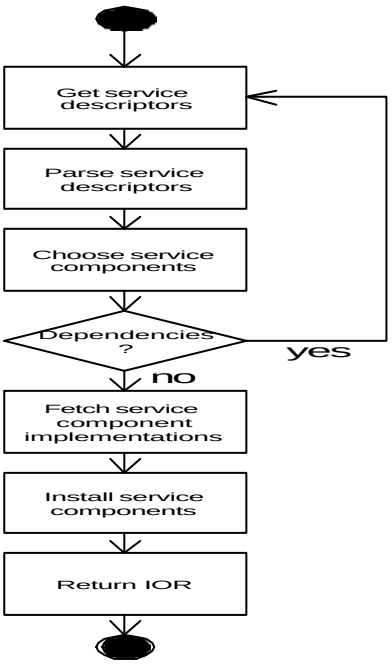
Dependencies

- Service release SBB
- VE creation SBB
- Network level service deployment SBB

Post-conditions

- Service components (simple implementations) are installed and instantiated on a specific node.
- An IOR is returned that allows accessing and configuring the service components.
- Binding information for service components is known by the service creation engine.

Sequence Diagram



Implementation Status of Involved Subcomponents

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
Node ASP manager	Receives service deployment request from network ASP manager		[4]
Service Creation Engine	Determines required service components Extracts EE types of service components	Determine interconnection of service components Interface with node Mgmt framework?	
Code Manager			
Service Registry			
Service Repository			

SBB Retrieve Information related to Service Deployment

This SBB describes the retrieving of service information.

Pre-conditions

- Service is released in the network

Dependencies

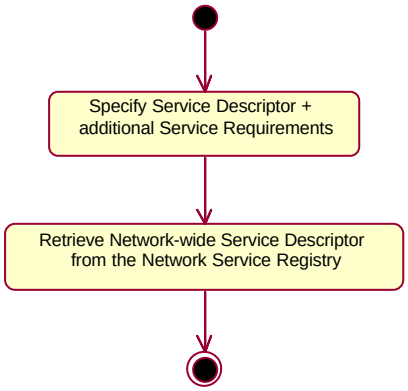
None

Post-conditions

- SP's part for service deployment is completed

Required Functionality of Involved Subcomponents

Activity Diagram



.

SBB Processing of Information Related to Service Deployment

This SBB describes the preprocessing of service information.

Pre-conditions

- Network-wide Service Descriptor and additional Service Requirements (optional) are specified.

Dependencies

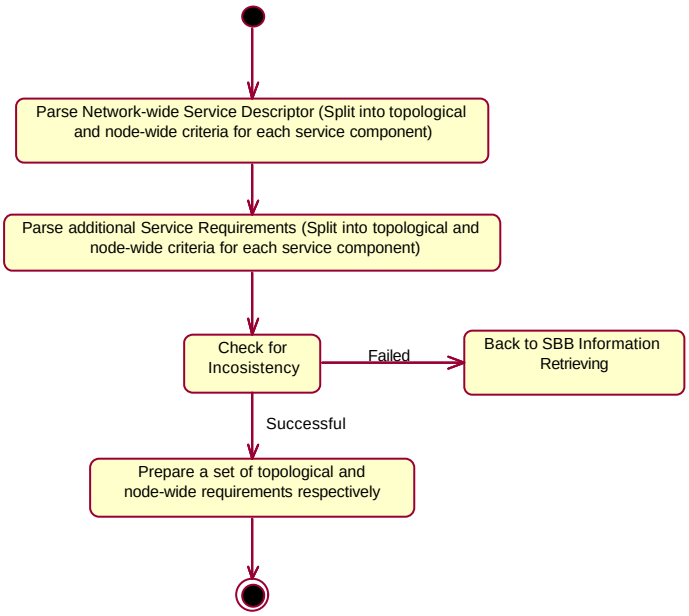
Retrieve information related to service deployment SBB

Post-conditions

- Resource Requirements of each Service Component are identified.

Required Functionality of Involved Subcomponents

Activity Diagram



SBB Mapping

This SBB describes the mapping process of service requirements to available resources.

Pre-conditions

- Resource Requirements of each Service Component are identified.

Dependencies

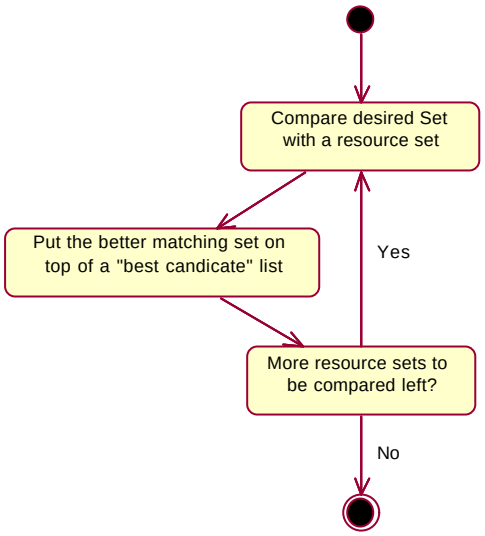
None

Post-conditions

- Two lists of best candidates are produced:
- Fulfilling topological types of requirements, and
- Fulfilling node-wide types of requirements.

Required Functionality of Involved Subcomponents

Activity Diagram



SBB Evaluation

This SBB describes the evaluation of the results of the mapping process.

Pre-conditions

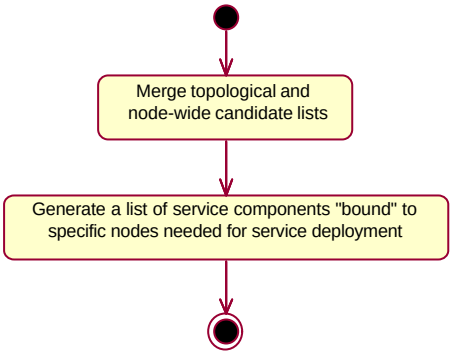
- The mapping process is finished.

Dependencies

Post -conditions

- An assignment of each Service Component of a service to a node, on which it is to deploy, is determined. In certain cases, service components cannot be mapped onto the available resources.

Activity Diagram



SBB Component Installation

This SBB describes the download, installation, and instantiation of service components onto a specific active node.

This SBB 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 terminates when all the code modules are instantiated and installed.

Pre-conditions

- VE is created.
- Service is released, i.e. service descriptors and service components are ready to be retrieved from service registry, and service repository, respectively.
- Network level mapping of service is determined ➔ nodes running service components are identified.

Dependencies

- Service release SBB
- VE creation SBB
- Network level service deployment SBB

Post-conditions

- Service components (simple implementations) are installed and instantiated on a specific node.
- An IOR is returned that allows accessing and configuring the service components.
- Binding information for service components is known by the service creation engine.

Implementation Status of Involved Subcomponents

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
Node ASP manager	Receives service deployment request from network ASP manager		

Service Creation Engine	Determines required service components Extracts EE types of service components	Determine interconnection of service components Interface with node mgmt framework?	Current state described in [1]
Code Manager	Coordinates installation of multiple instances Triggers installation of Components		
Service Registry			
Service Repository			

SBB Instantiate a Service Component

This SBB describes instantiation of a service component in a given VE/EE running on an active node.

The functionality of the SBB can be seen one step in the service deployment process but can be also used to manage the life cycle of the deployed service components.

Pre-conditions

Service component has been installed in the given EE.

Dependencies

Release Service SBB

Install Component SBB

Post-conditions

An instance of a given service component is created in the given VE/EE and a reference to the corresponding runtime instance of

Required Functionality of Involved Subcomponents

Network and Node ASP components

Implementation Status of Involved Subcomponents

Subcomponent Name	Relevant Current Functionality	Required Implementation / Open Issues	Related Documentation
Network ASP manager	Receives service deployment request from network ASP manager	Mapping algorithm under discussion	Deliverable 5
Service Creation Engine	Determines the initial configuration in terms of required service component instances on the given node.	Determine interconnection of service components	**[4]**
Code Manager	Coordinates instantiation of multiple instances Triggers instantiation of Components		[4]
Component Manager (Node Management Framework)	Performs instantiation		[4]

4.1.4 Security Aspects of FAIN

As already mentioned at the beginning of this chapter, security has been identified as an overall requirement of all core scenarios. This is the reason, why a separate section is attributed to security. In the following section, the security aspects of FAIN are listed shortly. As done in the core scenarios SBBs are deduced from the security aspects.

Security Area Interfaces

Security area interfaces are parts of the general blocks identified in the design phase of the security architecture. Her are listed only interfaces that are available as public interfaces to other node subsystems. At the moment this set is limited to few interfaces that are important for operation of the node and security subsystem. Interfaces, that are available through Security Manager, are:

1. addPrincipal,
2. setPolicy,
3. authorize,
4. sendCheck,
5. receiveCheck.

Interface addPrincipal

This interface is exported so the principal can be registered on the node. It should be used during create phase in the process of registering a VE, see section. The code behind the interface will be also used in the SBB of authentication in the case of active packets, see section.

The input to the interface is a digital certificate that will be defined by JSIS.

The principal related secure store or possibility to define principal alias as defined in section are neglected at the moment.

Interface setPolicy

The set policy interface is provided so the VE/service/component/port policies can be defined by a management entity. The interface is called by SID (component identifier), policy and name. tSysName relates to the naming issues on the node.

Interface authorize

Authorize interface is available to other system components and it is called by node enforcement engines like RCF manager. It accepts two SIDs, of the subject (calling component) and the object (accessed component) and environment of the call. At the moment environment of the call is the trickiest part which is not yet defined in the document and will be fostered during the integration and development process. Here environment relates to the older Thomas interface with component, port, operation, tPropertyList.

Call returns a Boolean value.

Interface sendCheck

Interface sendCheck is unchanged from the last milestone release (I hope). Per scenario as defined in section there is a need to define at least the dynamic way of passing the information about the next hop node address from the DeMux to security. Other issue is iAnep Packet interface and treating of the packet as a component to realize scenario in section.

Interface receiveCheck

Interface receiveCheck is unchanged from the previous milestone. There is an issue that was not able to supply VE and ServiceId (EEID) values which is also a matter of ANEP packet definition. This is done in the security area if not in, so the interface will change accordingly. Another issue is treating the packet as a component and build from ANEP options its security context.

Authentication Engine SBB

The authentication engine SBB verifies the authenticity of active packets. It 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 *Active Packet and Creation Option Building* and *Authentication in case of Active Packets*.

Authentication engine provides both data origin authentication and session's authentication. Active packets are authenticated for their data origin and management connections sessions are authenticated for the time of the session setup.

Active packets are authenticated per packet; such packets can pass many nodes so the proposed way of the authentication is based on the use of the digital signatures. In this way only parts of the active packet that don't change in the network can be authenticated. Proposed authentication doesn't provide transaction property for the protected data.

Management connections to the ANNs are sessions, usually CORBA connections to the node. For this type of the connection we propose use of the CORBA over SSL. User is authenticated once per session (authentication of the client to the server, but should be possible also vice versa) and the security context of the session is build in the same way as in the case of the active packets. To be able to attach the established security context somewhere, some kind of session proxy (component) is needed for each session. This proxy ``speaks for'' the entity in the connection and all security related decisions are made based on this component security context.

To be able to establish the security context the identifiers of the principal (VE identifier and principal attributes) should be available while setting up the connection (this means that the digital certificate must contain both identifiers). In the active packet case the service identifier should be explicit in the packet and also stated in the credentials.

Authentication on the node is explicit in the context of active packets or uses protocol like SSL to provide proper authentication of principal and data.

Active Packet Signing and Credential Option Building SBB

The goal of this SBB is to sign an active packet and build the credential option. For the signing one of the principal private keys, stored in the principal secure store, is used. Digital signature covers the active packet payload and the credential option itself.

To be able to validate so generated signature the credential option is build that points to a corresponding public key of the key pair, to which belongs the private key used to compute the signature.

We have planned three possibilities to point to a public key in D4 [6]: X.509 certificates, X.509 attribute certificates and Keynote credentials. To this we are adding another option that is easy to implement and is very straight forward; the public key is identified by its cryptographic hash of the length of 128 bits. Solution is similar to that planned for use in the case of Keynote credential; in this case we have in the credential instead the name of the principal directly public key. Also in this case this public key will be replaced with a hash of a key as a unified name of a principal. For getting a public key from it's ``name" same approach will be designed for both cases.

Active packet is signed at the originating node or intermediate node like a gateway. Signing is triggered because the active service (principal) is sending the packet (originating) on the principal node or it is signed by the node because the node responds with a replay; replay can be also an error message if any. These messages are both service specific; it remains open question if they are signed by the node or principal. First case is tricky because the node can be fooled to sign the message that it doesn't understand. This requires further study and service experience.

For signing to take place we have two options: first, that the signing is triggered by (De)Mux itself and second that the signing is triggered by an active service alone. As designed in D2 [5] the authentication is mandatory for active packets, so all packets must be digitally signed and their data origin is authenticated on the basis of the digital signature. To ease the development, the signature will be added by the Security subsystem during the sendCheck function call made by.

We are working on solutions that can ease the D2 requirement, for example like treating separately control and transport plain regarding the authentication.

Signing an active packet has to be treated in the same way on the client and network node. Important question here is where to gather needed data for active packet header including the building of credential option(s). It would be the right thing in this context to treat the active packet as a component. During the creation of the packet we could set the security context of the created packet included with the pointer (principalId) to the principal credentials. Technically iAnepPacket interface should treat only ANEP header options as defined in FAIN; payload and variable option should be transparent (byte streams) at this level of abstraction. It remains open who is responsible to add an serviceId to the packet payload if we agree that the serviceId can be stored there. Information from the security context can be used to build the following ANEP options transparently: VE, ServiceId and the credential option(s). Still needed information for hop-by-hop protection is then next hop node address or list of addresses. The interface between (De)Mux and security changes then slightly in the way that it invokes the interface with a reference to the component (packet) and the next hop destination.

Pre-conditions

Principal (or system entity) has to be defined on the node. A principal or a node has to have at least one valid key pair which has to be stored in the principal Secure Store. Service that originates active packets has to be started and its security context has to be defined.

Post-conditions

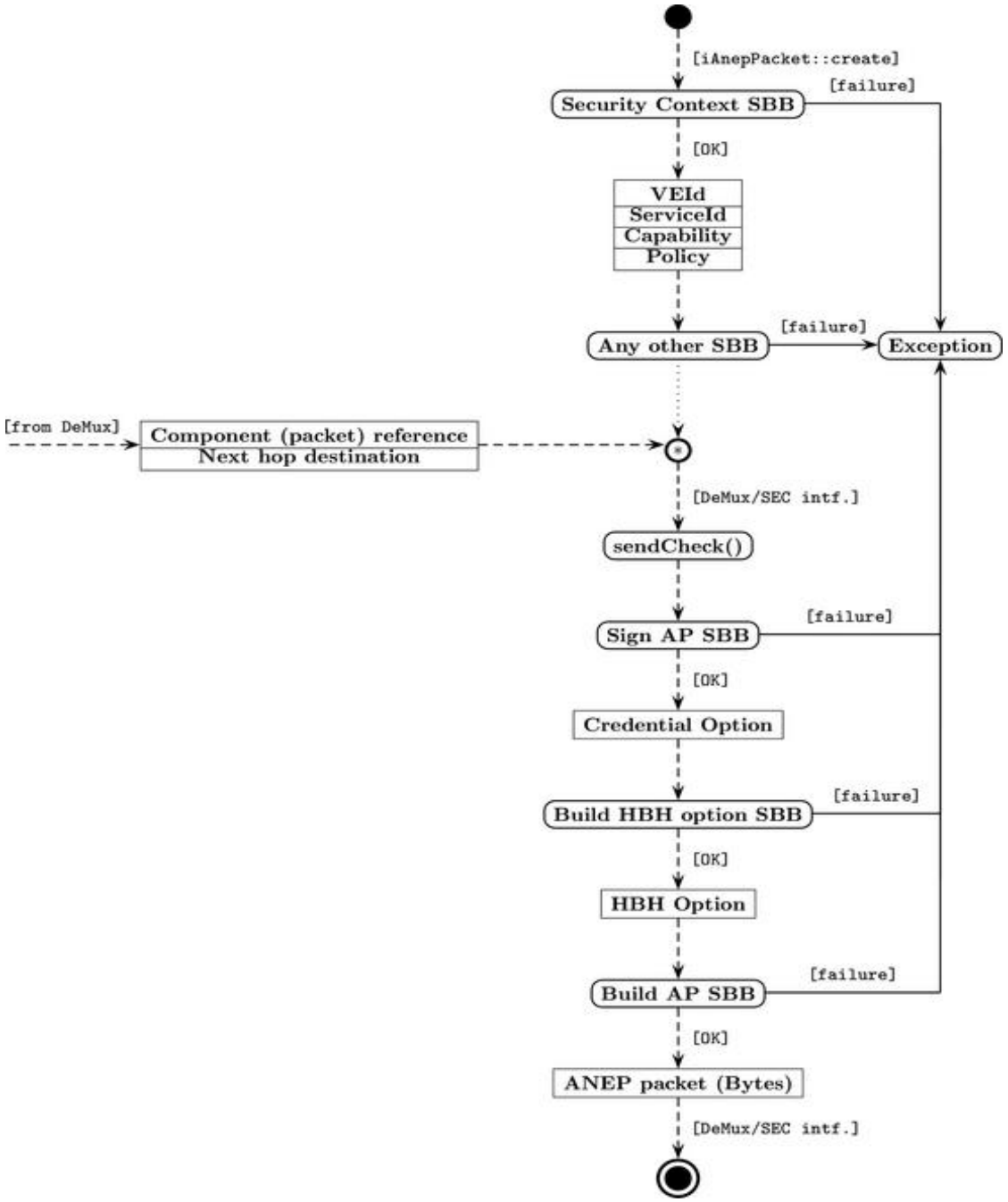
Build credential option with the digital signature.

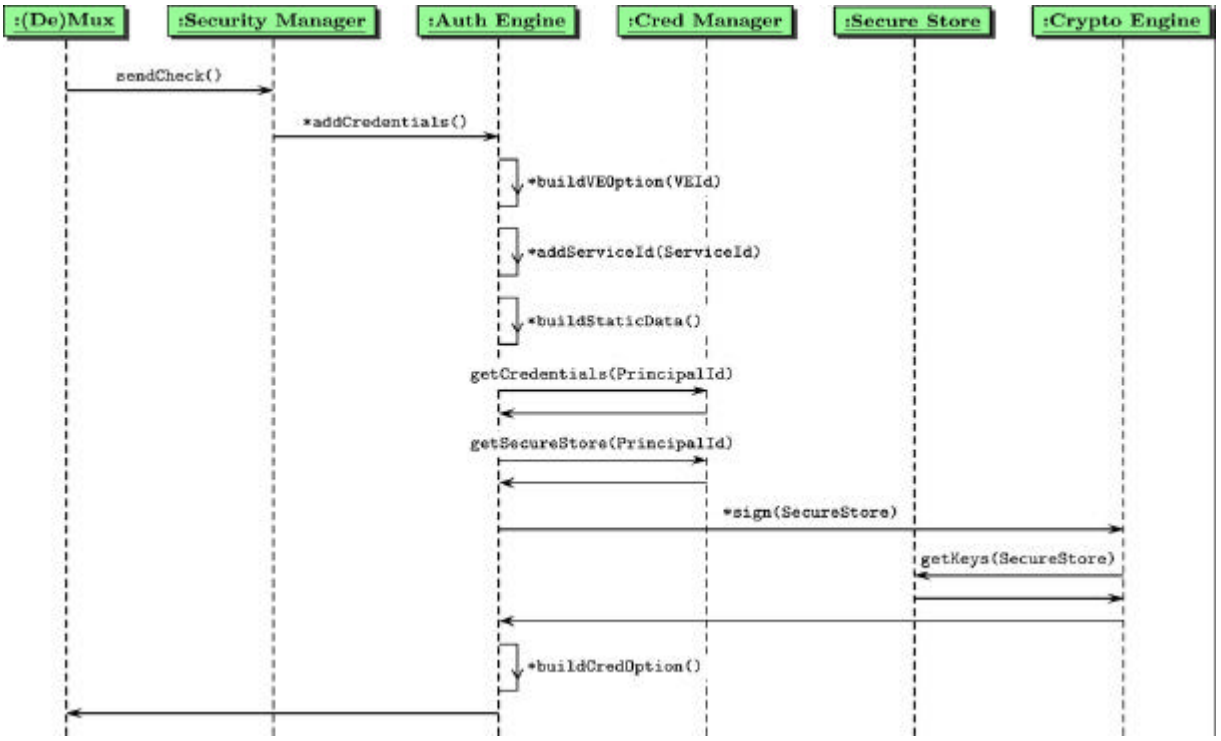
Dependencies

SBB depends on:

- Building of the Security Context SBB.
- Get principal credentials related information SBB.
- Get principal related secure store SBB.
- Interface definition between and SEC.
- Availability of iAnepPacket interface.

Sequence Diagram





Authentication in the case of Active Packets SBB

Authentication of data origin is done with digital signature mechanism. The SBB follows the description and sequence diagram of authentication in D2 [5]. If there are many authenticators in the packet (credential options) the procedure has to be repeated for every authenticator.

authenticate, fetchCredential, verifyCredential, validateSignature and resolve are Security subsystem local functions and are as such marked with a *. authenticate is entry level function for authentication engine. fetchCredential fetches the referenced credentials in the credential option from remote server. verifyCredential assumes verifying credential (including possible certification path), validateSignature validate digital signature of the active packet in the credential option, see [5] and [6], and resolve resolves information in the credential (principal identity and attributes). verifyCredential includes also signature validation, not shown in the above diagram of the signatures in the certificates of the possible certification path. From the verified credential (possible after verifying certification path) principal related public key is obtained. With this key we validate the signature in the credential option.

Procedure is repeated for every credential option in the packet.

Pre-conditions

A signed active packet (with a credential option) as described in the previous SBB. Demultiplexing subsystem is initialized and running.

Post-conditions

Authenticated active packet on the node.

Dependencies

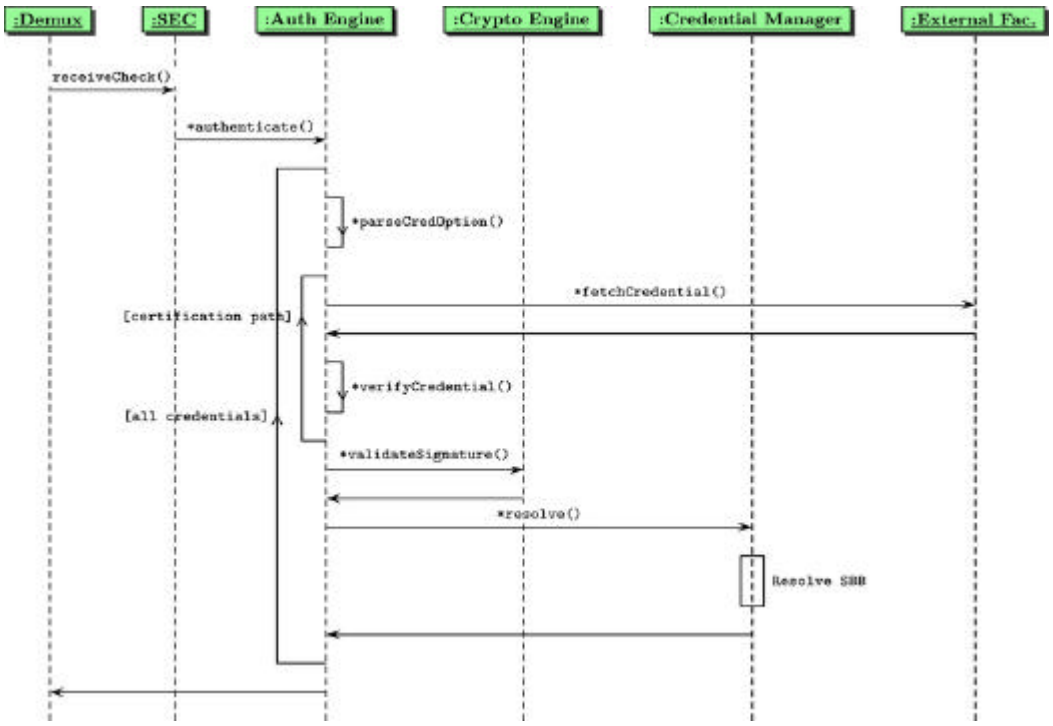
Possible dependencies on external services like certification authority, network services for obtaining principal public key certificates (DNSSEC, LDAP) etc.

SBB is dependent on common interface between Security area and Demultiplexing.

Dependency regarding other SBBs:

1. sign active packet SBB,
2. Resolve SBB.

Sequence Diagram



Authentication of Sessions SBB

Sessions are defined as communication between usually a client and a server which last for short period of time. Sessions are connection-oriented end-to-end on going communication between two entities.

At the beginning of the session peer entities are authenticated (client to a server and/or server to a client) what is called peer entity authentication. After authentication, entities can exchange a session key and use symmetric cryptography with keyed hash to provide data confidentiality and integrity service for exchanged data thereafter for the time of a session. Most used protocols for protecting sessions are SSL or TLS. SSL is also preferred protocol for protecting CORBA based communication services.

SBB plans to use CORBA over SSL for authentication.

Pre-conditions

CORBA implementation with SSL support.

Post-conditions

Authenticated entity for time being of the session connection to the node.

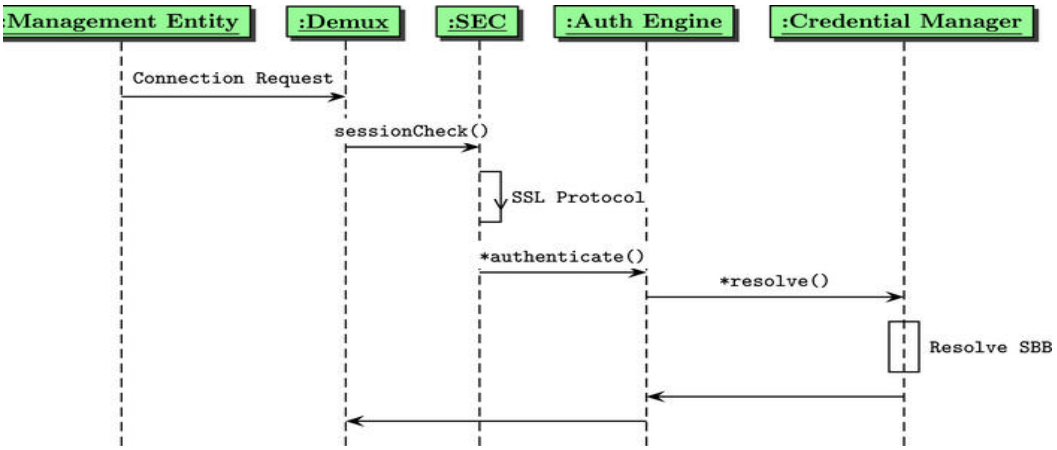
Dependencies

Depends on granularity of CORBA over SSL. Each entity that wants to manage the node should be connected to the node over separate session even in the case that more than one entity is connecting to the node from a single management station.

Depends on the following SBBs:

1. Resolve SBB.

Sequence Diagram



Security Manager SBB

Accepts request from enforcement engines, collect request related credential information, accessed object policy information and ask authorization engine for authorization decision. This decision is passed back to enforcement engines. It provides NodeOS interfaces for managing AN users related credentials and security policies.

Security manager is the core of the node security services. It exports the majority of the security area interfaces related to credential database, policy database and authorization engine and manages communication with those two databases and authorization engine. Besides is responsible for labeling decisions and building the security context of the components in the system.

The basic design decisions for security manager were: separation of the application and policy, separation of authorization decision and authorization decision enforcement, possibility of having multiple authorization engines (for example basic one and keynote with more expressive policy), general authorization decision mechanism.

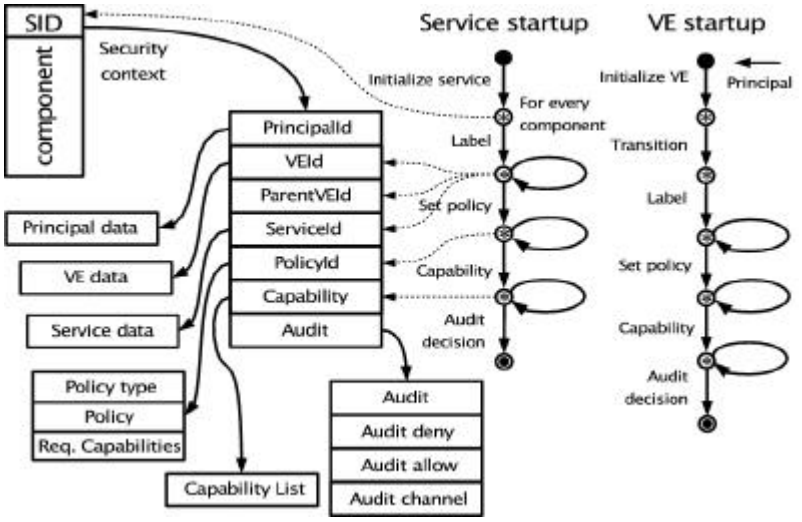
The security manager should provide: ability to clearly separate users and services on the node, additional fine grain policies for access to certain node resources like file, memory region or interfaces, default policies if there is no specific one available, list of past decisions, caching of authorization decisions.

Security Context

All decisions that the Security Manager make are based on the security contexts of the subject accessing the object, possible cached authorization decisions and environment of the access. These decisions represent a part of state information of the system.

Security context represents all basic information about the component that is needed to perform various security related operations.

Security context of a component is setup during the service start up. Security context is defined currently with the following parameters: principalId, VEId, ServiceId, component capabilities, component policies and audit vector. The security context of the component is attached to the component with a SID; SID is a component secure identifier which is opaque to the rest of the system. Security context is hold exclusively in the Security Manager and it is interpreted in it only. Security context, service start up and VE start up are presented in the figure below.



Structures:	
SID	points to a component security context
principalId	points to principal data
VEId	Virtual Environment Identifier
ParentVEId	parent VE Identifier
ServiceId	service Identifier
PolicyId	points to component related policy
Capability	Component capabilities
Audit	Audit vector

Security context structures in detail

SID

Opaque pointer to security context

Security Context

PrincipalId

Points to principal related data which is stored in the credential manager,

VEId

Points to Virtual Environment security related data, mainly VE identifier,

ParentVEId

Parent VE Identifier, which is assigned at VE start up from the security context of the starting component,

ServiceId

Identifier of the service on the node. Points to service related data structure, which contains service name, service code modules data and their related security information. Needs to be aligned with ASP,

PolicyId

Policy identifier, which points to policy data. Data type defines policy authorization engine to be used in the process of providing an authorization decision. In policy data is also a list of required capabilities to access the component. These capabilities are evaluated in the capability engine. Polices has to be defined in advance and can be supplied via the management system or dynamic through a packet,

Capability

List of component capabilities, supplied together with a service code modules or service descriptor.

Audit vector

Audit vector defines which decisions (denied or allowed) are audited to which audit channel.

Creating a SID SBB

This SBB describes the process of creating a SID. For our purposes the SID is a 32 bit random number. Other design issues are neglected at the moment.

Pre-conditions

Security manager is started. All other components get a SID from the security manager.

Post-conditions

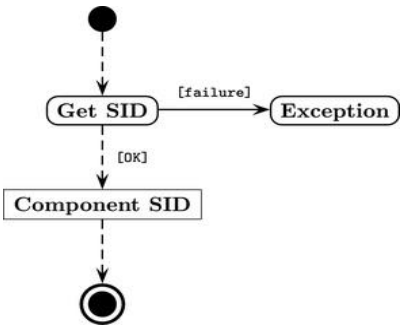
Unique SID is generated for every component.

Dependencies

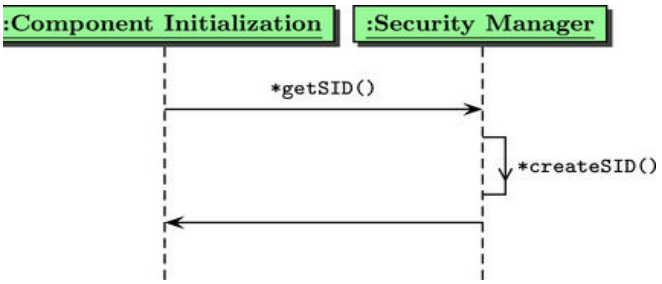
Depends on unique meaning.

Needs to be random and fast.

Activity Diagram



Sequence Diagram



Labeling SBB

This SBB describes the labeling process. The components that implements a service are labeled during the service start up with the label of particular VE and ServiceId. getServiceId() in the sequence diagram creates the ServiceId if the ServiceId is not defined already.

Pre-conditions

Security Manager is started. Component (or service) that starts a new service should be labeled and started.

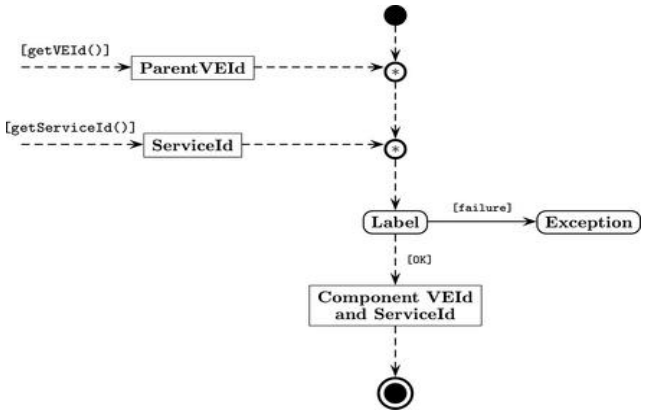
Post-conditions

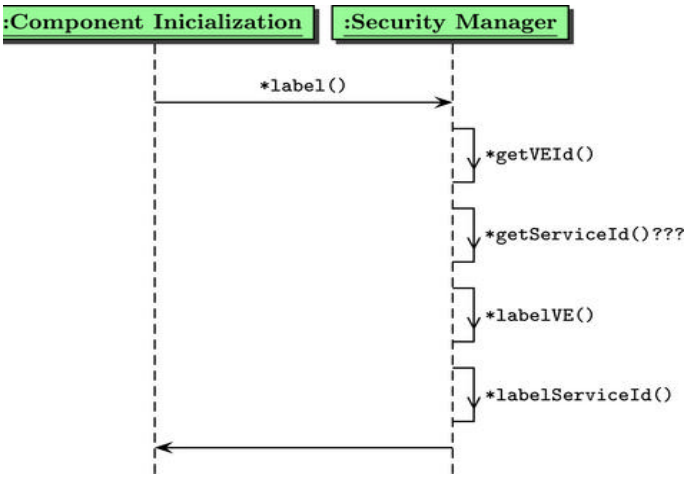
Label component in the context of certain services.

Dependencies

Many unresolved issues, especially with the starting points of the system. Problems between service and component at start up. Problem is that we know when the component is initialized but not when the service is.

Sequence Diagram





Transition SBB

Transition SBB is used only in the case of VE start up (activation). Current VE creation SBB assumes two steps in VE creation: create, which is aimed at resource reservation and principal definition and activate which actually starts iComponentInital in the name of the principal. Transition is related to the activate phase, when the component (parent) is starting a component with different VEId than it is its own. At that moment, as can be seen in figure below, if the authorization decision permits the component security context is relabeled and also principal data is set to the VE owner data. Association with the principal is related to the create phase; at that stage the principal has to be associated with the VE that he will own. create and activate are run by the parent principal (pVE). After that every component created through the VEs iComponentInital is automatically labeled with the VEId of the parent component except when the newly created component is transitioned.

Pre-conditions

Basic pVE services should be started. A component that can start a new VE has to be started and has to hold a capability to start a new VE. Principal that will hold a VE has to be defined on the node (assuming a create VE call, through profile, which defines also a principal VE resource box).

Post-conditions

iComponentInital which is labeled with the right VEId and ServiceId.

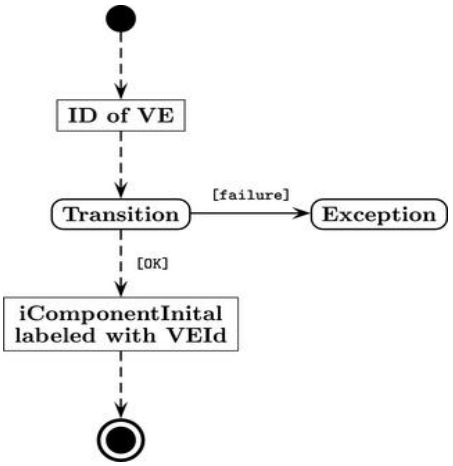
Dependencies

There should exists a mapping between a principal and VEId on the node.

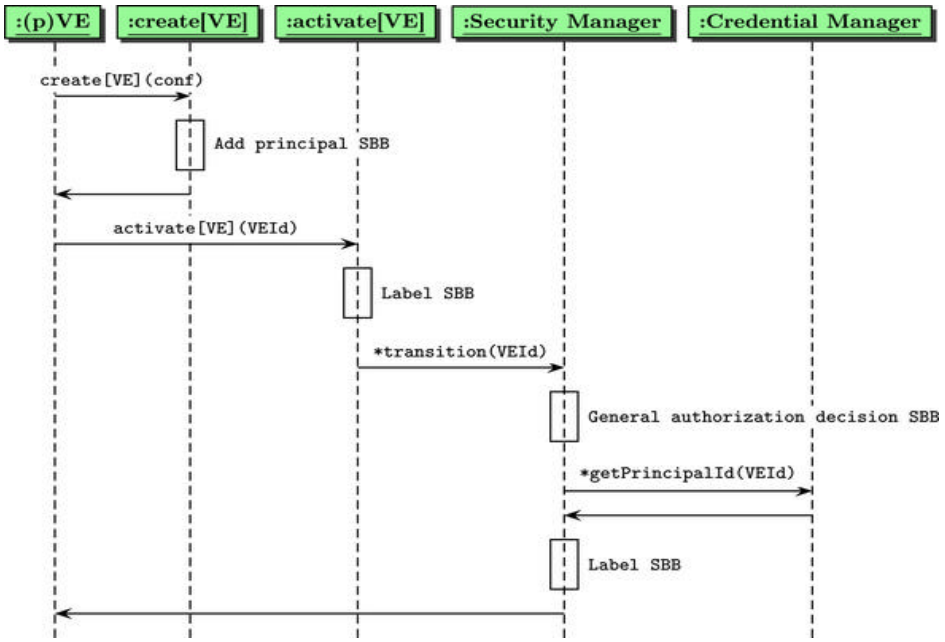
SBB depends on the following SBBs:

- low level engine SBB,
- Capability engine SBB.

Transition SBB



Sequence Diagram



Building of a Security Context SBB

Building of the security context is a crucial SBB because on the basis of this process the authorization decisions are made. In general we can distinguish between four cases when to build a context: for active packets, sessions to the node, for started components and transitioned components. All these subSBBs should lead to the same data structure that is then used in access control decisions.

In the case of started and transitioned components the basic parts were already covered in the previous SBBs. Such components can have also a policies attached which govern access to the component or component ports. Audit ``policy'' can be also attached though is not at high priority at the moment.

Other two cases will be covered separately.

Pre-conditions

Security subsystem is running. pVE icomponentInitial is transitioned to default values. If there exists component security policies (policy, capability) they are set during the security context setup. If the component needs certain capability to access system services this capability is defined.

Post-conditions

Security context of a component is built.

Dependencies

Labeling of services depend on definition of services. Will try to define security components as a service? The problem is that we want to separate with ServiceId basic services offered by the pVE or any VE. Problem of small ASP component acting as resolver of the service descriptor and the definition of the service start up.

- SBB add principal,
- SBB label,
- SBB transition,
- SBB set policy,
- SBB set capability policy,
- SBB set capability

Building of a security context of the active packet or a session is similar to the activity presented in figures above. Security context should be build for every active packet or a session to a node. Both cases are also similar so we will present in this SBB only the active packet case.

The information that we can build a active packet security context from has to be available in the packet. At first there should be information that we can match the packet to the VE/service on the node. VE and ServiceId should be explicitly stated in the active packet and also cryptographically protected with integrity service provided by digital signature mechanisms as described in D2 and D4 [6]. To get around initial problem, which sets these values, we require that these matching values are also clearly stated in the principal credentials (these are signed by system entity you trust).

So the labeling in the activity diagram on figures above corresponds to setting up the VEId and ServiceId values in the packet and verifying them is verifying their statement regarding the information (attributes) in the principal credential.

Capabilities are matter of the internal system and should not appear outside the system (node). Finer granularity of the access to the interfaces of some component that has certain capabilities or is otherwise protected is achieved with the component/port policy. These policies are in general internal matter of the principal (VE owner). The only requirement is that the policy type which defines the authorization engine is registered together with proper engine on the node. This fact has to be checked at the service start up, during policy setup, see SBB, section. Also for example the problem of active code that needs certain capabilities (let's say access to the routing table) has to be solved through its programming environment (can be proxy component) that posses needed capabilities. This component can be additionally protected with component/port policy.

PrincipalId is not explicitly mentioned in the building of the security context SBB. The association of the principal in the presence of the VE identifier can suggest that the notion of the principal in the context is irrelevant. This can be true for some components; in this case, for example in the service start up the value of PrincipalId is only attached to the component security context. In the case of other components, like those build in the case of sessions or active packets, the PrincipalId doesn't match the VE owner and this principal information is used in authorization decision process and not that of the VE owner. The distinction is also crucial in the case when a component can generate active packets; the credentials, related to the principal, who will be attached to the packet through credential option, are automatically got from the component security context. In this case the VEId can be related to one principal and the PrincipalId can point to other set of credentials. To be able to handle all cases transparently, we are using the same data structure in all cases.

Get Security Context by SID SBB

Getting a component security context by SID is a short SBB to complete the Security Manager SBBs. Security Manager is responsible to extract from component security context authorization engine relevant information and pass the information to the authorization engine. Based on this information authorization engine makes an authorization decision. EngineId is just internal mapping between parts of the security context and related authorization engine. Assumed to be predefined.

Pre-conditions Component is running and its security context was build. Component participates either as subject or as an object in the process of providing an authorization decision.

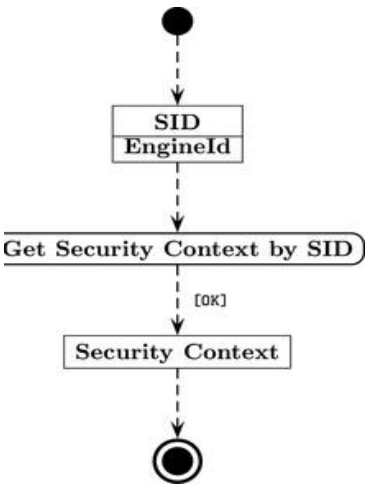
Post-conditions

Part of component security context is available to security manager.

Dependencies

The SBB depends on SBB building a security context.

Activity Diagram



General Authorization Decision SBB

As discussed in the section there are three levels of authorization decisions. Primary there is one related to the services and VEs and provide separation between them on the node, second related to the capabilities and third related to the component/port policies.

Low level policy can be overwritten by capability engine decision. Finer granularity regarding access from outside can be specified with component/port policy. Note that component/port policy cannot overwrite low level policy.

Pre-conditions

Services are set up and running labeled or transitioned on the node. Needed capabilities and component/port policies of the components implementing the service are set. Component or component on behalf of the user (an active packet or a session) is accessing another component.

Post-conditions

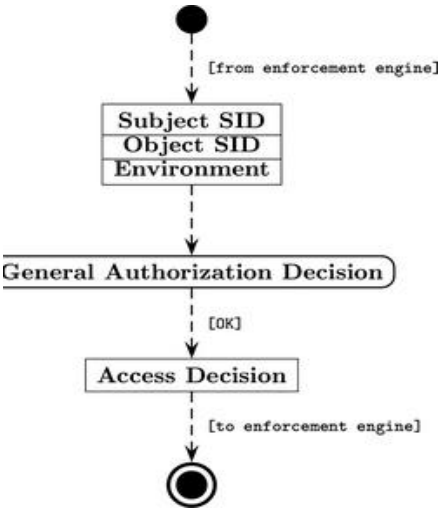
Authorization decision about the access is provided and returned to the enforcement engine.

Dependencies

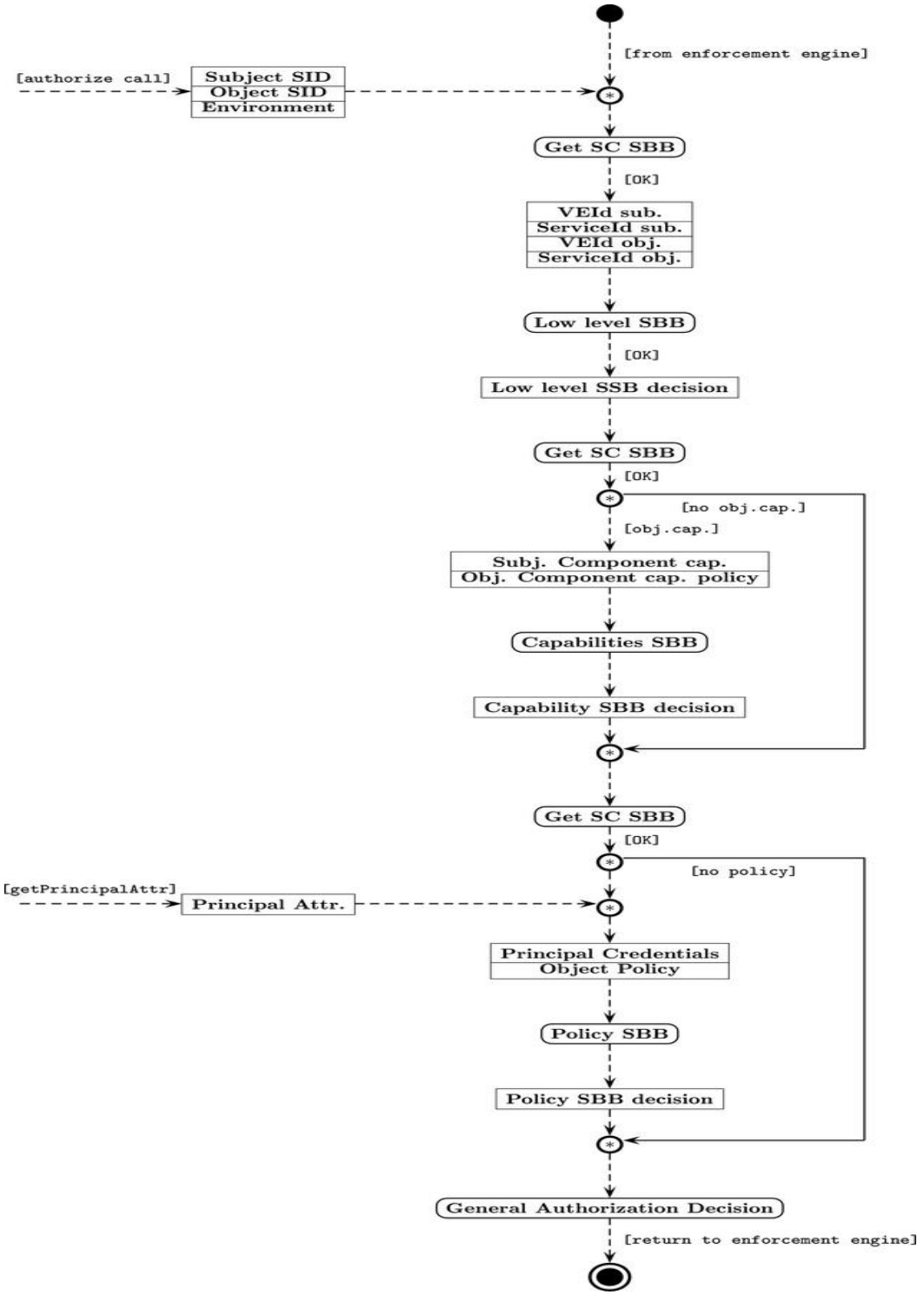
Depends on the following SBBs:

- building of the security context SBB,
- low level policies engine SBB,
- capabilities engine SBB,
- Component/port engine SBB.

Activity Diagram



Activity Diagram



Credential manager

Credential manager, when asked by security manager, searches credential DB and returns all credentials that are relevant for a particular request, which is currently subject to authorization. It also provides facilities for editing credential database, either manually by an authorized user, or automatically, i.e. searches and downloads credentials from an external credential repository.

Some of the credential manager interfaces are exported via security manager like NodeOS interfaces.

Principal owns a VE. Security Manager is responsible for keeping the relation between the VEs and principalIds.

Interfaces	
createCredDB	create new credential database
deleteCredDB	delete credential database
storeCredDB	store credential database
addPrincipal	adds new principal to credentials database
removePrincipal	remove principal from the credentials database
modifyCredentials	modify existing users credentials
listPrincipals	list registered principals on the node
searchPrincipal	search principal by defined attribute
getCredentials	get the principal related credentials
resolve	resolve the principal related credentials

Exceptions	
noSuchPrincipal	addressed principal doesn't exist
principalExists	principal already define on the node
resolveFailed	process of resolving credentials has failed

Structures	
Principal	principal identity and attributes

Interfaces in detail

createCredDB

the interface is added among exported interfaces for the following reason; principal can have its own credential database for its own definitions of a principal if allowed or negotiated. Otherwise the functionality of the call is used implicitly when the privilege VE is instantiated. So added principals are used only in the context of the VE and in the access control decisions based on the service/component policy or decisions made in principal supplied software. Credential databases of the principal are distinguished from node database on the basis of the security context of a database.

deleteCredDB

this call deletes VEs credentials database.

storeCredDB

store credential database to a persistent storage.

addPrincipal

Interface adds principal to the principal database together with principal attributes. General principal attributes can be access identity, group, role, clearance, audit identity, charging id etc.

It should be able to add the user to the credentials database in two ways: from the supplied credentials (digital certificate) or with a call with certain parameters. Note that in the latter case digital certificates and corresponding key pair can be needed for certain principal operation. In the first case the digital certificate (for example X.509 or X.509 attribute certificate) has to have defined principal attributes.

removePrincipal

Remove the principal

modifyCredentials

Modify principal attributes

listPrincipals

Return list of current principals

searchPrincipal

Search for a principal

getCredentials

Gets credentials from the remote location or local database for the named principal. This interface is considered public because of the possible pull model (in contrast to push model as assumed in addPrincipal interface). After this call the entity creating principal can call addPrincipal. Interface should accept method of getting the credentials as an input.

resolve

Resolves principal related credentials got with getCredentials to identity, attributes and generate credentials list.

Structures in detail

Current implementation will implement only few principal related attributes.

Principal

identity

Cryptographic hash of the user public key,

alias

User alias or human readable string,

attributes

Principal related attributes,

credentials list

List of the principal related credentials (digital certificates),

secure store

Points to the principal secure store.

While identity, attributes and credentials list can be result of the resolving process, the alias and secure store are mainly intended for end host usage (though at least one principal on the each node will have defined secure store).

Add Principal to Credentials Database SBB

This SBB adds the principal to a credentials database. Registered principals are stored and authenticated and resolved principals are not if the settings on the node are set as such.

Pre-conditions

Credential database is already created on the node. PVE related software is already installed on the node and operational.

Post-conditions

User is inserted together with extended attributes into the credential database on the node.

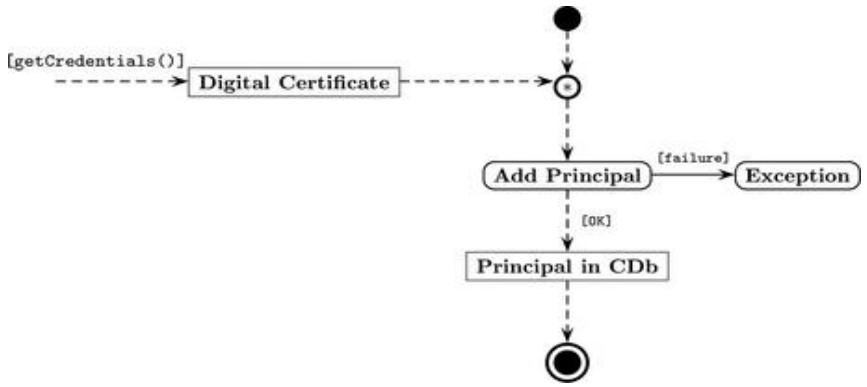
Dependencies

Credential database has to be presented on the node.

SBB depends on the following SBBs:

Resolve SBB,
General authorization decision SBB

Activity Diagram



Sequence Diagram

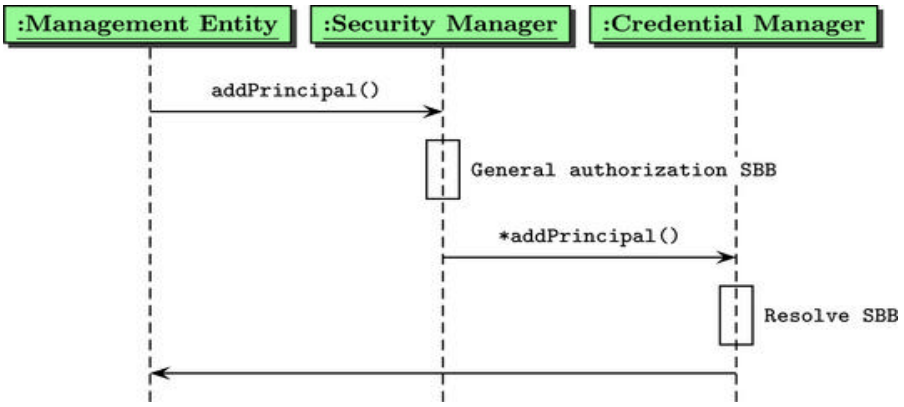


Figure 19: Inserting a principal into principal database

Removing Principal from the Credentials Database

This SBB removes the principal from the credentials database. As is shown on the figure we have assumed that the removing of the VE is guided by the management on the node and that includes the SBB of removing the principal.

Pre-conditions

All principal related components including VE (if principal owns a VE) should be terminated or removed (persistent storage). Principal owned service descriptors and code modules should be removed by ASP. RFC related principal policies should be removed.

Post-conditions

Principal is removed from the credential database.

Dependencies

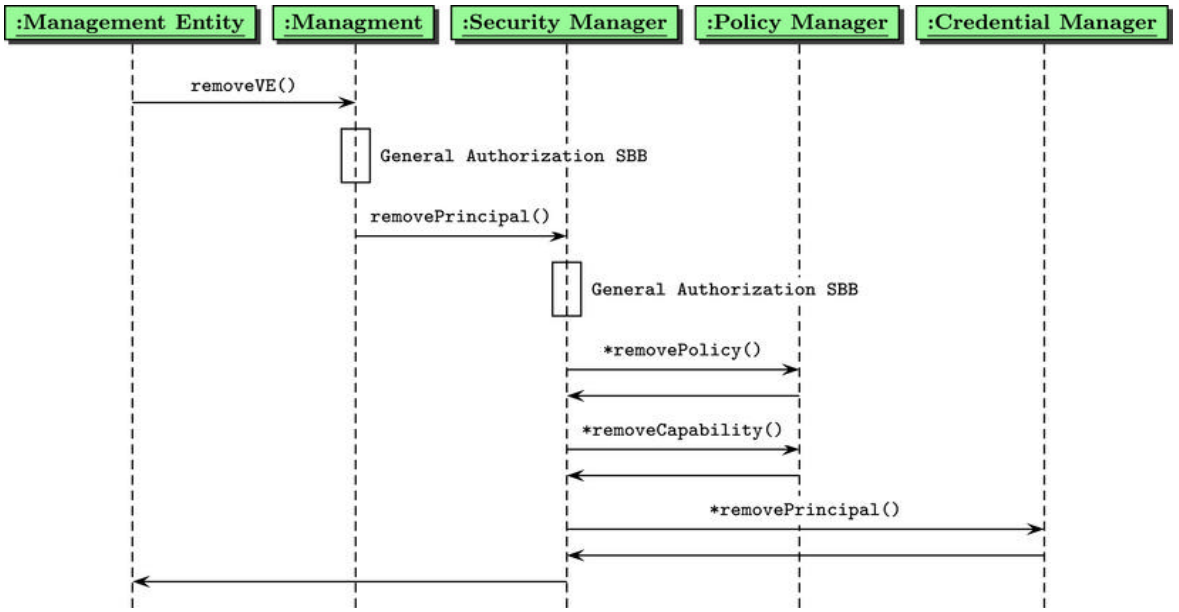
ASP: removal of principal related service descriptor and his own code modules.

RFC: removal of the principal related resource policies.

Management: guide the process of removing of the VE. Termination of the running components and returning of the resources to the node pool?

: removing principal related channels.

Sequence Diagram



SBB Fetch Principals Credentials

SBB covers the process of fetching or getting principal related credentials from the remote store or local node cache.

SBB Get Principal Attributes

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

SBB Get Credential Information

SBB covers the case when we need to get from principal credential (digital certificate) information which will be used in the SBB of signing an active packet.

SBB Resolve Principal Related Credentials

SBB 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

SBB covers the need to keep and get from somewhere the information about principal secure store where principal keying material is kept. This information is highly sensitive so only selected components will have access to this information. Used in the SBB of signing an active packet. If there is more then one credential to build the credential option this information can be related to the credential list.

Policy manager

Policy manager, when asked by the security manager, searches policy DB and returns all security policies that are relevant for a particular request, which is currently subject to authorization. It also provides facilities for editing entries in policy DB, either manually by an authorized user, or automatically, i.e. download policies from a centralized policy server.

There are number of policies that can be evaluated in the process of providing authorization decision and enforced on the node. In general policy type defines authorization engine. Policy types can be MLS, DT or more flexible like KeyNote policies (flexible in a way that you can define multiple different models with same policy language and single authorization engine). The policy should be detached from application, so it should not be hard coded in the application.

There exist at least two levels of policies in the system. First are low level system policies that define the default system behavior. The second level is policies that can be set by the principals that can start the active services.

Low level policies enable us to separate principals and their services on the node. Low level policies are set up during the labeling decision process that labels the service components (running processes or threads) with principal, service and other identifiers. In the same manner all other system resources which are needed for a service are labeled, like files etc.

Authorization decision on the basis of labeling is made in authorization engine and is defined as separate engine from second level authorization engines. There should be a possibility for multiple low level authorization engines.

Next level policies are defined as policies that can be set by principal starting the service. If these policies are not defined default node policies should be applied. These policies are meant mainly for ``external'' service ports and service resources like file or memory region.

When starting a service the principal should define a policy who can access the service and who can manage the service. While there can be many possible policy types and implementations the policy type must have corresponding authorization engine available on the node which can provide authorization decision.

Default policy for accessing the service ports is based on ACLs. Single ACL consists of various groups and their privileges. Privileges are simple and defined as follows:

- **r** read
- **w** write
- **x** execute
- **a** append

Groups depend on principal in question and how the principal has organized his services. Needed security services must be available on the node at advance; for example to do authentication and to get validated entity attributes.

For example the service can be setup by principal which plans for this service two set of users: users and managers. These two correspond to two groups with different privileges. Ports can be accessed for example via CORBA over SSL; for this session the node has to be authenticated to the client and the client to the node. Client credentials should have included suitable user's attributes and clearly defined principal name. On the other hand the suitable credentials can be included or referenced in ANEP packet and packet data origin is authenticated for every packet and users attributes validated on every node passed.

Till now we have defined two sets of policies: low level related to the VEs and services and general policies which govern access to the components and/or ports. First separate services and VEs in between and second provide fine grain principal access to the components or interfaces that those components offer. While this can be sufficient for an ordinary system for an network element which is shared among a lot of principals and act as an multi-user and multi task system this is not enough.

We can easily imagine single user on the active node performing multiple tasks; privilege VE is clear example of such an user. We don't want that the all user components have all the user privileges and it is true that these privileges can be decided and thus the burden of decision shifted to the fine grain policies. But mistakes of the administrators are common and there is with only two level policies always a danger that the ability of the components is misused. Therefore we propose that the ability of the components is separated between the components that really need them. Modularity achieved by component model is a great help to this problem; we can assign the component unique system abilities that support the principle of a least privilege, so that the software that needs certain abilities gets them to that extent which is needed to perform the designed and required tasks.

On the other hand separation principle that is introduced by low level policies is quite often too hard; we would like that in some controlled cases this low level policies can be overruled; obvious cases are during the VE start up and service shutdown or termination. During those two processes for example the processes are transitioned to new principal authority (and different VE and ServiceId) or services and/or VEs can be terminated; in this context we don't want that all VE components have default ability to do these two operations.

For this reason we have defined component/principal capabilities that define their ability regarding some special NodeOS operations. We can state some possible examples from NodeOS subcomponent point of view:

- management: boot_node, start_ve, start_service, terminate_service, terminate_ve, create_routing_table, create_template_repository...
- system: set_nodename, routing_table, neighbor_table, log, start_log
- rfc: reserve_resources, limit_resources...
- : set_filter_rule, send_raw, send_filter...
- security: create_cred_database, create_policy_database, create_auth_engine, security_service, low_policy, policy_database, audit, create_audit_channel...
- ASP

The list is not exclusive or mandatory in the moment; it tries to better understand the ability of NodeOS components regarding performing their ordinary work. Nice examples can be the ability to receive raw packets and Security area capability to process them. Another example is secure store ability to store principal keying material and Crypto Engine capability to read it.

For the reasons stated the capability policies are separated in two parts: component ability to do something which are expressed through required capabilities in the security context and component capability to access something that has ability to do the task.

This ``policies'' are mainly related to the operation of the node system and are not visible outside the node. Such policies help in general way to improve the node safety.

There is an issue how to set up default policies on the fly. Default policies should be restrictive but still usable.

There should be a very clear statement about who can set VE/service/component/port policies on the node. The principal capabilities are related to the VE: at the moment the principal can delegate those capabilities further to its sub VEs. So the capabilities are related to the relation between the parent VE and VE. Low level policies are always defined by the system; the only exception is when a component is transitioned under new VE or when the service is started under a VE. The default service policies are set by the owner of the VE; those policies can be overwritten by the owner by the meta policies set by owner, e.g. by definition of policies that are set by setPolicy interface of the component/service.

We have three levels of policies as described above. The low level polices are defined at the node start up (but should be able to change them if desired). Changing per VE should be possible but includes other parts of the system (labeling). Capabilities for the principal must be known at the time of the principal creation. Each new component that is started up must be checked that the component capabilities are equal or smaller than the principal capabilities. When the VE delegates the capabilities further to a new VE the same procedure must be followed.

The interfaces defined in the following table are related to class of dynamic policies; the exceptions are interfaces related to the principal capabilities.

Interfaces	
setPolicy	adds policy to the policy database
removePolicy	remove policy from the policy database
changePolicy	change the policy for a component
listPolicy	list the specified policy
searchPolicy	search policy by defined attribute
setCapability	set the principal or component capabilities
listCapability	list the principal or component capabilities
removeCapability	remove the capability from the principal or component list
searchCapability	search for a certain principal or component capability

Exceptions	
noPolicy	policy does not exist
policyExists	policy already defined
capabilityDoesNotExist	no such capability
capabilityError	could not set the capability

Structures	
tPolicy	policy
tCapability	capability

Interfaces in detail

setPolicy

setPolicy sets policy for accessing the certain component port. The inputs for the interface are policy type, policy and port. The relation of the policy and the component is implicit but has to be known. The port parameter can be a port type (if such exists, like monitor, manager etc.) so the same policy is applied to the all component ports.

Among policies we count also a capability list, e.g. list of capabilities that the component which wants to access this component has to posses.

The policy can be implicitly implied to a service as a whole including with port type note.

The call returns the policyId, a pointer to a policy. If the same policy is applied to many ports with port type as parameter the policyId return is same for all ports.

removePolicy

Remove the specified policy. Removing the policy causes that the default policies are enforced. The call takes as input policyId.

changePolicy

Change policy is dependent on policy type. In worst case use removePolicy and setPolicy.

listPolicy

List a port policy and return its human readable representation.

searchPolicy

search for a specific policy in the policy database. Search parameters should be various, like policyID, port, port type or various policy attributes.

Depends on type of policies defined...

Returns a list of policies together with ports and components...

setCapability

set the capability of the component. The capabilities are meaningful only to a processes or threads. Either takes a list of capabilities or a single capability.

If we adopt centralized approach the call returns a capabilityId.

Call also sets a capability list of a principal.

listCapability

List a capability list of a component

removeCapability

Remove certain component capability

searchCapability

Search for a specific capability in component, service or on the node

Structures in detail

tPolicy

tPolicyType

policy type identifies the type of a policy (which selects the authorization engine),

tPolicyString

string that represents a policy

tCapability

Capability is represented as string.

tCapabilityList

List of capabilities of one component.

SBB Setting Principals Capability

This SBB assigns a capability list to a principal. matchParentCapability checks that the assigned capabilities to a principal are possessed by its parent.

Pre-conditions

Principal is already created on the node and basic pVE software is already installed. List of available node capabilities exists and it is well understood in the current node context.

Post-conditions

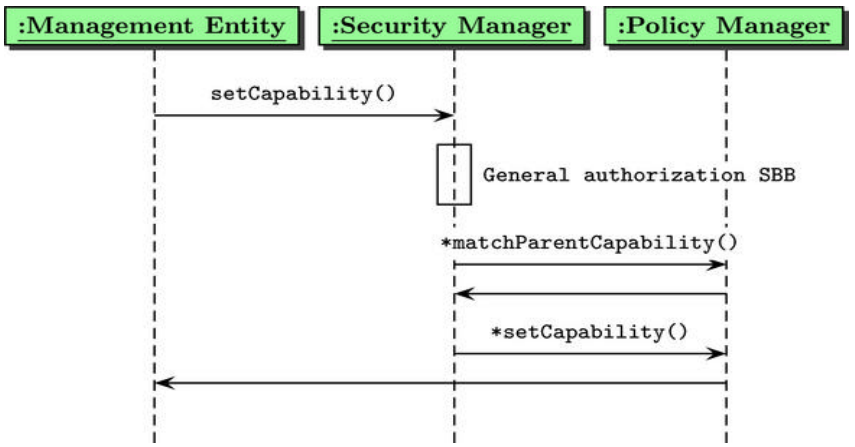
Principal holds certain number of capabilities in his capability list.

Dependencies

SBB depends on the following SBBs:

- add principal to credentials DB SBB,
- start pVE,

Sequence Diagram



SBB Set Component Capabilities

This SBB assigns a capability list to a component. matchPrincipalCapability matches available principal capabilities.

Pre-conditions

Principal is already created on the node and basic pVE software is already installed. List of available node capabilities exists. Principal holds a certain capability list.

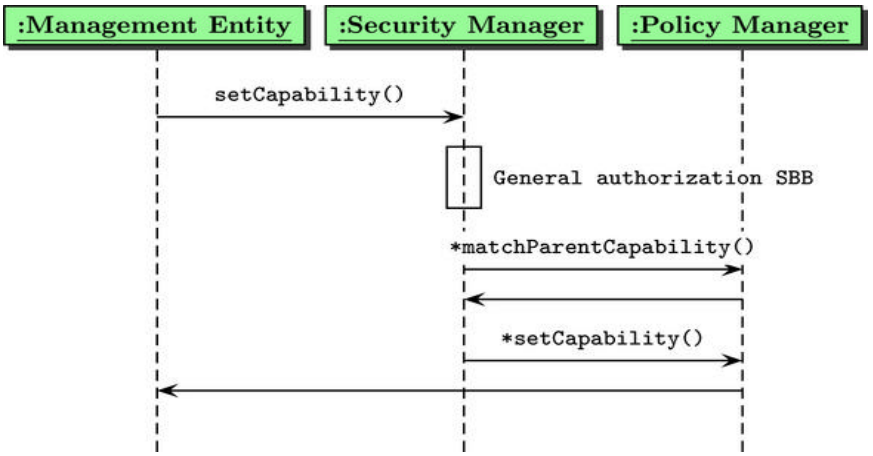
Post-conditions

Component is running with certain capabilities.

Dependencies

Many.

Sequence Diagram



SBB Removing Component Capability

This SBB removes a capability from the component capability list.

Pre-conditions

Principal is already created on the node and basic pVE software is already installed. List of available node capabilities exists. Principal holds a certain capability list. Principal component holds a list of certain capabilities.

Post-conditions

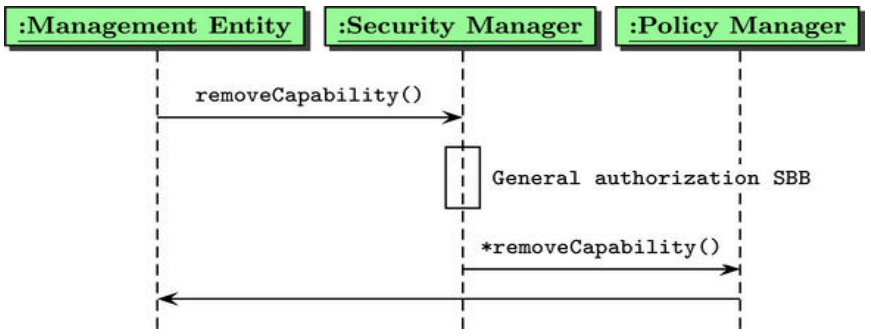
Component capability list is reduced.

Dependencies

SBB depends on following SBBs:

- setting a principal capability SBB,
- setting a component capability SBB**.**

Sequence Diagram



SBB Removing Principal Capability

This SBB removes a capability form the principal capability list.

Pre-conditions

Principal is already created on the node and basic pVE software is already installed. List of available node capabilities exists. Principal holds a certain capability list. Principal components hold a list of certain capabilities.

Post-conditions

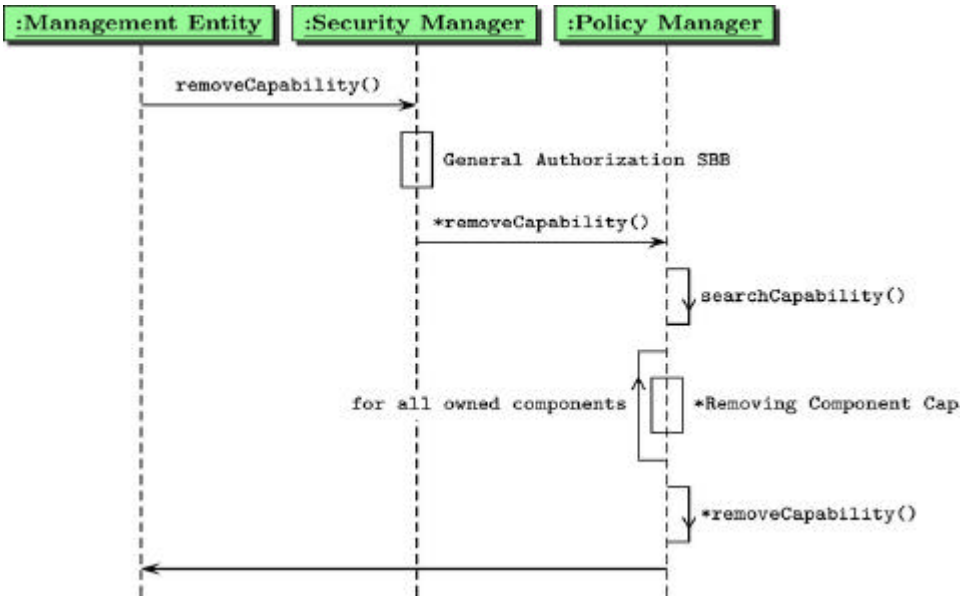
Principal capability list is reduced.

Dependencies

SBB depends on the following SBBs:

- setting a principal capabilities SBB,
- removing component capability SBB,
- general authorization decision SBB,

Sequence Diagram



SBB Add Component Policy

This SBB adds a component policy to a policy database.

Pre-conditions

Component should be already initialized. Component Id and the ports Ids should be known (management ports, control ports and in/output ports). Interconnection with other components should be defined.

Post-conditions

One of the component policies is defined.

Dependencies

SBB depends on exact definition and marking of the component ports and identification of the components on the node.

Sequence Diagram

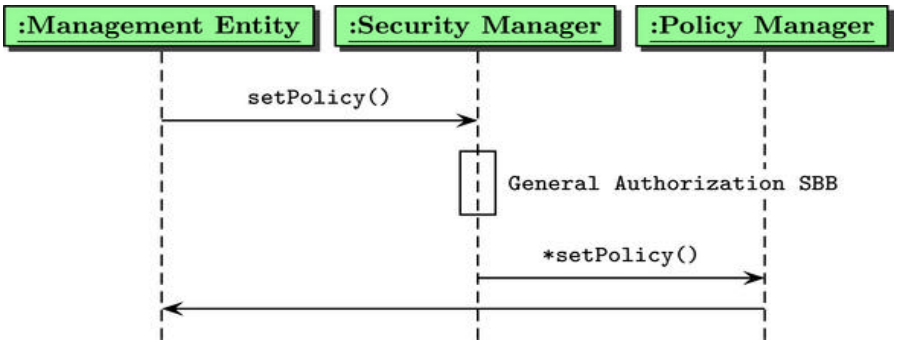


Figure 25: Adding a policy to a component

SBB Removing Component Policy

This SBB removes a component policy from a policy database.

Pre-conditions

Component should exist and the policy must be defined.

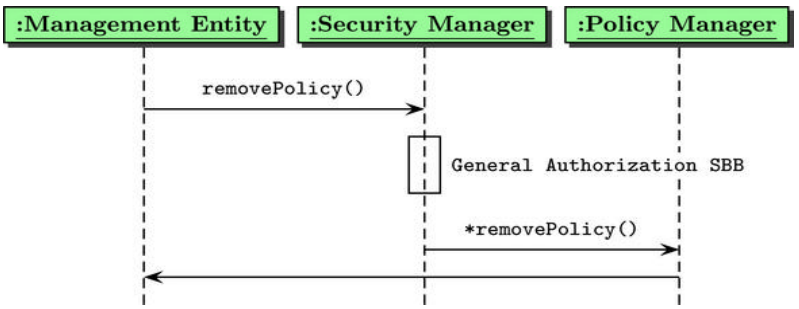
Post-conditions

One of the component policies is removed.

Dependencies

SBB depends on exact definition and marking of the component ports and identification of the components on the node.

Sequence Diagram



Naming Issues

Naming issues: internal and external. For example naming issues are essential to be able to define a policy for example for a port etc. Internal and external naming can be mixed? Mapping between both? For example how the management will define policies for the service that we will map internally?

Internal names

- names of components,
- names of ports,
- components interfaces (define NodeOS interfaces) to be able to correlate with capabilities

External names

- names of nodes,
- names of principals,
- names of services,

Authorization Engine

``is responsible for making a decision whether a given user request to execute specific action or to access/manipulate particular object within an active node is authorized or not. Authorization engine provides this Service'' to all enforcement engines in an active node.''

There are there general types of policies as defined in Policy Manager section. As we planned in D4 [], there can be several authorization engines on the node. In general case there are at least three engines which corresponds to described three types of policies.

Low level policies are default restrictive policies. Capabilities granted to the components are permissive policies. Service or components policies are principal defined policies which are permissive in the context of default restrictive policies (VE, service). In the context of authorization engines these three engines are stacked on the top of each other. This process is further explained in the security manager section.

On the other hand for service or components policies the policies are described by policy type which defines which authorization engine will be used.

SBB Low Level Engine

Authorization engine in this context decides whether the component accessing other component belongs to the same VE and service.

Pre-conditions

Basic components of the node are setup and run in the context of the pVE. If a component belongs to a certain VE its principal has to be defined. Components are already created and properly labeled or transitioned.

Post-conditions

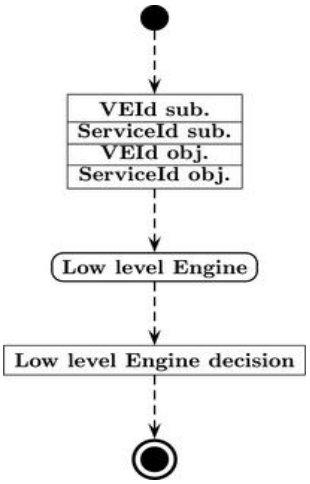
Low level authorization engine provides low level engine decision.

Dependencies

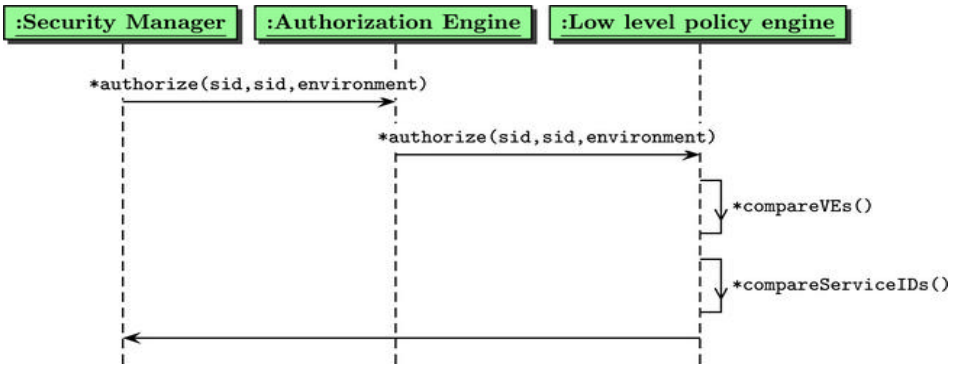
Depends on the following SBB:

- get security context by SID SBB,
- create security context SBB,
- Labeling decision SBB.

Activity Diagram



Sequence Diagram



SBB Capabilities

Authorization engine in this case decides if the certain component has the capability needed to access component or component port (interface).

Pre-conditions

Basic components of the node are setup and run in the context of the pVE. If a component belongs to a certain VE the principal capabilities has to be known. Principal has to determine which capabilities the component has. Components capabilities are set during the component start up.

Post-conditions

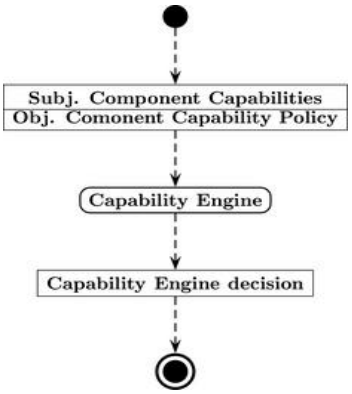
Access to certain component or component port is granted.

Dependencies

SBB depends on the following SBBs:

- get security context by SID SBB,
- create security context SBB,
- set component capabilities,

Capability Engine SBB



Sequence Diagram

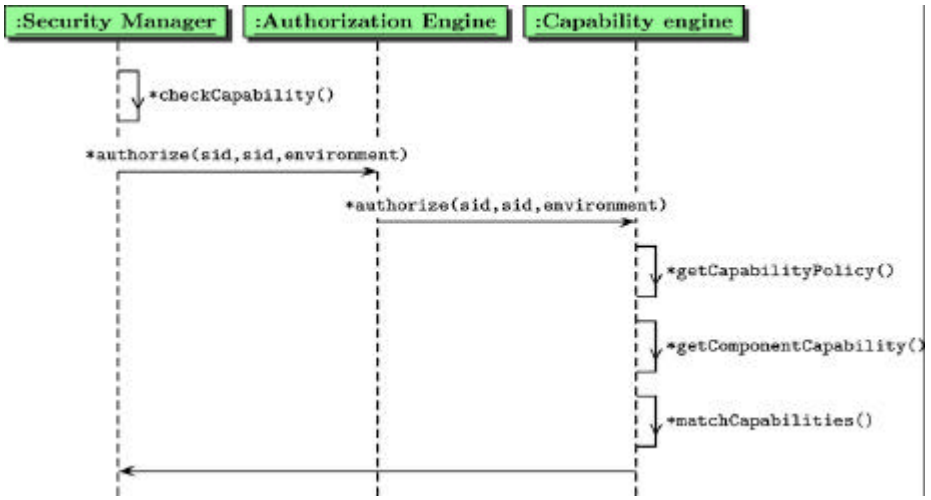


Figure 29: A process of providing a capability engine decision

SBB Policy

Authorization engine in this case 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. Though we said in D4 [6] that the Security Manager collects relevant policies to the call, these policies are usually defined at the service start up. Call check policy only evaluates if the policy exists for the context of the call (component or port policy) and then invokes authorization engine.

Pre-conditions

Basic components of the node are setup and run in the context of the pVE. If the component belongs to a VE, the VE has to be started and the principal has to be known on the node. The principal is responsible for setting the policies for the components or component ports. Such policies are used to fain grain control the access to the component ports. If there is no policy the access to the component port is decided based on the low level policy (VE and ServiceId)

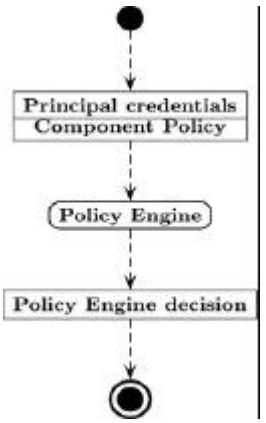
Post-conditions

Access to certain component or component port based on the defined policy is granted or denied.

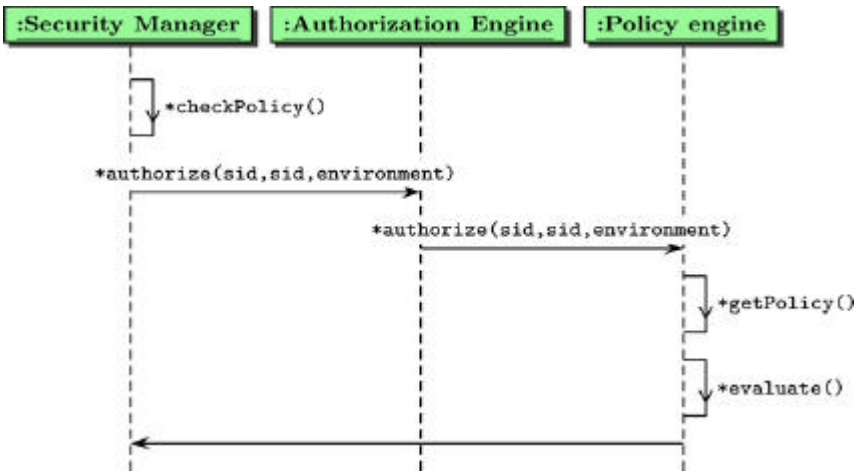
Dependencies

None.

Activity Diagram



Sequence Diagram



Connection Manager

Connection Manager is used to manage secure associations with neighbor active nodes. Associations can be configured manually or their configuration can be supported by automatic management and by triggering a key exchange protocol with neighboring active node.

There are two set of interfaces: first, that can be set by a management means to establish SA between two nodes and second that can trigger automatic key exchange protocol between two nodes.

Interfaces	
initSA	initialize SA data
storeSA	store SA data in persistent storage on the node
deleteSA	delete SA data on the node
stopSA	stop active SA
startSA	activate SA
listActiveSA	list the current active SAs

listStoppedSA	list stopped SAs
updateSeq	update sequences on peer nodes
changeSA	change the active SA for peer connection
exchangeSA	dynamically exchange and setup SA data

Exceptions	
updateSeqError	error in updating sequences
changeSAError	error in refreshing keys
exchangeSAError	exchange protocol failed

Structures	
tSAData	sending or receiving side SA data

Interface in detail

initSA

Initialize Security Associacion data object from tSAData. SA data is separated for receiving side and sending side. The reason for this is, that it is not necessary that the links between two peer nodes are always bidirectional (over same path) or that are indeed bidirectional. If the tSAData is transferred over the network the integrity and confidentiality of the data has to be provided. The data can access only Connection Manager or authenticated and authorized user.

storeSA

Store tSAdata to persistent storage on the node. Persistent storage should be a file or a database.

deleteSA

Delete tSAdata from the persistent storage.

stopSA

Stop the active SA. Data in the SA cannot be anymore used to validate the integrity property of the active packets received over peer connection.

startSA

Activates the initialized SA.

listActiveSA

Lists current active SAs.

listStoppedSA

List stopped SAs.

updateSeq

Updates sequence to the specified sequence value. Used in the cases when peer nodes have crashed and latest sequence number cannot be restored from the persistent storage. Interface function has two possible arguments, one that actually update the sequence in SA and the other that triggers a sequence update protocol with a peer node. Integrity of the exchanged data and data origin authentication must be addressed for the update.

refreshKey

Refreshes the symmetric key of the SA. Refreshing the key can be triggered because of the various reasons: policy can be set on validity of the key, because of the amount of traffic already transferred over the SA or any other reason. Interface function has two possible arguments, one that actually refresh the key with given value and one that triggers the refreshKey protocol with a peer node. Integrity of the data, data origin authentication and confidentiality of the key has to be addressed during the key refresh.

exchangeSA

Triggers key exchange protocol with the peer node...

Structures in detail

Secure Association data is defined with following parameters:

tSAData

send/receive identifier

Identifies that the SA is sending or receiving SA,

node identifier

Node identifier that uniquely identifies the peer node,

SA identifier

Identifier of the SA with the peer node, is unique identifier of the SA on the node, disregard node identifier,

symmetric key

Symmetric key used for building keyed hash,

algorithm

Algorithm used for keyed hash,

last sequence number

Highest sequence numbers of the packet received or send,

key start valid time

The time when the key becomes valid,

key end valid time

The time when the key is not valid any more,

status

Status of the SA that are active or that stopped.

Send/receive identifier reduces the number of exported interfaces. Node identifier uniquely identifies peer node. Planned identifier is cryptographic hash of the length 128 bits of the node private key. This identifier is used in protocols that update sequence, refresh key or trigger key exchange. It enables that multiple SAs are defined between peer nodes but only one of the SA can be active at the time. The length of the symmetric key depends on algorithm used; planned supported algorithms are either HMAC [7] or SHA-1 [8].

Protocols description

The idea here is as follows: define simple sequence diagram for the protocol and the messages exchange. Define the logic of the protocol implementation in LTL and convert it to code which defines protocol state machine. Implement the protocol as the set of protocol logic and protocol objects. define simple LTL rules for the set of concurrently running protocols and combine them in simple protocol group logic which controls all the protocols that are part of the protocol group.

The *protocol group* used to manage SA(s) data is defined as group of protocols working on common data set or task in this case on keeping the hop-by-hop integrity working in any situation. Planned protocols are:

Updating sequences

In some situations the node can lost the last sequence number for receiving the hop-by-hop protected packets for example in the case of node crash. In this case the node can request from the peer node(s) the last highest sequence number. To be able to do that the information about previous active SA(s) has to be available on the node or provided in other means. The updating sequence has two steps, a request and response:

$$A \rightarrow B: \ A, SA, C, H_k(A, SA, C) \quad (1)$$
$$A \leftarrow B: \ B, SA, C, Seq, H_k(B, SA, C, Seq) \quad (2)$$

A and B are nodes identifiers, SA is SA identifier, C is challenger cookie and Seq is corresponding highest sequence for particular SA of the node B. The communication is integrity protected by keyed hash H_k, where k is current active SA key. In exchange there is also protocol dependent identifier included in the exchange but not shown in the steps of the exchange.

SA change

SA change enables two nodes to change the current active SA. SA has to be initiated before and set to stopped state.

Key exchange

Key exchange is based on Diffie-Hellman key exchange. The protocol is similar to Station to Station protocol [] as it uses public key cryptography for entity authentication and provides mutual explicit key authentication. Protocol is defined in three steps:

$$A \rightarrow B: \ A, \alpha^x \mathrm{mod} p, S_A(A, \alpha^x \mathrm{mod} p) \quad (1)$$
$$A \leftarrow B: \ B, \alpha^y \mathrm{mod} p, H_k(\alpha^x, \alpha^y), S_B(B, \alpha^y \mathrm{mod} p, H_k(\alpha^x, \alpha^y)) \quad (2)$$
$$A \rightarrow B: \ A, H_k(\alpha^x, \alpha^y), S_A(A, H_k(\alpha^x, \alpha^y)) \quad (3)$$

A and B are nodes identifiers of the nodes exchanging keys, S_A is a signature of data send by node A, $H_k(\alpha^x,\alpha^y)$ is keyed hash of concatenation of α^x and α^y and k is shared secret key $k = (\alpha^y)^x \mathrm{mod} p$. y and x are random secrets selected by A and B and p and α are published in advance. The protocol has some overhead regarding original protocol but it reuses the general protection mechanisms as defined in FAIN framework (authentication with digital signatures). $H_k(\alpha^x,\alpha^y)$ provides key confirmation property. In real protocol some additional data is also signed like protocol dependent identifier.

Key exchange corresponds to exchangeSA. For now only the key is negotiated and other tSAData values have to be predefined in the node policy like the validity of the Key.

SBB Management Based Exchange

The SBB can be done as is presented on the sequence diagram. In this way tSAData is brought to the node, initialized and SA started. For the usable SBB actually two structures have to be initialized and started on each node (send and receive). Procedure has to be repeated for every pair of direct peers that communicate. For the data that is exchanged between nodes and the management station user triggering the action has to be authenticated on the node (actually his connection if done over SSL) and integrity and confidentiality of the data send is also issue.

Pre-conditions

The pVE has to be started including with loaded basic services.

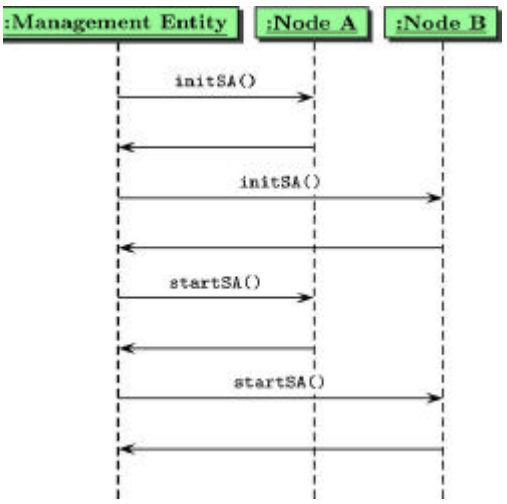
Post-conditions

The needed SA data is initialized and available to Integrity Engine which can start to accept or send active packets.

Dependencies

None.

Sequence Diagram



SBB Management Triggers Exchange

Management station triggers echangeSA protocol between the managed node and its peer. Procedure should be repeated for every managed node peer node. Management entity has to be authenticated on the node and its actions authorized.

Pre-conditions

The pVE has to be started including with loaded basic services. This includes nodes valid key pair and existence of the public key certificate.

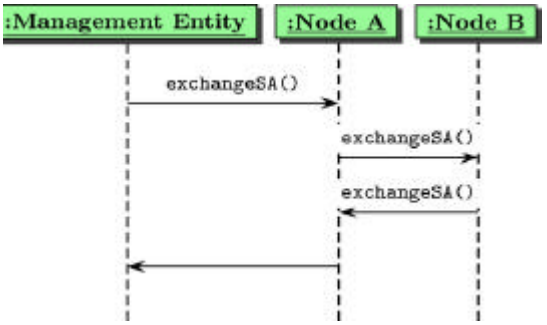
Post-conditions

The needed SA data is initialized and available to Integrity Engine which can start to accept or send active packets.

Dependencies

None.

Sequence Diagram



SBB Automatic Discovery and Exchange

SBB is triggered during the boot procedure of the node (or at the start up of the pVE) as is shown in sequence diagram. The node starts peer neighbor search protocol and triggers with discovered neighbors exchangeSA protocol.

Pre-conditions

The pVE has to be started including with loaded basic services. This includes nodes valid key pair and existence of the public key certificate.

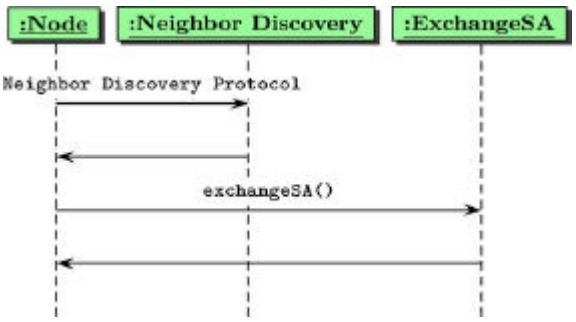
Post-conditions

The needed SA data is initialized and available to Integrity Engine which can start to accept or send active packets. Post-condition is valid for all found directly connected peer neighbors.

Dependencies

None.

Sequence Diagram



SBB Tearing Down SA

The last SBB for the Connection Manager is related to the ANN operation and plans the tear down of the active SA. SBB should show that the tear down of the SA will prevent the flow of active packets between two peer nodes. Bonus should be if it can show that the SA can be listed and activated again.

Pre-conditions

Active node should be fully operational and the ANEP packet flow should flow through the node.

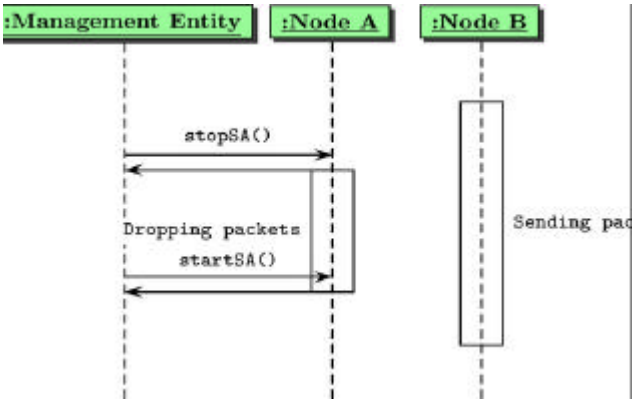
Post-conditions

All active packets coming from peer node to the managed node should be dropped. If SBB is extended, SA can be activated and active packets should be enabled to flow through the node.

Dependencies

None.

Sequence Diagram



5 GENERIC APPLICATION SCENARIOS

The generic application scenarios are composed of sequentially interconnected SBBs. The intention of using generic application scenarios is to have the functional concepts of FAIN reflected in an intuitive and enfolding manner. The generic application scenarios are therefore situated in cases that are in their requirements and demands close to reality. They are nevertheless 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

5.1 DiffServ Scenario

In the DiffServ application 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 5-1. 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.

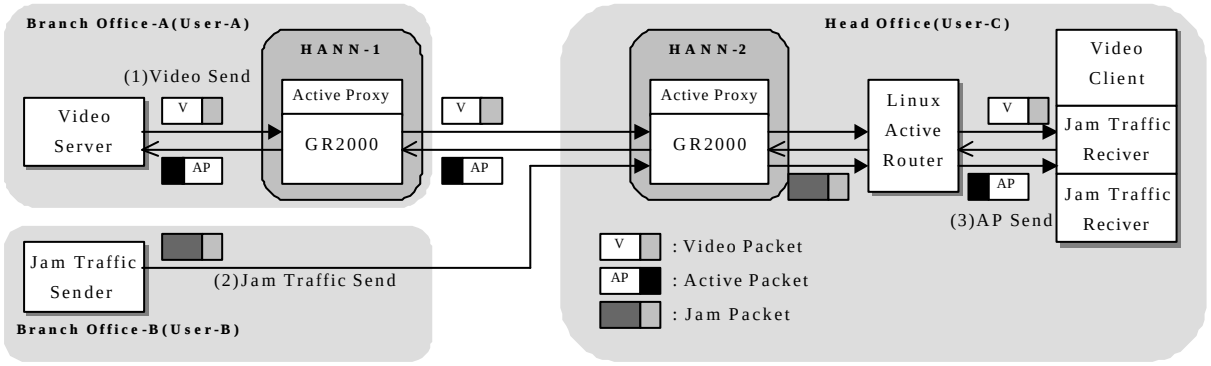


Figure 5-1: DiffServ Demonstration Scenario

Step-by-step Description

The DiffServ demonstration with GR2000 will be shown as follows:

1. The privileged VE instantiates the basic service components. (VE function)
2. The privileged VE creates a new VE creation with an allocation of resources. (VE function)
3. The RCF allocates some bandwidth and the DSCP codes to the new VE. (RCF function)
4. A jam traffic sender starts to send traffic and checks the allocated bandwidth.
5. The jam traffic sender stops sending jam traffic.
6. A video sender starts sending video data.
7. The jam traffic sender starts to send jam traffic again which intentionally creates network congestion.
8. The monitoring components detect packet discards at an active network node. (Monitor function)
9. The SNAP-EE injects an active packet; this is then signed, sealed and encapsulated as an ANEP packet. (SNAP/SEC function)
10. The component intercepts the ANEP packet and retransmits it to a Security component. (function)
11. The Security component checks the integrity of the ANEP packet, authenticates the data origin and sending principal, builds a security context of the packet and returns a verdict. (SEC function)
12. The accepts or discards the ANEP packet depending on the verdict. (function)
13. The retransmits the ANEP packet to the SNAP-EE. (function)
14. The SNAP-EE gets the GR2000 router's configuration and makes a request to the DiffServ Controller. (SNAP function)
15. Request is authorized by the SEC (SEC function).
16. The DiffServ Controller configures the GR2000 by setting a DSCP code. (RCF function)
17. The video and jam data pass through the GR2000 with their assigned DSCP codes.
18. The SNAP-EE sends a new SNAP-ANEP packet to the component. (SNAP function)
19. The sends the SNAP-ANEP packet to the Security component. (function)
20. The Security component inserts the necessary security information to the ANEP header of the SNAP and returns it to. (SEC function)
21. The component sends the SNAP-ANEP packet to the next active node. (function)

At the next active node, the procedures from 10 to 21 are repeated.

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

22. A jam traffic sender starts to send jam traffics at the Linux based router and congestion is occurred.
23. Steps 8 to 13 same as above.
24. The SNAP-EE request to VE Manager the creation of a Traffic Class, which will create a DSCP to bandwidth mapping. (SNAP function)
25. The VE Manager requests the creation to the Traffic Manager. (VEM function)
26. The Traffic Manager creates the Traffic Class. (RCF function)
27. The Traffic Class configures the Linux TC (Traffic Controller) in order to create the mapping. (RCF function)

5.2 WebTV

An SP wants to offer its customers (end-users) a WebTV service (cf. Figure 5-2). We call this SP, WebTV-SP and it broadcasts a video program in the Internet that end-users are able to watch, irrespective of their terminal capabilities. The WebTV-SP requests from the ANSP to set up an Active Virtual Private Network (AVPN) wherein he can deploy services that may be customized to meet customer requirements. Customers then subscribe to this WebTV service by directly contacting the WebTV-SP server. In this context, one of its customers uses a terminal that is not capable of displaying correctly the video stream of the WebTV content. For instance, this particular customer may use a handheld device with low processing power and a low access bandwidth. In this case, the WebTV-SP can individually select and process the video stream destined to his customer by deploying an audio/video transcoder in the network so that the video stream received by the handheld device is of the same format. As a result of an SLA agreed between the ANSP and the SP, policies are sent to the ANSP MI. Consequently, the ANSP PBNM receives a QoS policy and enforces it on both the NMS and in all appropriated 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 in all appropriated 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. The WebTV-SP is now ready to configure his AVPN by sending policies that are customer specific. The SP also installs the transcoder and duplicator service components. In addition, the SP deploys service-specific policies in the WebTV-SP PDP of its MI. 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.

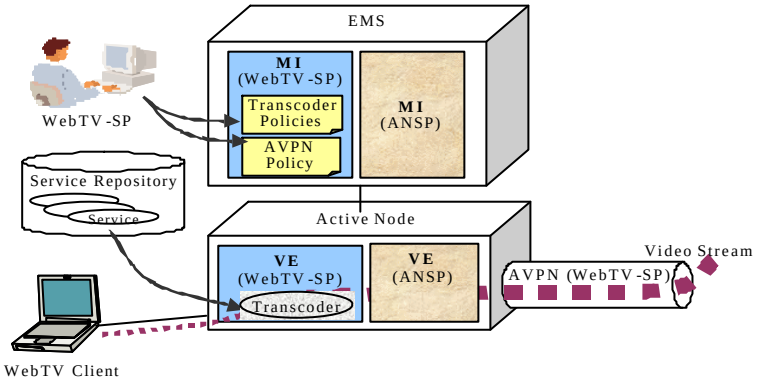


Figure 5-2: Web TV Scenario

Step-by-step description

1. WebTV-SP makes transcoder and duplicator available in the network. And also PDP/PEP. Which has been implemented using the same methodology followed to implement the ANSP management system?
2. WebTV-SP broadcasts a concert (Celtic music)
3. End-user connects to the concert portal (web page)
4. A SIP negotiation occurs between end-user and portal
5. The portal triggers the deployment of the Transcoder and the duplicator by contacting the ANSP-MS and giving it the parameters of the end-user (e.g. which video format it expects)
6. ANSP-MS receives a QoS policy enforced on both the NMS and the EMS
7. a VAN is created for the WebTV-SP and for this particular service.

8. ANSP creates a Management Instance (MI) in the management station the demo topology
9. After VAN is already activated ANSP-MS contacts Net ASP for deployment of the transcoder and the duplicator services.
10. Policies of WebTV-SP are sent and enforced in the appropriated EMS(the one who manage the active node where the Transcoder has been deployed): this results in the deployment of WebTV-SP PDP/PEP and afterwards the configuration of the service components
11. At this point it is possible to see the transcoded video at end-user sites

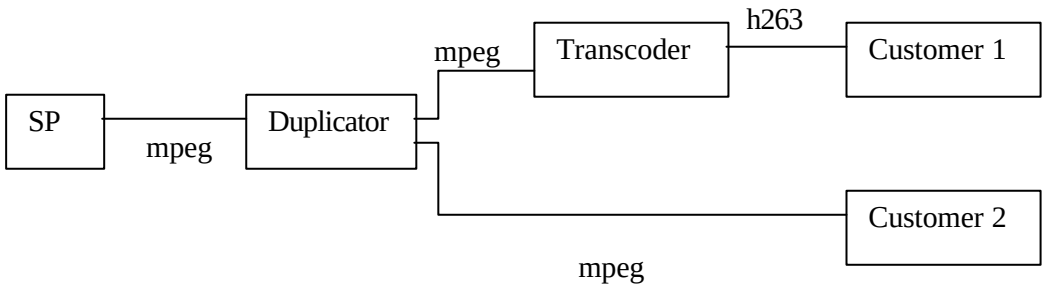
Extension:

A second Customer wants to be subscribed (C2)

12. SP contacts with ANSP-MS to trigger the deployment of a second Transcoder, but using the same VAN, if it is possible (if not the current VAN will be extended).
13. ANSP-MS contacts with Net-ASP to retrieve the topology requirements associated to this new service.
14. ANSP-MS generates appropriated policies (QoS and Delegation) and it enforces them into the NMS.
15. RM detects that VAN needs to be extended adding a new branch.
16. NMS send the appropriated QoS and Delegation policies to the appropriated EMS to modify current VE resource profiles and create/activate the VE required.
17. After VAN have been extended ANSP-MS contacts Net ASP for deployment of the second Transcoder Service.
18. Policies of WebTV-SP are sent and enforced in the appropriated EMS: This result in the deployment of the WebTV-SP PDP/PEP, monitoring probe if it is required and the configuration of the service component.
19. At this point it is possible to see the transcoded video at second end-user site.

Architecture/Setup

Topology:



The transcoder and the controller components are deployed via the ASP mechanism when requested in the SP's VE. In the final demonstrator, it will be triggered by the SIP proxy when the SIP request from the client (wishing to watch the WebTV) will be processed by the SIP proxy (see doc on GS1 for more details).

As a consequence of creating a VAN for this particular SP, a management instance is created and configured; there is one MI at network level and also one at element level. At the beginning only the PDP Manager will be instantiated inside each MI, after an incoming request is forwarded to this MI, the PDP Manager should decide to trigger the deployment of the specialized functional domain contacting with the ASP 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 should configured by means of policies, which will configure the monitoring system to receive those events/alarms from transcoder controller. These policies should have the next form:

Policy #1

IF

Packet Loss

LOWER THAN

20%

THEN

REMOVE VIDEO

Policy #2

IF

Packet Loss

LOWER THAN

10%

AND

CLIENT *WITH*_REMOVED_VIDEO == TRUE

THEN

RE-ADD VIDEO

The incoming request to the MI can be sent when the deployment of the transcoder is requested: the PDP/PEP and Transcoder/Controller are very close so we can imagine to deploy both at the same time.

The deployment request is sent by the SP (SIP Proxy) to the ASP-NMS Coordinator.

The latter can trigger the deployment of the SP's PDP/PEP at the same time.

Re-configuration of the transcoder

Up to now, one instance of the controller monitors the RTCP sessions (one for the Audio stream, one for the Video Stream) for one client.

Therefore there are as many controller instances as clients. A controller instance is created when a new client is added to the transcoder.

Upon instantiation the controller takes a list of thresholds as parameters. These thresholds represent the percentage of packets lost for the session. It is then possible to define several levels of QoS. In our example, there are only 2 levels, corresponding to the 2 actions: remove Video or re-add Video.

Up to now, the only re-configuration that is possible for one client is:

- To stop sending the video stream when the quality of the link is bad (for instance if the access networks is overloaded).
- To re-start sending the Video stream when the quality of the link is good again.

How the controller works

First, the transcoder is deployed and instantiated in an active node. When started, the transcoder opens 2 RTP sessions (one for Audio, one for Video) with the WebTV Emitter and does no transcoding (because there is no client yet).

Then the transcoder is configured to transcode the WebTV Emitter formats to the client format for this given client (Audio format, Video format, IP Address, IP Port, Size of the video).

When the client is added, an instance of the Controller is created. This instance is in charge of monitoring the RTP sessions between this client and the transcoder.

All monitoring information is carried within the RTCP (Real-Time Control Protocol) protocol that is associated with the RTP Protocol (RTP for the data themselves, RTCP for the control).

The RTCP ports are calculated automatically like this (add 1 to the RTP ports):

Video RTCP Port = Video RTP Port + 1

Audio RTCP Port = Audio RTP Port + 1

When this controller instance is created, it will receive the monitoring information sent by the client by listening on the RTCP ports.

To monitor the link, the number of packets lost is taken into account.

Based on the total number of packets received and the number of packets lost, a percentage of packets lost is calculated.

As seen in the previous section, when the instance has been created, parameters have been passed as arguments. These parameters help to determine at which level of QoS is associated the percentage of packets lost.

Interface controller – probe

After gathering the service configuration policies, the PDP accesses the monitoring system hosted in the ANSP management instance in order to register its interest in receiving the appropriate RTP events (delegation of functionality). The monitoring system uses the information contained in the associated filters to configure the data acquisition layer. As a consequence, the monitoring system will request the deployment of a probe in the target EE. The ASP is responsible for installing and connecting the probe with the transcoder controller. Appropriate access rights should have been set up (delegation of access rights).

Additionally, the monitoring system adjusts the threshold level for the packet-loss variable through the monitoring facet exposed by the controller. Such behavior makes it feasible a fine-grain control of the quality levels offered to the customers.

Finally, when the percentage of packets lost reach a threshold, then an event will be sent to the SP PDP:

The interface may look like:

```
module Events {
        // Exception definitions
        exception EventNotSupported {
        };
        // Low-level event definitions
        valuetype Event {
        };
        valuetype RTPEvent: Event {
                public controller::ClientParameters parameters;
                public unsigned short level;
        };
        // Interface definitions
```



```
        interface i_Probe {
                void notifyEvent(in Event event) raises EventNotSupported;
        };
};
```

Here level represents the level of packet loss. A level of 0 means that no packets are lost and a level of 1 means that the percentage of packets lost is greater than the threshold defined in the policy.

Such information will be packed in a CORBA structured event and subsequently delivered to the notification channel so that it can be distributed among the interested entities.

The resulting interface is generic enough for being used in different monitoring scenarios while at the same time it provides the necessary flexibility to define a wide variety of event structures to be sent to the probe.

Interface SP PDP - SP PEP

When PDP makes a decision about a policy it should pass this decision to the PEP. This one will be in charge of process the request.

That's the interface offered by the PEP:

```
module org{
 module ist_fain {
 module apbm {
 struct t_Parameter {
 string name;
 any value;
 };
 typedef sequence <t_Parameter> t_ParameterList;
 struct t_request {
 string command;
 t_ParameterList parameterList;
 };
 module pep {
        interface i_pep {
                oneway void decision (in t_request request);
 };
};//pep
};//apbm
};//ist_fain
};//org
```


When the PEP receives a request, it checks the action and executes it. The PEP should maintain a table with all controller//client references. How does the PEP should update this table? Does the Transcoder should update this table adding a new row when a new client is added? Or, does the PEP should check each Controller instance to know which Client is monitoring by looking up Client Property associated to each Controller instance.

The transcoder already maintains a table of clients (a transcoder instance manages its list of clients added but it doesn't know the clients of other transcoder instances) and it already checks if the video stream was off before re-adding it or was on before removing it.

Interface SP PEP – controller

When receiving a command from the PDP, the PEP must enforce this policy. It will result to an invocation to the controller interface.

The interface may be:

```
module org{
 module ist_fain {
 module services {
 module controller {
        struct ClientParameters {
                string IpAddress;
                boolean video;
                string videoFormat;
                short videoPort;
                string videoSize;
                boolean audio;
                string audioFormat;
                short audioPort;
        }

        boolean removeVideo (ClientParameters thisClient)
                // return true if OK
        boolean addVideo (ClientParameters thisClient)
                // return true if OK
        void setPacketLossPercentage(ClientParameters thisClient, short percentage)
                // sets the threshold above which the packet lost percentage is considered too high
 }
 } // services
 } // ist-fain
} // org
```


When the controller instance receives an action to process from the SP PEP, it will reconfigure the transcoder for this given client (remove or re-add Video), i.e. the client given as a parameter.

5.3 Web Service Distribution Scenario

For the full and extensive documentation of the Web Service Distribution Scenario please refer to [9].

In the Web Service Distribution Scenario, Web (HTTP) traffic is distributed within the network among several distributed servers in order to provide reliability, performance and scalability for web services.[11]

Motivation

Web technology has been the single most important technology responsible for the explosive growth of the Internet. It is not only the uniform technology that accommodates Internet surfing by end users, but it is as well mission critical for business applications both between and within enterprises. In this context, a number of requirements need to be addressed:

- To accommodate increasing numbers of customers,
- To provide scalable, reliable and efficient web services,
- To be able to quickly add new services and modify existing services,
- To reduce end-to-end traffic and server load.

To meet this diversity of requirements coming from customers and service providers we propose to design and implement an environment for web services based on Active Networks. In order to present our ideas, we first revisit the state of the art in web technology. Web technology (e.g., HTTP, HTML) has been developed originally as a pure client-/server architecture, where End Users are connected with the Web Service Provider site via an application-unaware IP network. Usually, such an IP network is composed of an access network for both the End User and the Web Service Provider site and an IP core network (Figure 5-3).

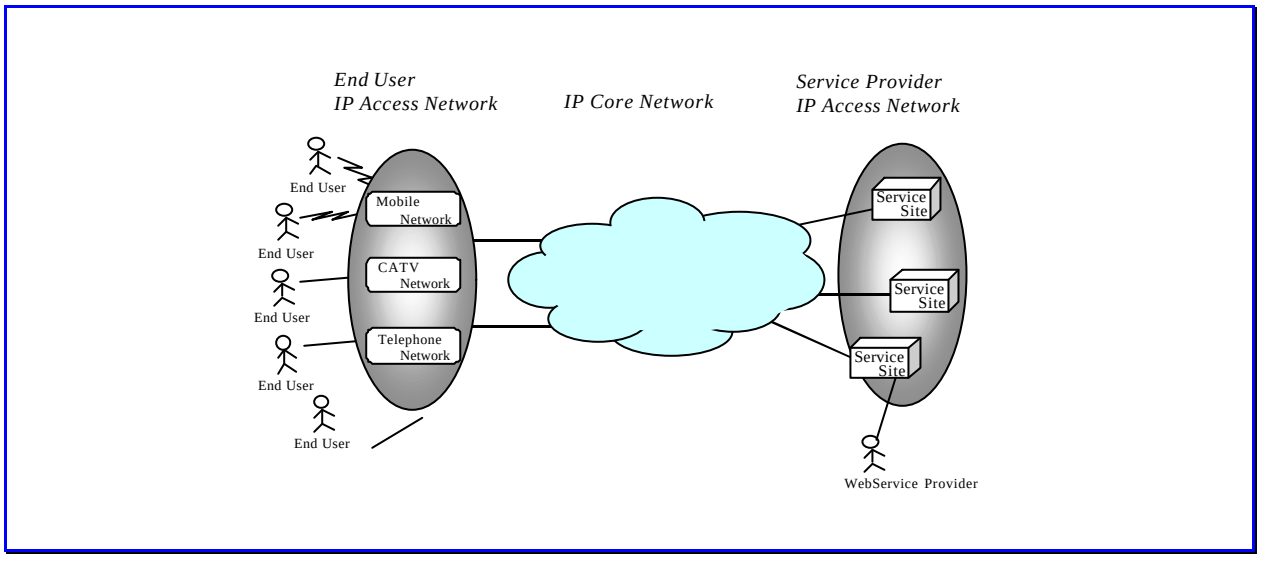


Figure 5-3: Physical Network Architecture of Web Services.

[11] 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 [10].

This simple approach, where the „intelligence“mainly is located in the Client and the Server, was one of the main reasons for the enormous success of the internet. However, it is also well-known that over the public internet, this pure client-/server model on top of a “dumb” IP network has shown to lack important characteristics, which are required for today’s and future Internet applications. These include reliability, performance and scalability as well as the possibility to take network internal conditions (e.g. the available bandwidth for a particular end-user) into account.

In order to overcome these deficiencies, the network infrastructures have already been extended in various ways. Some solutions which have been proposed and deployed in response to the growing demands of existing and new applications are: (see also Figure 5-4)

- Web caching [11], where static content is stored within caches within the network to reduce the response time for subsequent requests.
- load balancing/layer 7 switching [12], where requests are distributed to several servers in a server farm. To clients, this server farm appears as one “virtual web server”, which is reachable by one IP address, which in reality is served by multiple servers.
- content distribution networks [13], [14], where large volume content such as images or videos is pushed onto dedicated content distribution servers, which are distributed world-wide. These servers all have an own IP address and DNS name, and the traffic is redirected based on features included in the HTTP protocol.

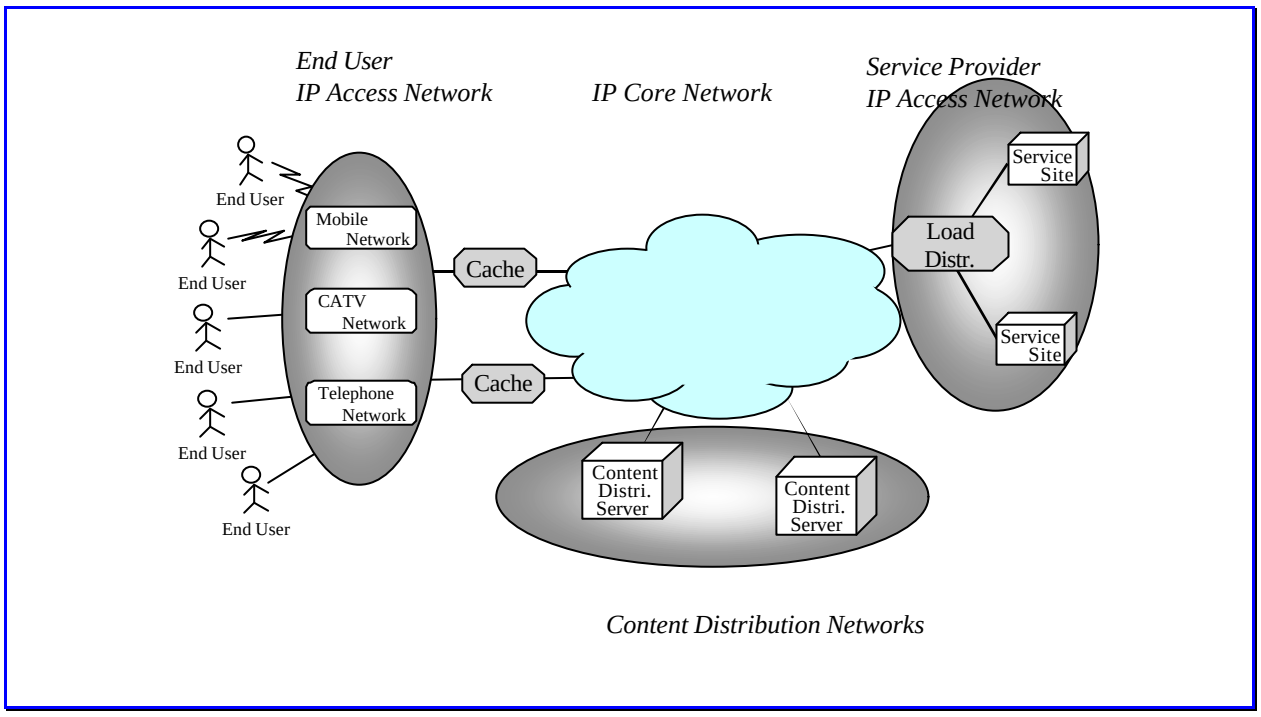


Figure 5-4: Web caches, content distribution servers and load distribution servers

However, these can only be considered as ad-hoc solutions to specific problems. For instance, caches only deal with static content. Important features of today’s web services, such as e.g. the observation of page hits of a particular page by the service provider or a third party, can still only be implemented by relying on centralized servers. We therefore believe that also today’s web services already can benefit from an architectural framework that can support an easy, rapid, and uniform way of deployment of new solutions that need “intelligence” within the network.

In addition, more network services are requested by customers and designed by service creators which clearly show that there is a demand for extensibility and flexibility on the ISPs and operator's infrastructure. [15] gives some examples of web-services which would clearly benefit from such a mechanism. A good example is a personalized *stock quote* service. Stock quotes are frequently changing data which can hardly be cached. If in addition a service for distributing stock quotes should be personalized (e.g. with respect to the portfolio of a certain user), it becomes inevitable that with current approaches a high network- and computing load is generated on a centralized server. A distributed architecture would permit personalization and information distribution within the network, being clearly more scalable.

Architecture

The basic idea of our active web service infrastructure is to exploit the capabilities of activeness in the following two ways:

- On the one hand, so called "service nodes" within the network is implemented using AN technology. These service nodes are "Active Web Servers", which can be programmed by the web service provider. They provide for instance persistent storage in order to allow the service provider to store content locally and an environment to execute web service logic (e.g. JavaBeans). The service provider downloads content and service logic onto them and by this way can implement features such as content distribution and personalization, aggregation of user replies or fast response times.
- On the other hand, so called "redirection nodes" are also implemented using AN technology. They provide features which allow filtering out 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.

Of course, both kinds of functionality may be combined within one physical node, and there may also be nodes which provide a functionality which is a mixture of both. However, we usually expect them to be separate, both physically and logically. On the one hand, programming a network (e.g. load balancing algorithms, routing algorithms) requires different skills than programming web services and hence will be carried out by different actors. Second, both kinds of nodes will usually be implemented with different design objectives: While redirection nodes will be some kinds of routers with an emphasis on network throughput, service nodes will require at least some part of functionality of web servers (i.e. huge computing power, persistent storage).

Also note that both kinds of activeness are complementary and can be implemented independently: The redirection of traffic by redirections can also be beneficial if a distributed set of web servers is implemented and operated completely independent of AN technology. Vice versa, a distributed, AN-based implementation of a web service infrastructure can also be used with conventional techniques for load balancing. In the sequel, we call any web service which is implemented using some kind of activeness an "Active Web Service".

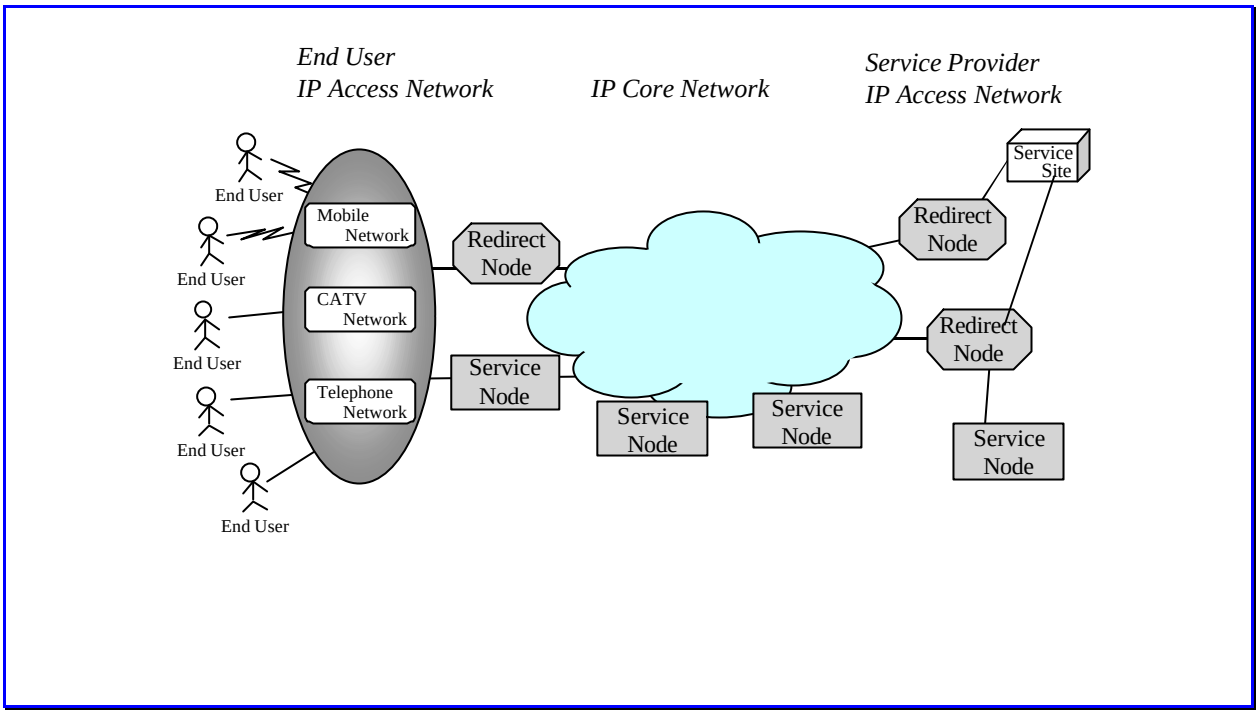


Figure 5-5: Service nodes and redirect servers implement active web services

The overall scenario is depicted in [9]. Both redirection nodes and service nodes will be usually 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.

One design goal is that 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. This assumption takes into account that updating several hundreds of millions of web clients is not only infeasible, but also unnecessary.

To sum up, the proposed application can be viewed as a continual evolution of existing web infrastructure:

- Traditionally, the web has been based on a "dumb" IP network, offering only pure best-effort IP routing and forwarding capabilities.
- Recently, the web has been enhanced by installing caches, switches and content distribution networks. However, the "intelligence" (i.e. the service logic) still resides on centralized servers operated by the web service providers.
- In the future, the web might by a fully distributed computing infrastructure, with service intelligence distributed on the IP and/or on the application layer.

Justification of the Use of Active Network Concepts

We expect that with Active Network technology web traffic can be handled flexibly within the network. Potential benefits 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. 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.

Overview

In order to minimize the necessary changes for the web servers and web clients, we make minimal assumptions about web servers and clients for our demo scenario:

- It should be possible that web servers work isolated; i.e. no additional mechanisms should be required which are implemented by the web servers. In particular, web servers can use a local date base, e.g. to store data used to process sessions.
- The service provider can use a standard web server (e. g. apache).
- Clients can use standard browsers to access the web server.

This implies that the used protocols must not be changed in any way and the mechanism is fully transparent both to web servers and web clients. To be more precise, the implemented mechanism can roughly be described as follows:

The customer issues an ordinary HTTP request to the system, containing the static IP address which it has looked up by the DNS service.

The system forwards the request through a number of routers. The routers exchange routing information for that purpose, as well as they provide basic IP connectivity via that interface.

Eventually, the request reaches an active node which contains redirect logic. The active node filters the HTTP request and forwards it to an execution environment operated by the NSP.

The execution environment continuously collects information about the availability and load of service nodes. Depending on some strategy, the incoming HTTP traffic is distributed among the available service nodes.

The HTTP request is executed at a service node. The execution may involve access to stored content and computations of the service logic.

A central concept in web services is that of a *session*, i.e. of an association of state between client and server. While the HTTP protocol is stateless, sessions are implemented by various techniques, such as encoding state in URLs or cookies. The concept of “session” has the following implications on our implementation:

- Requests which are not part of a session: each of these requests may be redirected independently.
- Requests which are part of a session: all requests which are part of a session must be redirected to the same web server due to the fact that we do not want to transfer state between web servers.

Objectives

In this section, we describe the relationship of our demo scenario to the FAIN evaluation framework. It is intended to map our demo scenario onto the FAIN architecture and the FAIN evaluation framework [16], [17].

Demonstrated Architectural Concepts

The following concepts are demonstrated by this demo:

- Creating Virtual Environments as Part of the Virtual Networks Creation
    - This demo does not use Virtual Environments. Creating Virtual Environments is part of another demo described in **[18]**.
- Resource Control for hard Resource Partitioning
    - Resource Control is not part of this demo.
- Deployment of different Types and Instances of EEs
    - Manipulation of PromethOS modules using an extended version of Iptables provided by PromethOS. This provides an intermediate step towards a full integration of PromethOS plugins into FAIN execution environments. A next step will be described in **[18]**.
- Creating and Operating Component-based EEs
    - This demo will use only one EE on one node.
- Interoperable Infrastructure
    - Not used.
- Creation of a new VN Management Domain as Part of the VN Creation
    - Not used.
- Use of new VN Management Domain to manage Services and Resources
    - Not used.
- ASP Specification and Deployment
    - Not used.
- Tuning the Active Network for maximizing Performance
    - Using PromethOS kernel modules for processing HTTP traffic will be a step towards maximizing the performance for processing web traffic.
- Using Active Networks for Policy Distribution
    - Not used.
- Simple fault management functionality
    - There is an automatic reconfiguration capability which will remove shut down web servers automatically from the list of allowed targets. When the server becomes available again, it will be reactivated automatically after some time. As long a there is at least one working web server, the end user will see a usable network (with maybe degraded performance of course).
    - The interactive configurability of PromethOS plugins enables a service provider to react promptly to changing requirements (e. g., web servers which must be shut down, increasing load, etc.).
- Network-level deployment mapping

- Not used.

- Active Network Upgrades
  - PromethOS allows the dynamic loading and unloading of plugins. This is a first step towards a dynamic upgrade of an active node.
  - There are currently no provisions for a seamless upgrade without interrupting a running service. There will be at least a short break when the old plugin is unloaded until the new plugin is loaded and configured.

Contribution to the Evaluation

- Flexibility Property
  - dynamic loading and unloading of PromethOS plugins
  - interactive configurability of PromethOS plugins
- Security Property
  - currently relies on the fact, that PromethOS plugins can only be loaded by root. For a better integration into the FAIN security model see **[18]**.
- Portability Property
  - PromethOS plugins can be deployed only on active nodes running PromethOS
  - PromethOS itself is based on Linux
- Reliability Property
  - AN Concepts are used to in this scenario to implement reliable Web Services
- Performance
  - throughput of web traffic to be measured
- Interoperability Property
  - ---
- Timeliness Property
  - end-to-end delay: no hard requirements, should be sufficient for interactive use.
  - hard to measure anyways, as it depends on many factors (delay imposed by the web servers, delay on active nodes, delay on non active nodes passed by packets, etc.).
- Openness Property
  - Standard web protocols are used and therefore the approach is “open” to any web user or web service provider

5.4 Video on Demand Scenario

In active network nodes, PromethOS is going to play a role as a Linux kernel-space NodeOS. As such it is of major importance that PromethOS is able to be managed easily from Execution Environments (EE) in user space.

In FAIN, the Virtual Environment Manager (VEM) has been developed. VEM, an EE in user space, is concerned with most of the node's management issues like service deployment and the like. An integration of VEM and PromethOS is thus required to show interoperability between different types of EEs

In FAIN an Active Network Node (ANN) has been designed to support multiple VEs, EEs and EE-instances. VEs may include several EEs, in user as well as in kernel space. VEs are the major component involved for customer differentiation, i.e. a customer is identified by its VE. A VE serves as a resource container. Supporting multiple VEs, multiple customers can be run on a single node without interfering each other.

PromethOS has been extended to handle several VEs to differentiate among customers; several EE-instances in kernel space; communication among them; and communication among the EEs involved used in FAIN. The VEM has been enhanced to control PromethOS according to the management interface offered by the PromethOS user space library.

This generic application scenario shows the integration of the VEM and PromethOS in order to proof the proper interaction between different EEs. The VEM will be able to control PromethOS. Several VEs will be instantiated, resources assigned for operational control (however, no resource limit-enforcement will be carried out in this scenario). In a VE, at least one EE will be instantiated in kernel space in which the Wave Video plugin as demonstrated at the Barcelona Review Meeting will be run. Thus, the demo will present the Wave Video scaling functionality as explained in D4 according to customer's different requirements.

Architecture/Setup

An active node and a source for a video stream are located at one site (e.g. a service provider). At a different location, the video receiver is installed. Between the source and the active node, a high-bandwidth link provides sufficient bandwidth. This link models a high-bandwidth backbone. Between the active network node and the video receiver two different links are emulated. These links provides different capacities according to the bandwidth allocated by the service level agreements among the network provider and the customers. 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. A customer request arrives at the ANN signalizing the VEM to deploy a VE, assign resources to this customer, instantiate an EE, and deploy a Wave Video scaling plugin into this component.

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 provided by WP4. The required Wave Video plugin code will be fetched from the WP4 code server and deployed on the FAIN ANN.

A second request submitted by a different customer arrives at the ANN to create second instances of this configuration with different resources assigned to the VE created. Node additional code fetching is required since the code is still available in the local cache of the node.

The creation process is implemented and controlled by the VEM fully. As soon as this process completes, the video source is signaled to start transmission of the video flows. A small helper application simulating a Video on Demand source receives the request and serves the video out of the huge database of different videos.

Objectives

By this scenario, a video transmission over the Internet has been implemented. Customers are connected to the backbone of the Internet (the ETH ANN) by links with different capacities. The management process of the data path based video scaling in the Wave Video plugin is handled fully by the control plane of the FAIN active network: A service specification of FAIN's WP4 arrives at the FAIN ANN; this specification is processed by the service creation engine; the VEM is responsible for service deployment on the node and service configuration. PromethOS provides the in-kernel EE and the ability to support different customers.

5.5 Managed Access Scenario

This part of the demonstration makes use of active packet technology to manage packet filters across networks that have different owners.

Assigning Access Network Services: source host policy

In this scenario, a host joining a network is assigned a set of network services according to some arrangement specific to the host. Here for example, all the road warriors get access to a DNS and a web cache, but only road warriors 2 and 3 get access to a local SMTP server and may access remote POP servers. Only road warrior 3 is allowed to use the IPSec protocols to establish tunnels.

	Services				
Road Warrior	DNS	Webcache	SMTP	POP	IPSec
1	1	1			
2	1	1	1	1	
3	1	1	1	1	1

Separated Provisioning: Access and Service

As a use-case this is a simple two-stage, two-party scenario

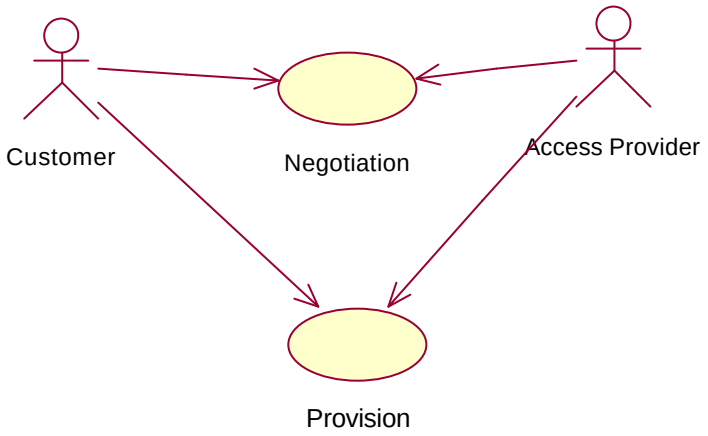


Figure 5-6: Use-case: customer and access provider

This is the way most networks are managed. The access network provider can only manage his own equipment and assumes that the customer has his own arrangements with the rest of the network, the service providers.

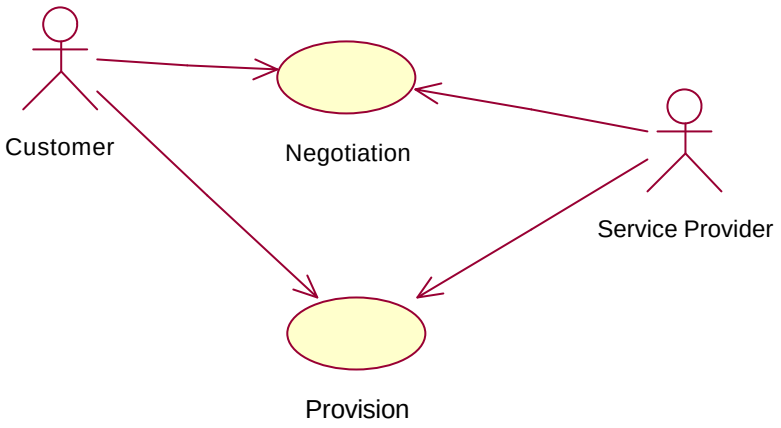


Figure 5-7: Use-case: customer and service provider

The negotiation activity is a sign-up procedure. The technology needed to implement the provision activity is given in Figure 5-8: Deployment: Access Network.

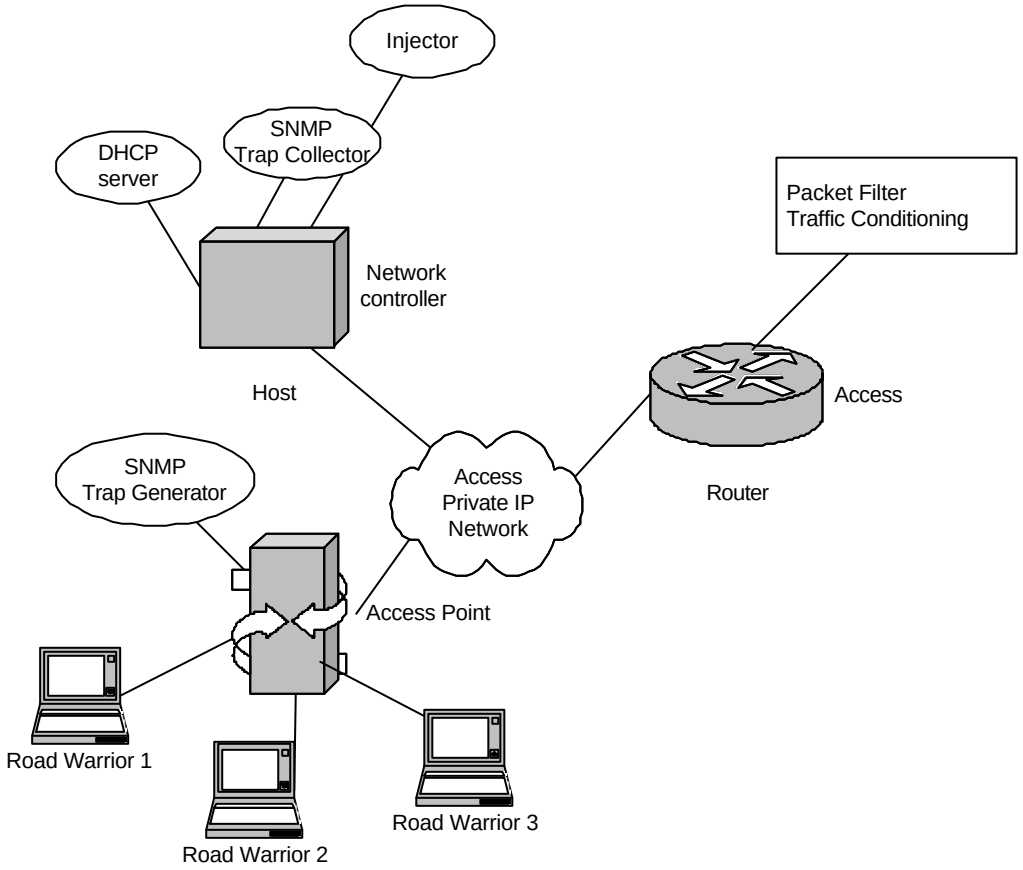


Figure 5-8: Deployment: Access Network

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 remainder of the network is shown in Figure 5-9: Deployment: Backbone and DMZ. 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.

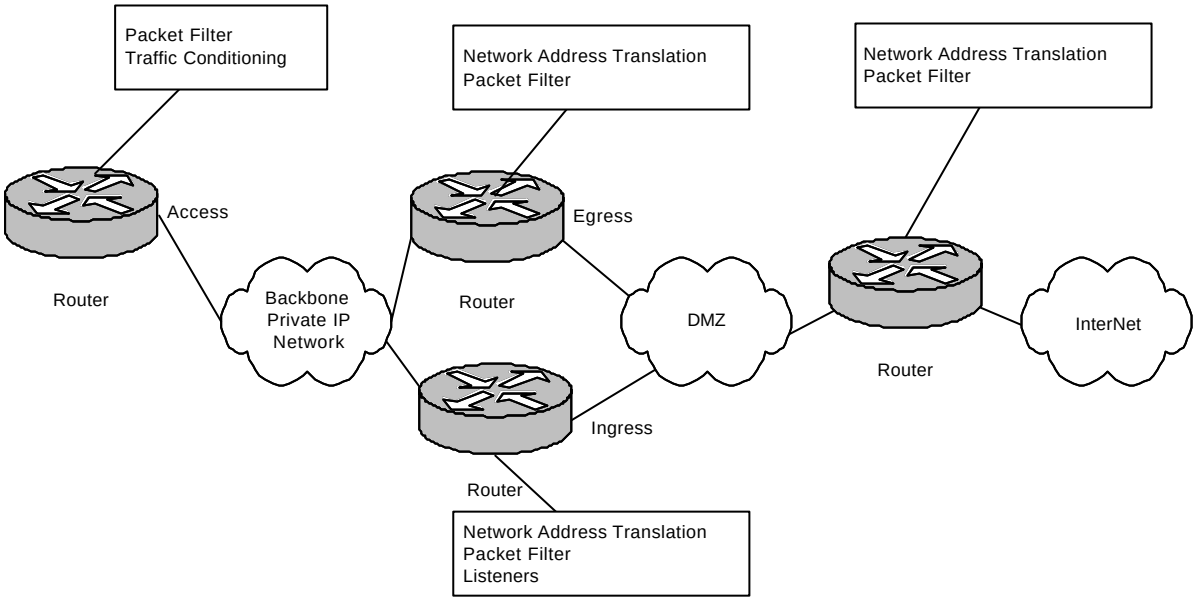


Figure 5-9: Deployment: Backbone and DMZ

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.

Integrated Provisioning: Access and Services

In this scenario, the access provider, the service provider and the customer jointly negotiate the services to be provided. This then becomes a tri-partite activity.

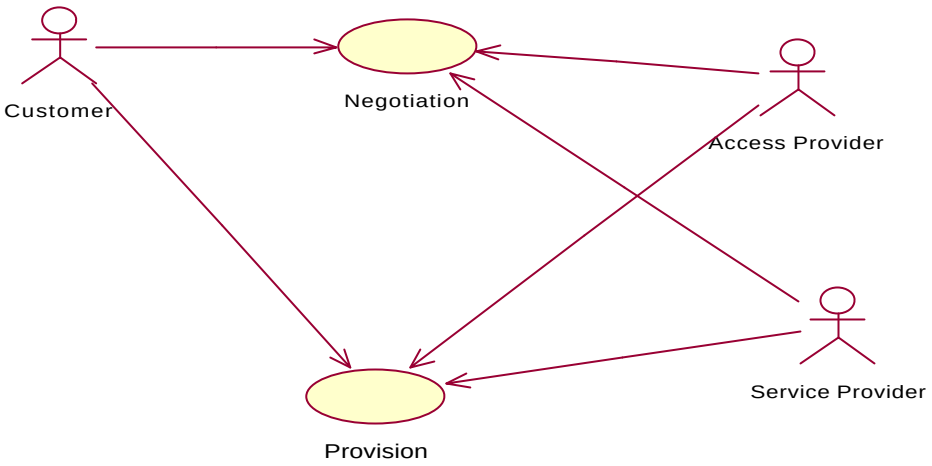


Figure 5-10: Use-case: access and service provisioning

This scenario would need an active router for the access network. Every time the customer sends a packet to a service he has not previously accessed an active packet is dispatched that attempts to configure the local network and the remote network to provide the service. The network would be the same as that given in Figure 5-8: Deployment: Access Network, but the router of the access network would be able to intercept packets and dispatch active packets. It is unlikely that this scenario can be implemented in the lifetime of the project, so it is best ignored for now.

System Design

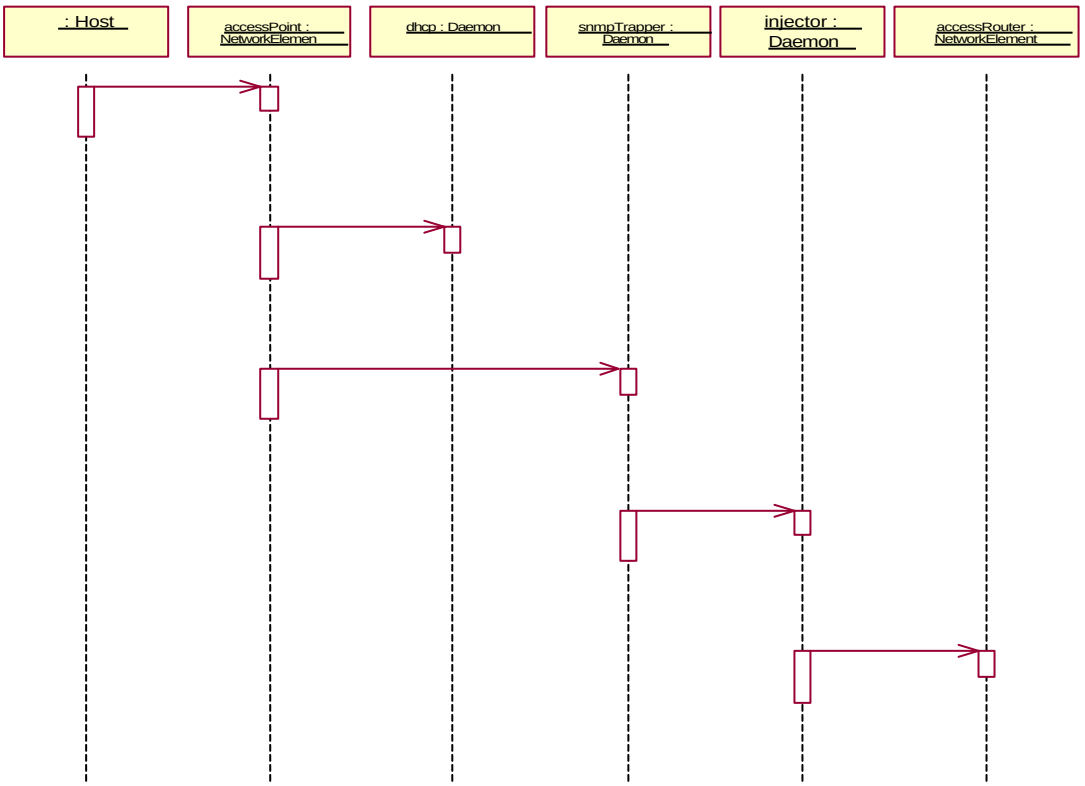


Figure 5-11: Access Network Operations

The host notifies the access point that it wants an IP address. The access point passes this to the DHCP daemon running on the network controller and, on success, raises an SNMP trap. This is collected by a daemon which injects a packet which is passed to the router.

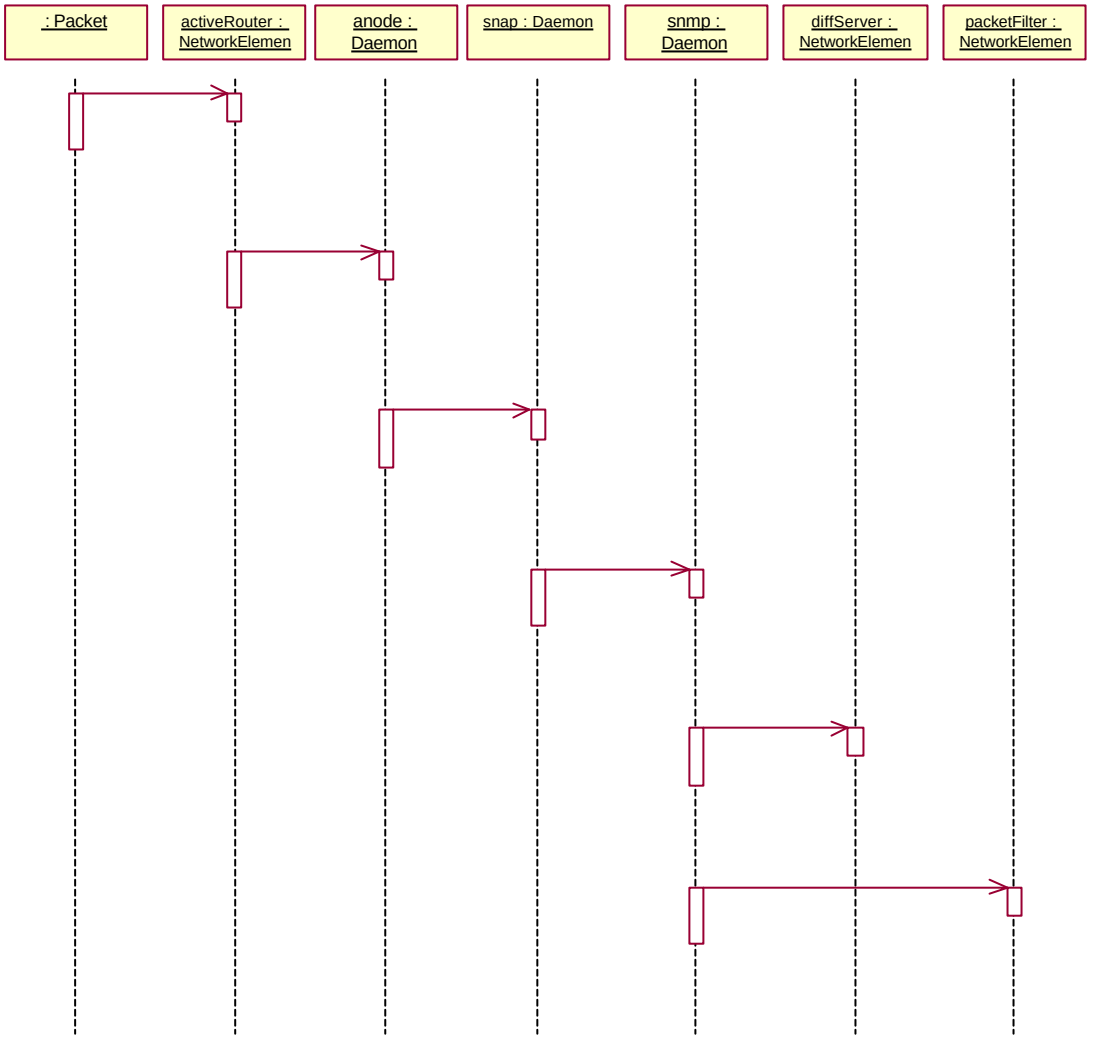


Figure 5-12: Network Element Control

A packet arrives is trapped by the active router which passes it to the active node daemon. Its component detects it as a SNAP packet and passes it to the SNAP daemon, which decodes and executes the instructions. These are sent as SNMP commands to the SNMP daemon. The daemon then executes the policy implementation to program the packet filter and the DiffServ routers.

5.6 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 [19]. It describes the idea “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 show 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 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 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 during registration.

1. **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.
2. **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 generic application scenarios 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:
- 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.

5.7 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 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 (service) 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,
9. The 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 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 and in order to be sent to the next hop.

6 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.

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

6.1.1 AN Node Type A

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

6.1.2 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.

6.1.3 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 will be based in PCs with Linux OS. As programming platforms both stations need OpenORB CORBA platforms over which management components will be build.

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).

6.2 Network Topology and Interconnection

6.2.1 Testbed topology

Figure 6-1 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.

The complete network is shown in Figure 6-1.

6.2.2 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.

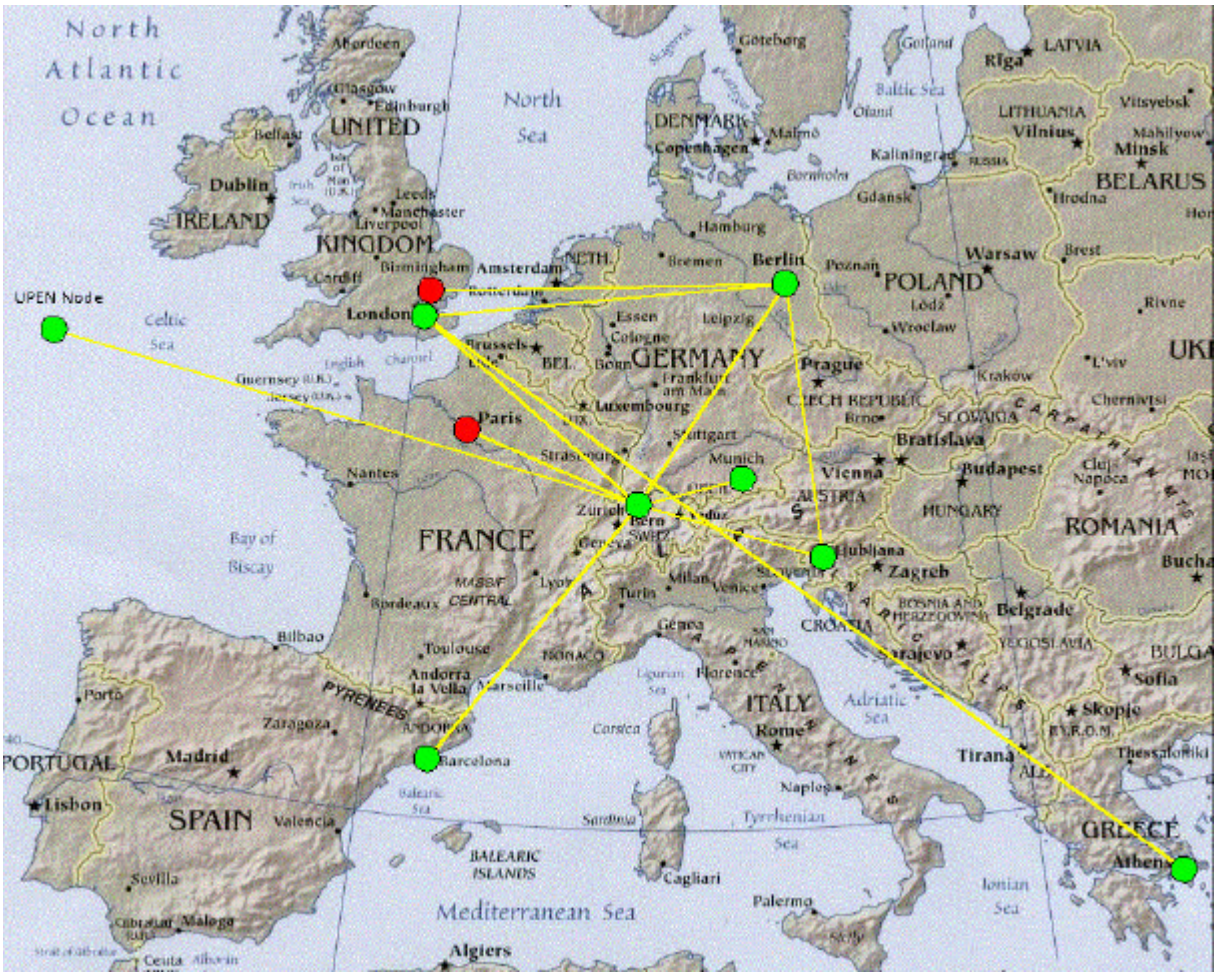


Figure 6-1:Testbed topology

6.2.3 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.

6.2.4 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.

6.2.5 Sites overview

Figure 6-2 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.

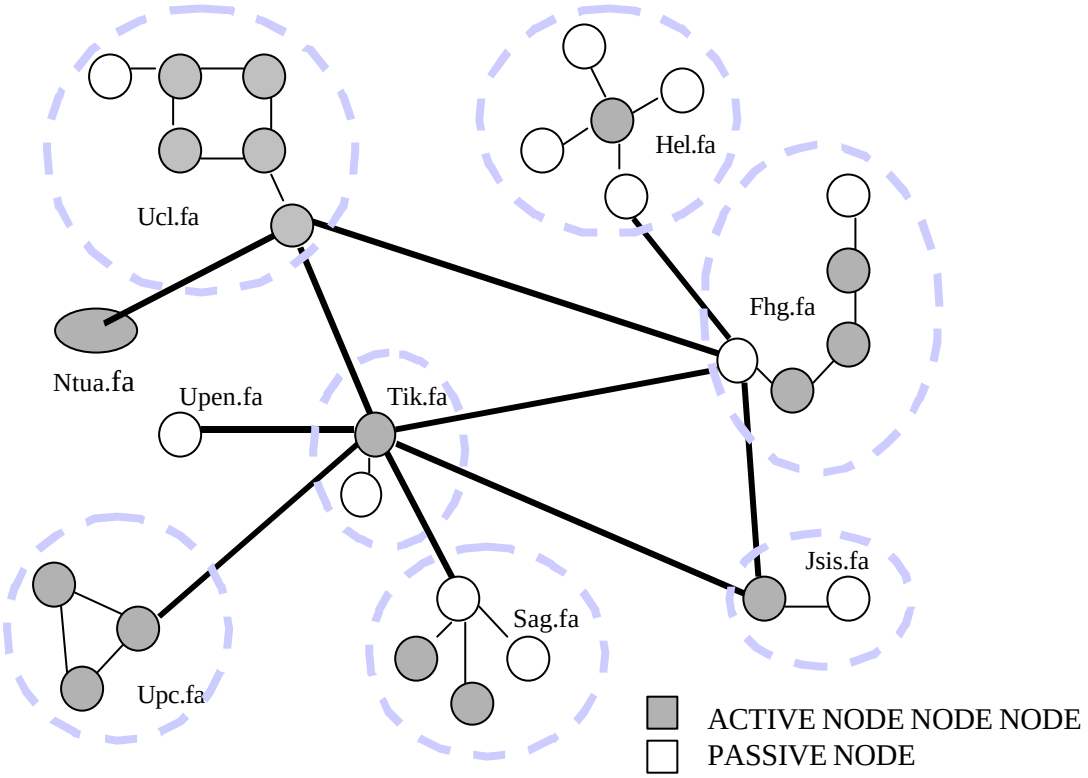


Figure 6-2: Nodes Overview

6.2.6 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.

7 APPLICATION SCENARIOS

The application scenarios are the same as generic application scenarios that are mapped to a specific infrastructure. As told previously, the generic application scenarios are similar to a declaration in programming languages, whereas the application scenarios are their respective instantiation.
Since the application scenarios are supposed to be shown as demonstrations of FAIN results they are often referred to as demonstration scenarios. The terms are used interchangeable in the following sections.

7.1 DiffServ Scenario

The detailed version of the DiffServ Scenario can be found at [25]. 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 5-1. 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.

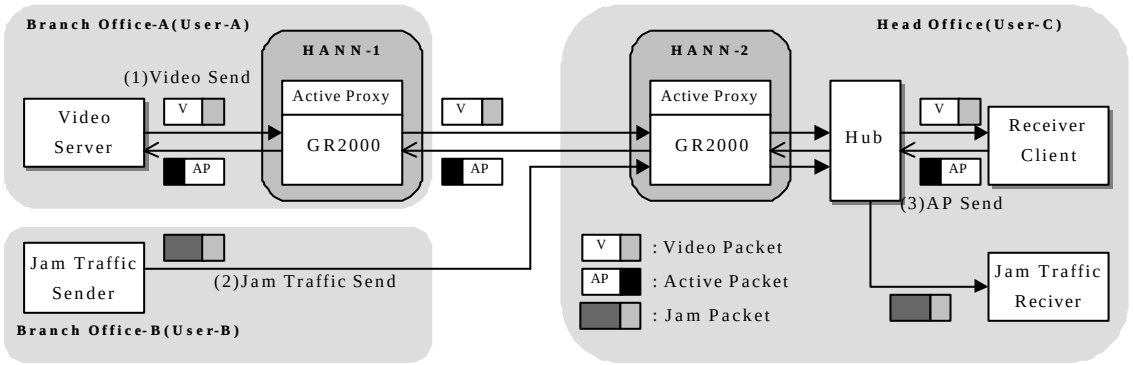


Figure 7-1: DiffServ Demonstration Scenario

7.1.1 HEL Test-bed Configuration

The Figure 7-2 shows a current network configuration of the HIT/HEL test-bed. For simplest, we used only one HANN for checking of the DiffServ function in our test-bed. In addition, we implemented the video receiver and the jam traffic receiver into one terminal. Besides, we used 10Mbps bandwidth for the output of the HANN, especially G4 port of the GR2000.

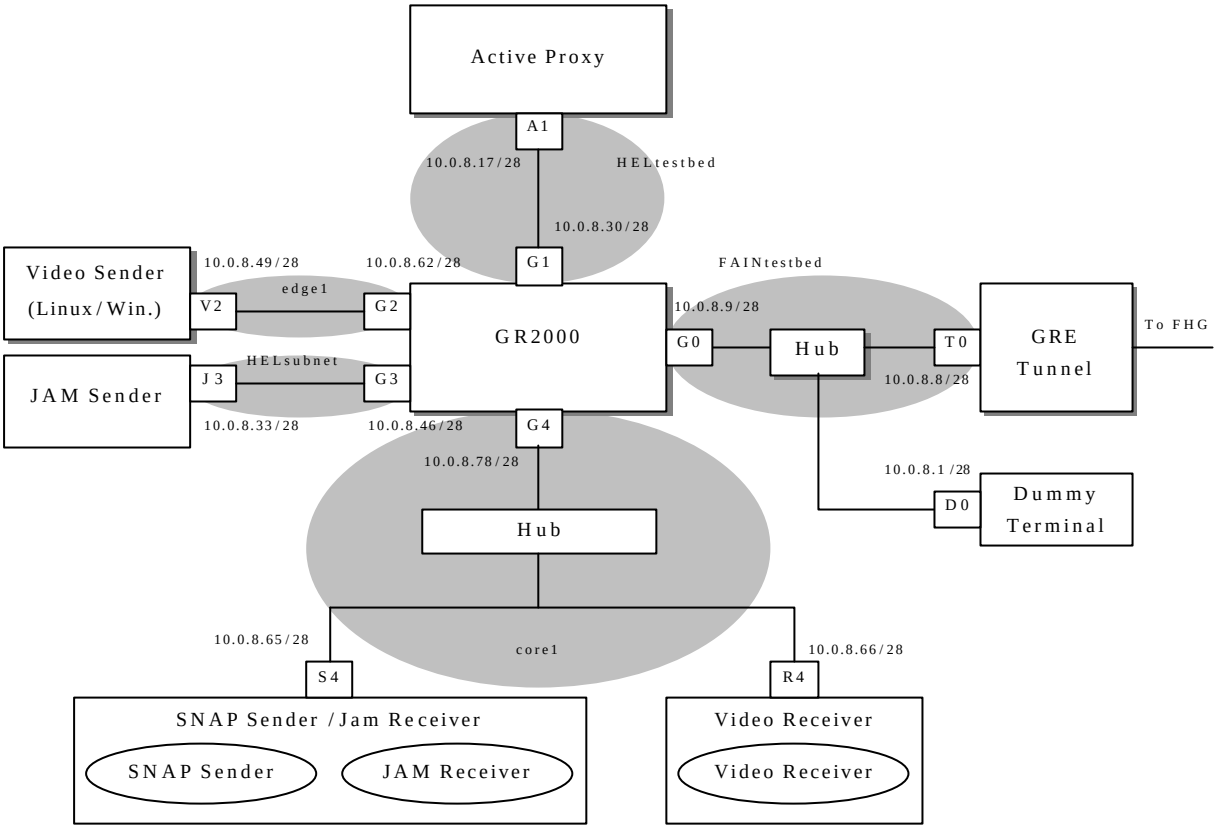


Figure 7-2: HEL Test-bed Configuration

7.1.2 FHG Test-bed Configuration

The Figure 7-3 shows a current network configuration of the FHG test-bed. In this configuration, we used 10Mbps bandwidth for the output of the TC-100, especially T2 port of the TC-100.

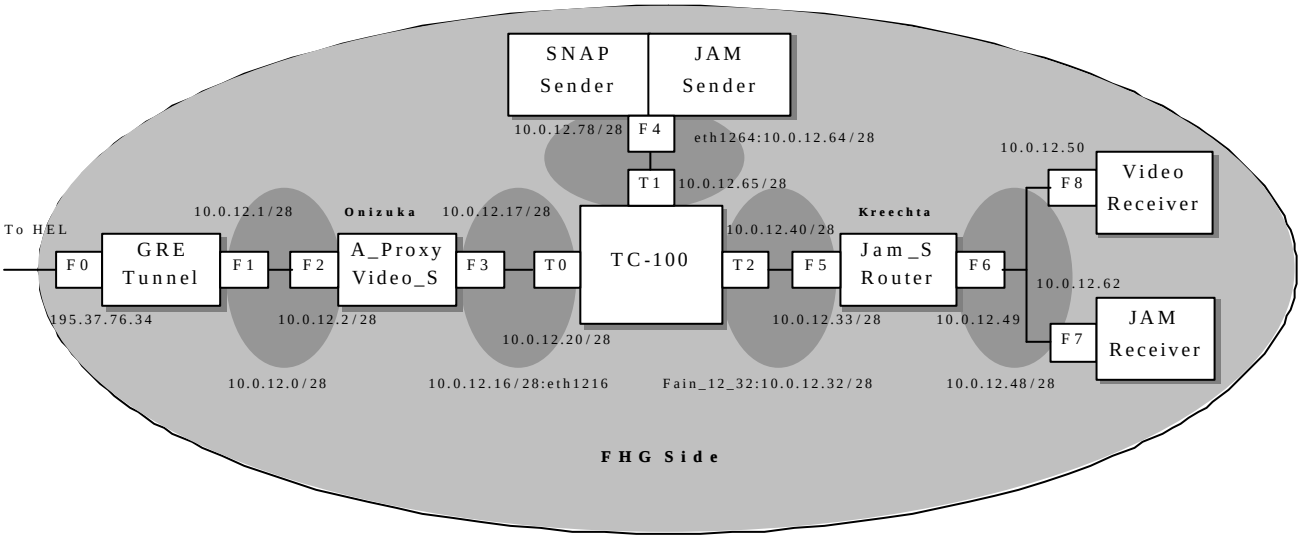


Figure 7-3: FHG Test-bed Configuration

7.2 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 7-2 and Figure 7-3.

7.2.1 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.

7.2.2 Setup and Demonstration

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.

7.3 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. 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.

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.

Network Setup

Figure 7-4 shows how the WebTV scenario is mapped to the FAIN testbed.

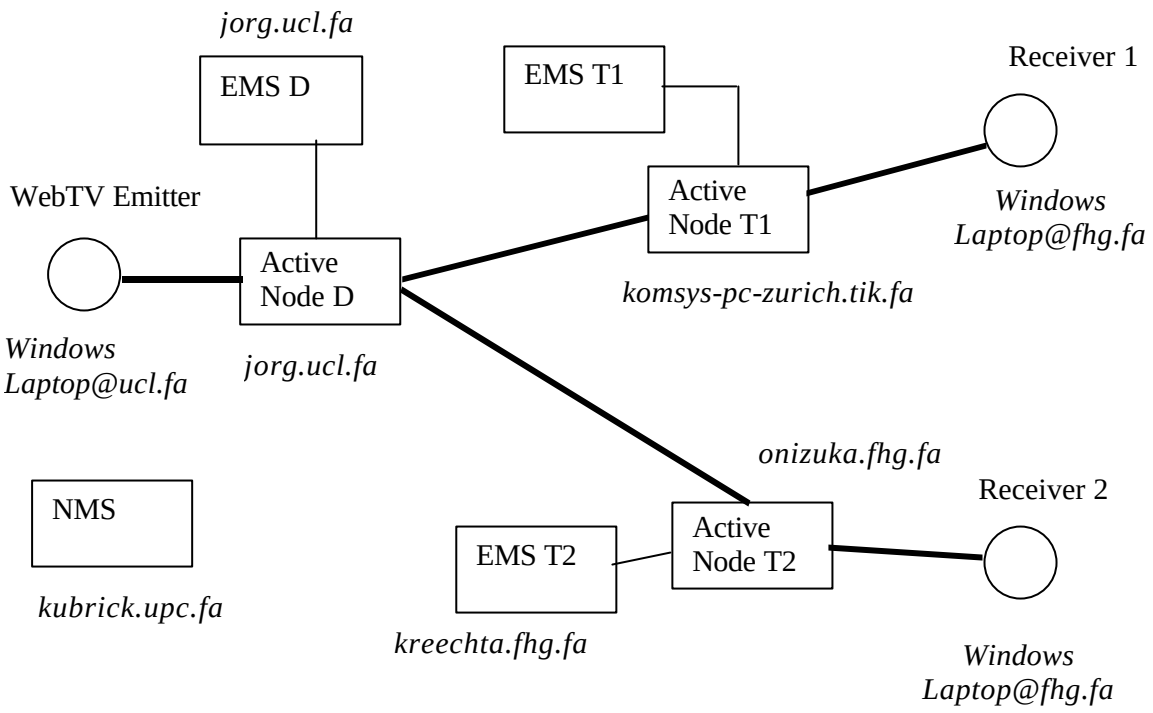


Figure 7-4: WEB TV mapped to Testbed

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)

7.4 Web Service Distribution Scenario

For the full and extensive documentation of the Web Service Distribution Scenario please refer to [9].

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.[12]

The overall scenario is depicted in Figure 5-5. Both redirection nodes and service nodes will be usually 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. The application scenario is subdivided in following 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..
- **Demo2:** 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.
- **Demo3:** This demo is an extension of Demo2 using several client-server pairs. Additionally a simple graphical configuration option is offered.
- **Demo4:** 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.
- **Demo5:** 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.
- **Demo6:** This demo adds support for sessions using the parser introduced in Demo5.

7.4.1 Network Setup

In this section, we describe the network configuration, both in terms of a general set-up which can be used by anyone who intends to run the demo in his own network, and in a mapping of the general set-up to the FAIN testbed, which is only relevant for those which want to run the demo on the FAIN testbed (and therefore have access to the testbed).

Set-up of a standalone demo

The generic network topology of our demo is shown in Figure 7-5.

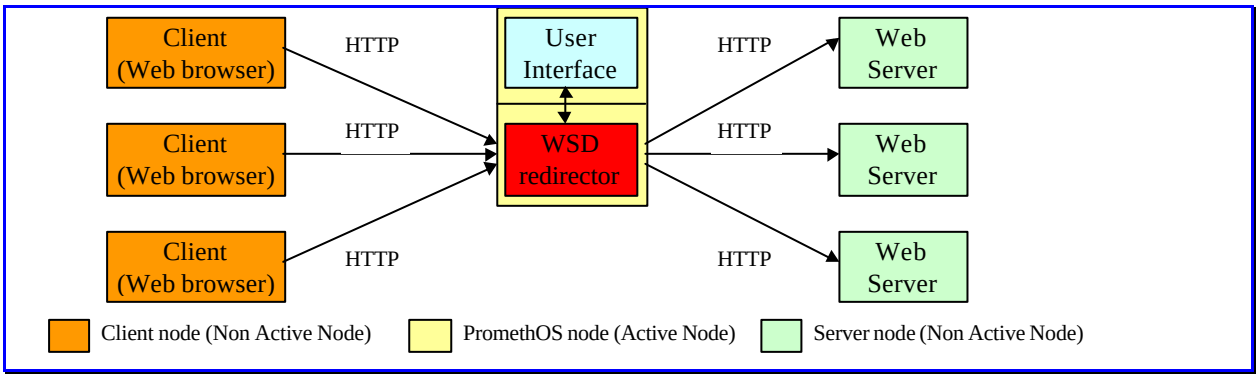


Figure 7-5: Network topology used by the demos

[12] 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 [10].

Active Node(s)

- PromethOS Nodes: For the demos there is only one node of this type. The node is responsible for the redirection of web traffic to different web servers. Additionally it runs the user interface for configuring and monitoring the PromethOS modules.

Each of the demos uses the following parts on an active node:

- 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.
- A PromethOS module: A PromethOS module is used for the actual manipulation of the network traffic.
- User interface: It is used to configure and monitor the PromethOS module.

This is shown in Figure 7-6.

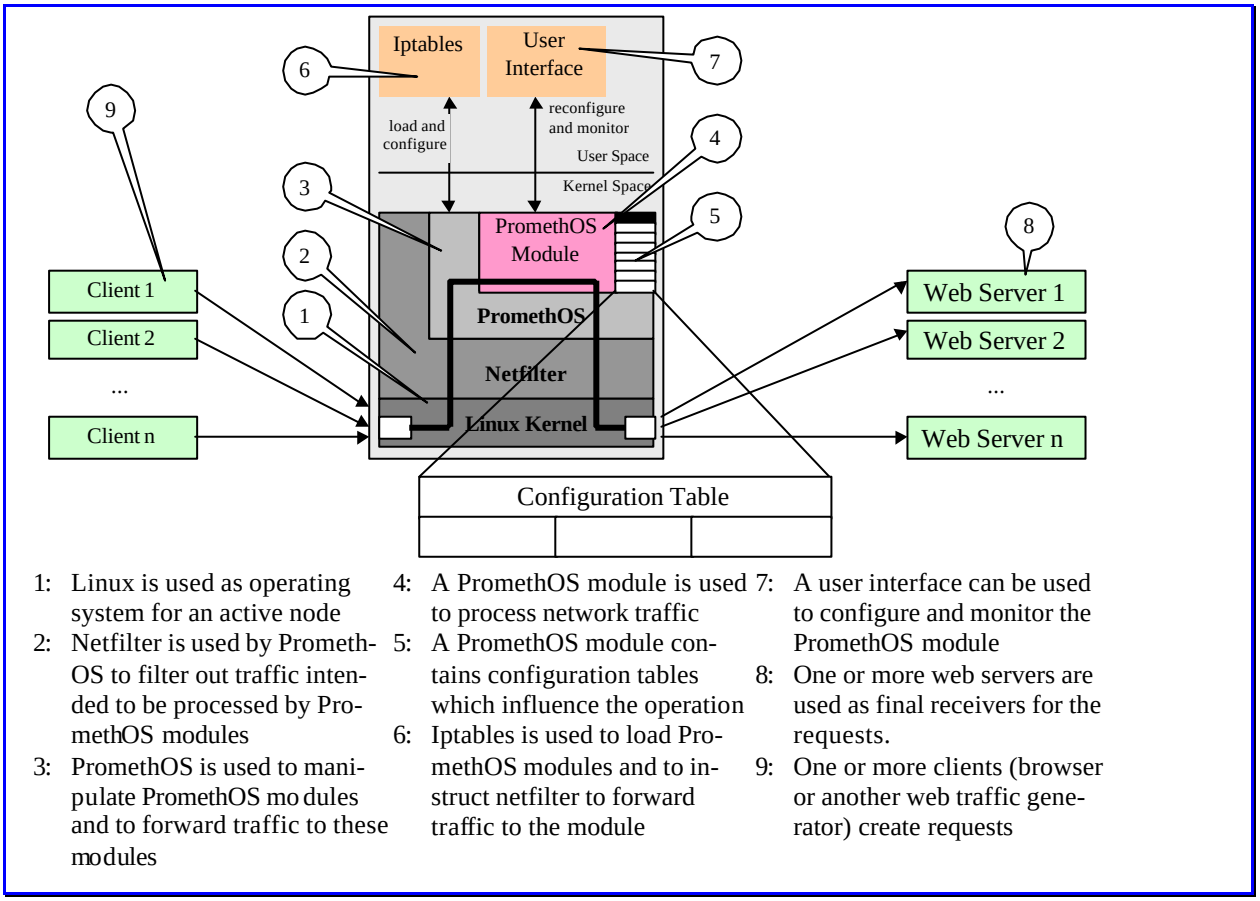


Figure 7-6: General Structure of a PromethOS Demo

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 7-5

Non-Active Nodes

- Client nodes: These nodes run a normal web browser or some other web traffic generator. There is no need that these nodes are PromethOS nodes.
- Server nodes: These nodes run a normal web server. There is no need that these nodes are PromethOS nodes.

Services

- None.

Set-up of a demo on the 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. Two possible mappings are shown in Figure 7-7 and Figure 7-8.

Possible set-up for demonstration taking place at FOKUS

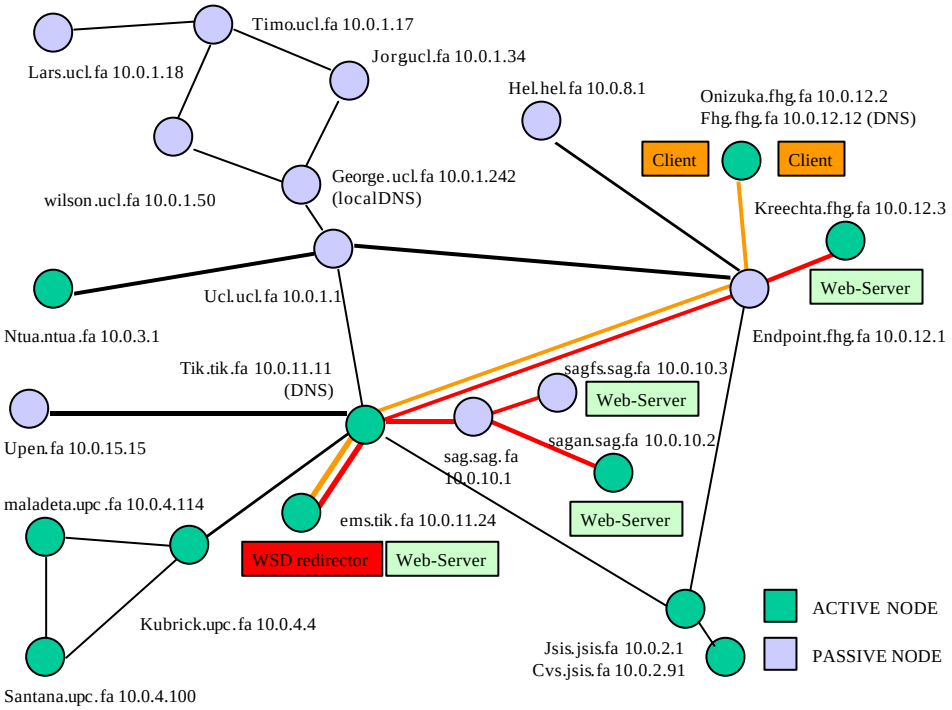


Figure 7-7: Network set-up for demonstration at FHG

Possible set-up for demonstration taking place at JSIS

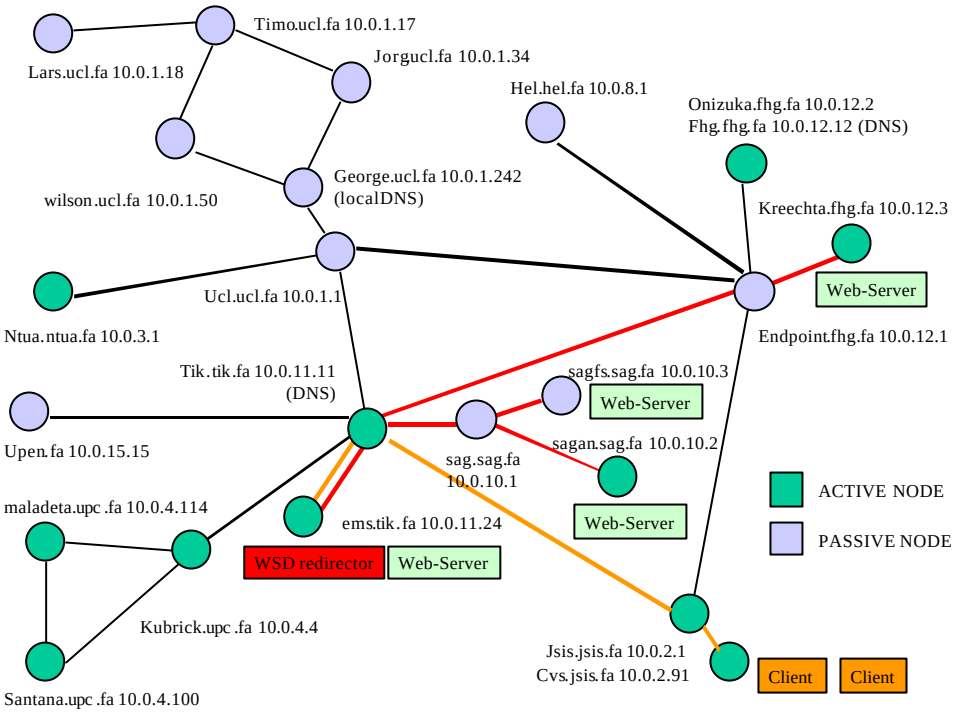


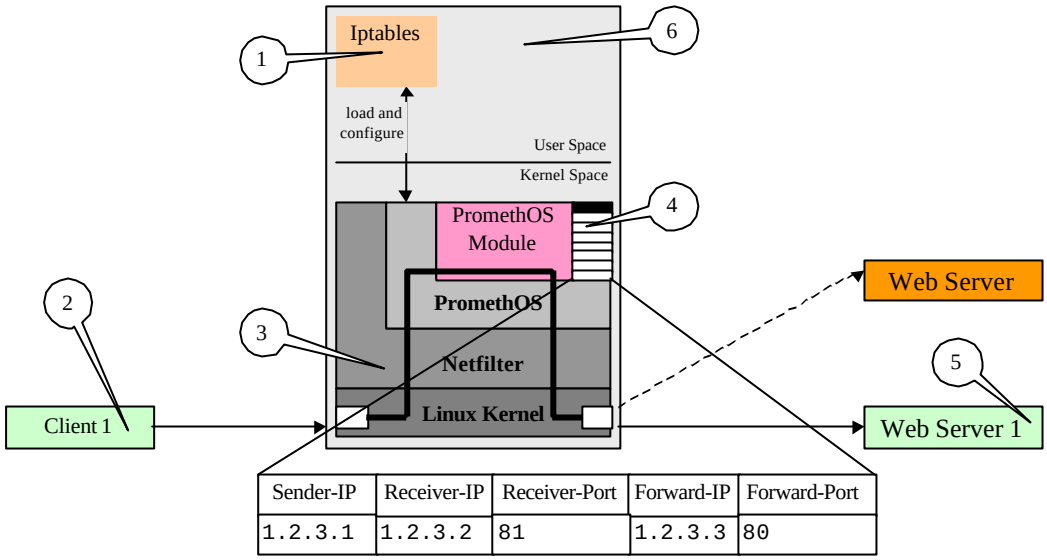
Figure 7-8: Network set-up for demonstration at JSIS

7.4.2 Description of Demos

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 simple redirector for TCP data. All packets, which are sent by a specified client and are 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.



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 Prometh-OS 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: This demo has no user interface

Figure 7-9: Structure of Demo1

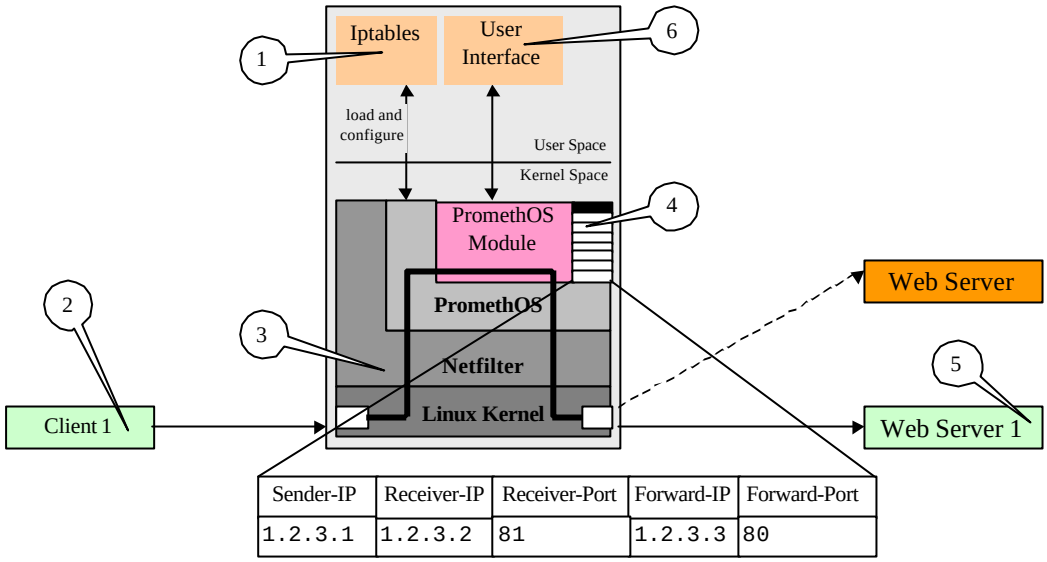
This demo allows to specify exactly one pair of client and server address which can be re-routed to another computer. Iptables is used to load the module and insert the necessary configuration data. The configured data can’t be changed at run-time. The demo uses only one client and one web server.

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 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 7-10.



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 7-10: Structure of Demo2

Demo 3

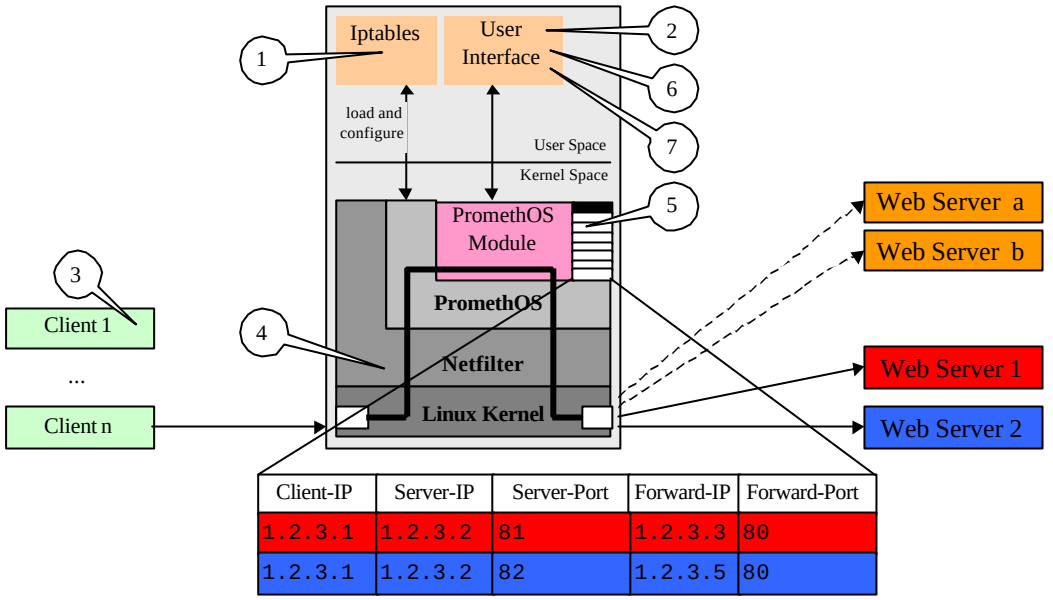
This demo is an extension of Demo2 using several client-server pairs. Additionally a simple graphical configuration option is offered. The user interface can be used to change the current configuration at runtime. This demo can use more than one client and more than one web server but there is no load 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 7-11 for an example).



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 7-11: Structure of Demo3

User Interface

The user interface contains a set of input fields which can be used to enter the necessary data. The following information is required:

- IP address of the client (the port of the client is not considered, because is changes for each connection).
- IP address of the original server
- destination port of the original server
- IP address of the final server
- destination port of the final server

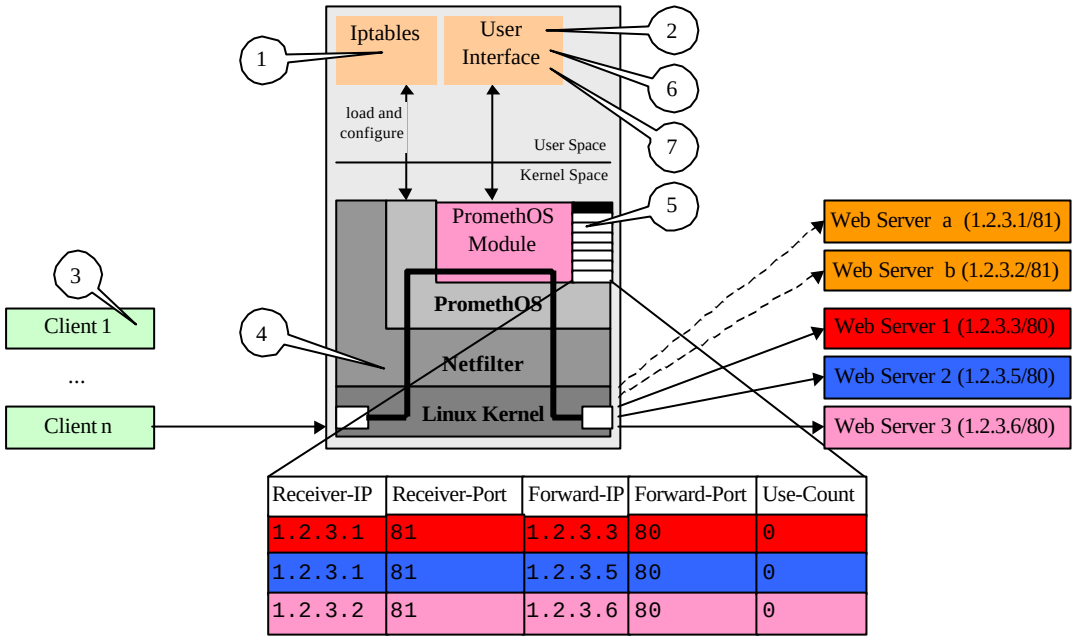
The button **Add** is used to transfer the data from the input fields into the internal tables of the kernel resident module. The button **Remove** is used to delete the entry from the internal tables, which matches the contents of the input fields. The button **Show** shows the current contents of the internal tables. **Clear Input** clears the input fields. **Clear output** clears the output field in the lower part of the user interface.

The menu **Table Entries** contains all table entries from the internal tables. If an entry from the menu is selected, the appropriate data are inserted into the input fields.

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 7-12 and Example 1 below).



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 7-12: Structure of Demo4

User Interface

As shown for Demo3 a simple graphical configuration option is offered. This can be used to change the current configuration at runtime. To do so the user interfaces contains a set of input fields which can be used to enter the necessary data? The following information is required:

- IP address of the client (not used in this demo).
- IP address of the original server
- destination port of the original server
- IP address of the final server
- destination port of the final server

The button **Add** is used to transfer the data from the input fields into the internal tables of the kernel resident module. The button **Remove** is used to delete the entry from the internal tables, which matches the contents of the input fields. The button **Show** shows the current contents of the internal tables. **Clear Input** clears the input fields. **Clear output** clears the output field in the lower part of the user interface.

The menu **Table Entries** contains all table entries from the internal tables. If an entry from the menu is selected, the appropriate data are inserted into the input fields.

Demo 5

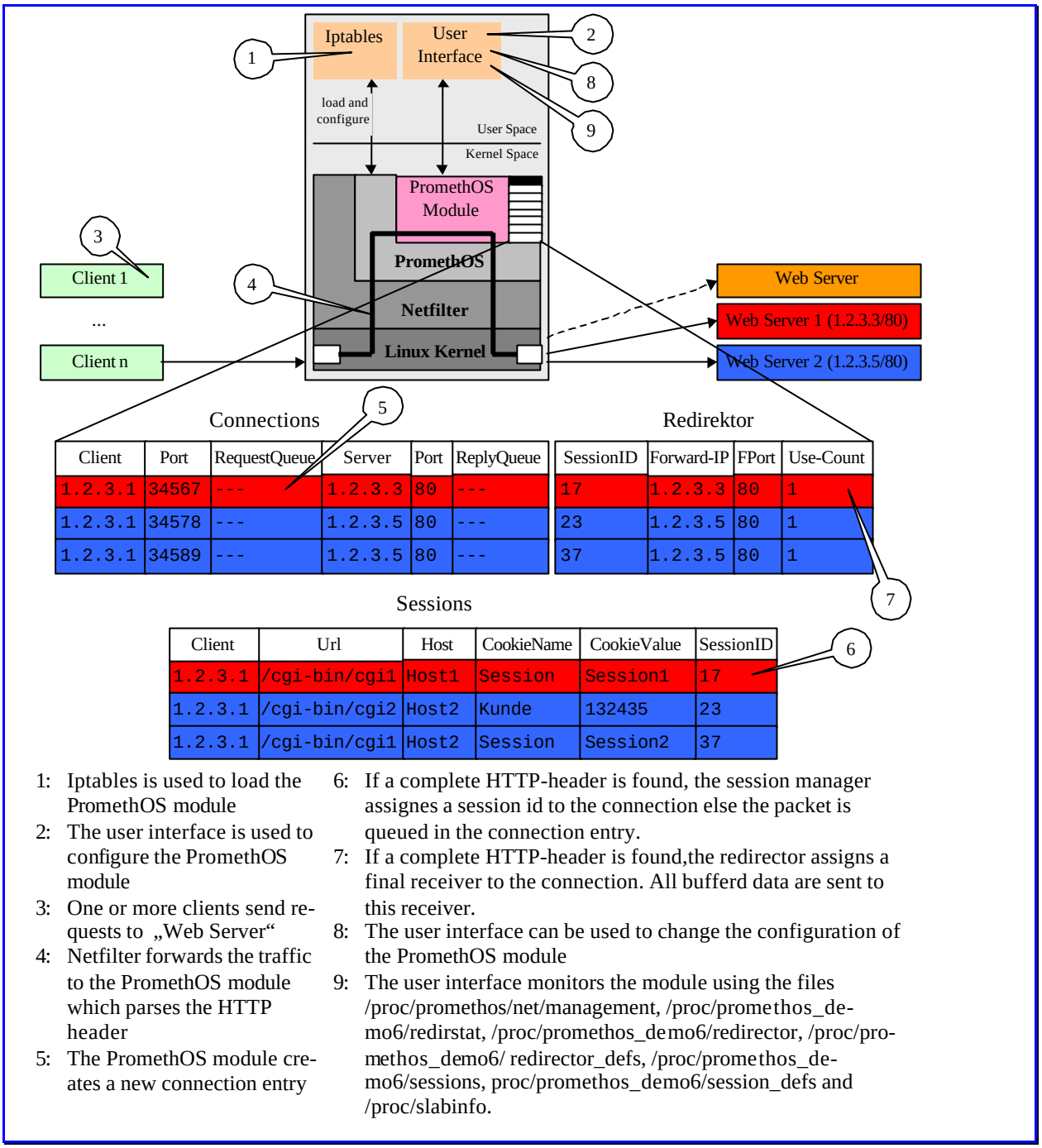
Demo5 is derived from Demo4. Demo5 adds a HTTP Parser to Demo4. The parser is used internally to examine all HTTP headers and to collect the data contained in the header. The collected information is not used in this demo however. A goal of this demo it to determine which load such a parser represents.

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 7-13.).



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	---

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

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

Figure 7-13: Structure of Demo6

User Interface

The user interface can be used to enter the necessary configuration data and to query the current state.

This demo contains 2 sets of input fields:

- the upper line is intended for the input of information for the session manager
- the lower line is meant for the input of information for the redirector.

The session manager needs the following data:

- prefix for the URLs which are to be combined into a session
- the name of a cookies, which is to be used as session cookie

- the address of the host the client (browser) sends the messages to
- the port the client (browser) sends the messages to

Note: The host must be indicated in the form, which is used by the browser to insert the host into the HTTP header. If e.g. both hostname and IP address are used, then both forms must be entered into the tables. If thus a computer with the name *testcomputer* has the IP address 1.2.3.4, then both forms must be inserted with URL and session cookie into the tables. The browser must use the same form for the host name for the entire period a session is valid. The conversion of a host name to an IP-address is not possible at present.

The button **Add** is used to insert the entered data into the internal tables. The button **Del** is used to remove the entry matching the data in the input fields from the tables. The button **Show** shows the current contents of the session table (the contents of this table are specified by the session manager and can not be changed). The button **ShowDef** shows the contents of the session definition table (those are the data entered by the user).

The redirector needs the following data:

- IP address of the original server
- destination port of the original server
- IP address of the final server
- destination port of the final server

The button **Add** is used to insert the entered data into the internal tables. The button **Del** is used to remove the entry matching the data in the input fields from the tables. The button **Show** shows the current contents of the redirector table (the contents of this table are specified by the redirector and can not be changed). The button **ShowDef** shows the contents of the redirector definition table (those are the data entered by the user).

Load Generators

For the demos described so far there are some load generators which create web traffic. The load generators are not demos by themselves but are used for the described demos.

All load generators have the following input fields:

Destination: This field is used to specify the destination of the requests

Port: This field is used to specify the destination port

Path: This field is used to specify the file/directory or web service to be used

Count: Number of requests to send

Sleep: Number of milliseconds to wait before the next request is sent

Web Services

The described demos use two simple web services to show redirection of web traffic in the case when sessions must be considered.

Counting Web Service

This web service maintains a simple counter for each session. The counter is incremented by 1 for each received request.

There are two instances of this web service:

- testcgi3.cgi
- testcgi4.cgi

For each request the following result is returned to the requestor:

```
----------------------------
SessionID = ssssss
----------------------------
Call Count = nnnn
```

`ssssss` represents the session ID which is generated by the web service for this session. `nnnn` is the number of requests received so far by the web service for this session.

Maze Generator

This web service generates different mazes for each session. For each request received from a client, part of the maze is returned. It is the responsibility of the client to display the received parts of the maze.

There are two instances of this web service:

- testcgi5.cgi
- testcgi6.cgi

The web server returns three different results:

- `INIT <colour> <sequence number> <width> <height> <num cols> <num rows> <cell width> <cell height>`

  This is the first command returned for a new maze.

  `<colour>` Background colour of the maze.

  `<sequence number>` sequence number, this number is incremented for each command returned by the web server

  `<width>` Width of maze in pixel

  `<height>` Height of maze in pixel

  `<num cols>` number of columns in maze

  `<num rows>` number of rows in maze

  `<cell width>` width of a maze cell

  `<cell height>` height of a maze cell

- `NEXT <colour> <sequence number> <cell xpos> <cell ypos> <dir> <op>`

  For each element of the maze one NEXT command is returned.

  `<colour>` Background colour of the maze. Must always have the same value as the colour returned with the INIT command.

  `<sequence number>` sequence number, must have a value one higher than the value contained in the previous command.

  `<cell xpos>` x-position of the cell the command operates on

  `<cell ypos>` y-position of the cell the command operates on

  `<dir>` type of the element inserted into the cell, different types of horizontal and vertical rectangles or maze walls.

  `<op>` operation, this can have the values

  0: draw a solid block, used to show a path through the maze

  1: draw a grey block, used to mark dead ends in the maze

2: draw the maze walls, <dir> contains the walls

- `EOF` This is the last command for each maze.

Requirements for a demo using the FAIN testbed

There is a preinstalled demo using the FAIN testbed. This demo uses the following testbed nodes:

- The active node is located at ems.tik.fa (10.0.11.24).
- The following nodes are configured as server nodes running apache:

  kreechta.fhg.fa (10.0.12.3)

  sagan.sag.fa (10.0.10.2)

  sagfs.sag.fa (10.0.10.3)

  ems.tik.fa (10.0.11.24)

  The required web services are already installed and can be used without further activity. The web server uses parts of the java documentation as static content.

- The client node(s) can be placed at any convenient node in the testbed. For the installation of the necessary load generators see chapter Installation and Configuration – Non-Active Node – Client Node.

To show a demo using the testbed the following step has to be carried out at least once:

- Install client nodes as described below in the chapter Installation and Configuration – Non-Active Node – Client Node. You need at least one client node. The client node should be a different computer than the computer running the active node (e.g. do not use ems.tik.fa as a client node).

Before you can show a demo, the following steps should be carried out:

- Login as root to the active node ems.tik.fa in a separate window and change to the directory /home/fainwsd/promethos/demo. This window is later used to start the demo on the active node.

  Note: To login to the active node you need the root password for ems.tik.fa.

- Login to the client nodes, change to the directory <promethosroot>/demo and run some load generators.

runload3: Run three load generators (PromethosDemoHttpLoad). The load generators do not use sessions. They can be used for the demos DEMO1 – DEMO6

runload3s: Run three load generators (PromethosDemoSessionLoad). The load generators use sessions. They can be used for DEMO6.

- runload3m: Run three load generators (PromethosDemoMazeLoad).

7.5 Video on Demand

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.

7.5.1 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.

7.5.2 Network Setup

As depicted in Figure 7-14 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.

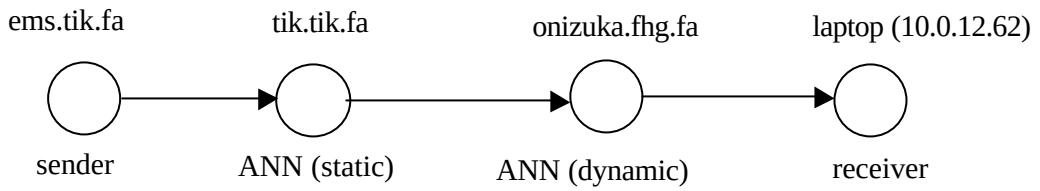


Figure 7-14: Network Topology for Video on Demand Scenario.

7.6 FAIN Mobility Demonstrator

The "FAIN Dino Park" (i.e. the 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.

7.6.1 Architecture/Setup

The generic network topology of our demo is shown in Figure 7-15. 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 sender nodes run common web browsers or some other web traffic generator. They are mobile clients like notebooks or PDAs. The receiver nodes run a normal web server. The scenario uses two nodes of this type. They play the role of content servers (e.g. static web pages or Web TV).

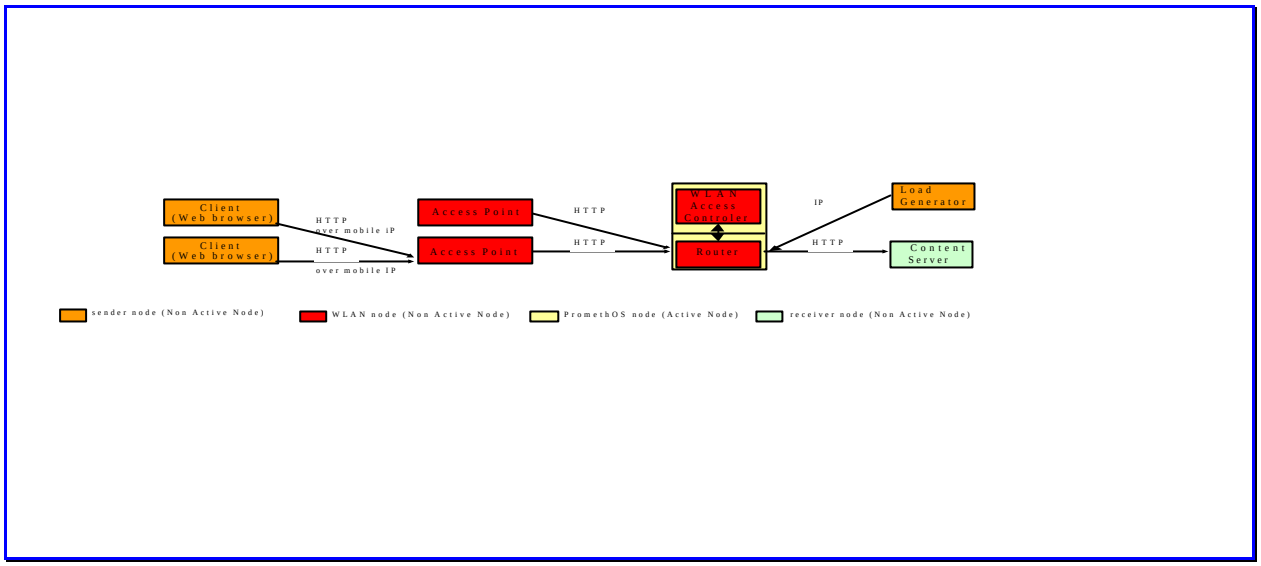


Figure 7-15: Network topology used by the demos

7.6.2 Network Setup

The Web servers used in the scenario are located at Berlin (kreechta.fhg.fa) and at the ETH (ems.tik.fa). In order use the preinstalled demo using the FAIN testbed, a FAIN Active Node must be installed at ems.tik.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 regulating 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 WebTV. Requesting WebTV generates the load at one access point (use of WebTV not yet decided on).

The structure of Demo2 is shown in Figure 7-16.

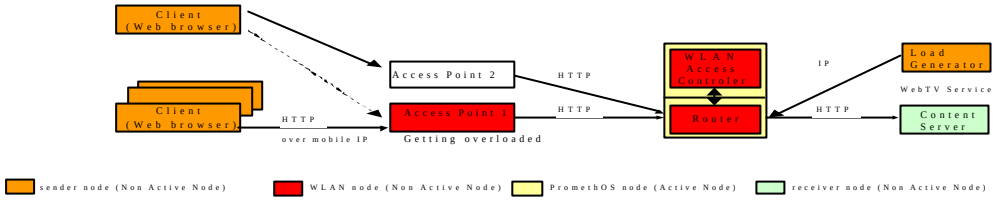


Figure 7-16: Structure of Demo2

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.

8 EVALUATION OF THE ARCHITECTURE AND IMPLEMENTATION

8.1 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 8-1 sketches the relation between ownership and cost of a property's feature, depending on the level/location and operational plane where it occurs.

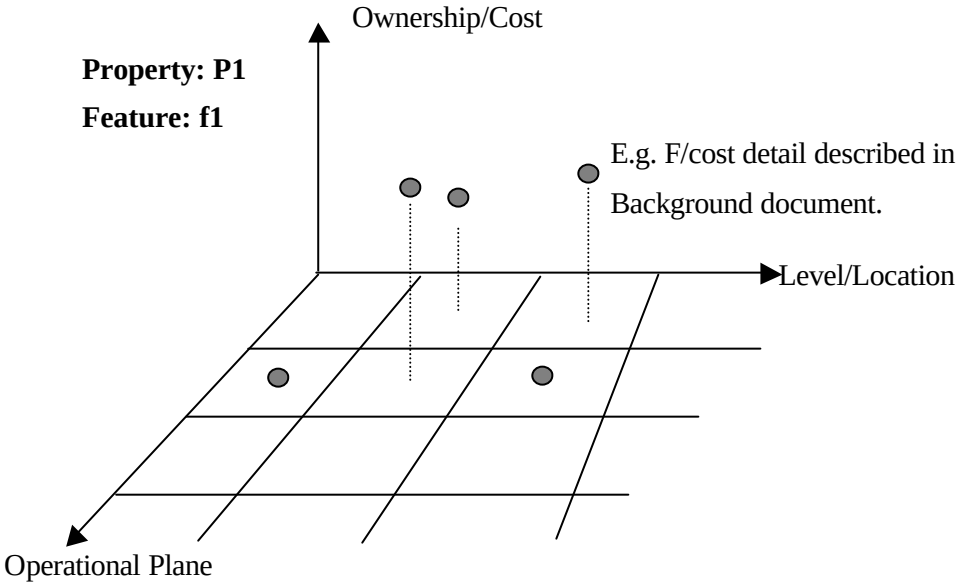


Figure 8-1: Evaluation Model for Features and Properties.

Figure 8-2 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.

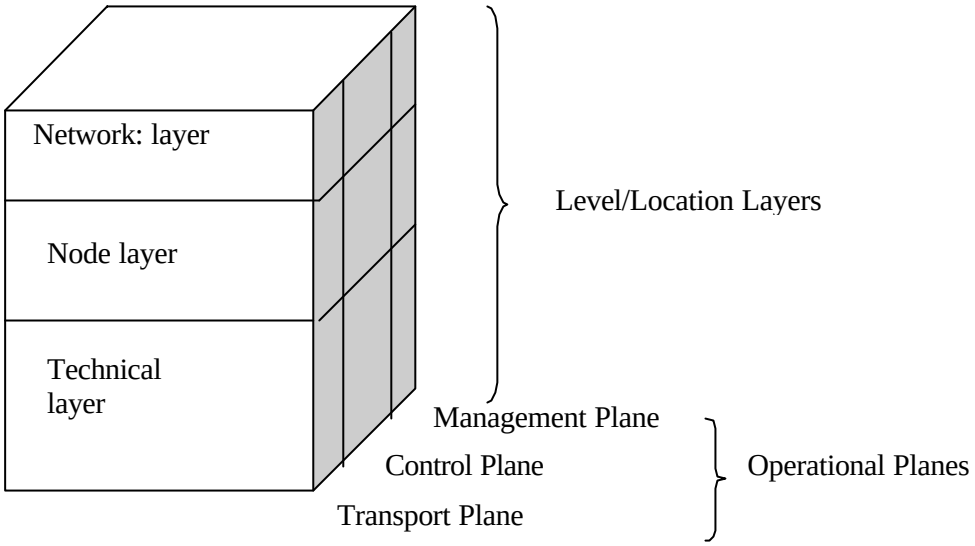


Figure 8-2: Reference Model for Operational Planes and Level/Location Layers.

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

8.1.1 Templates and Representation

For the representation of the evaluation (as suggested in Figure 8-2) 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 8-3 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.

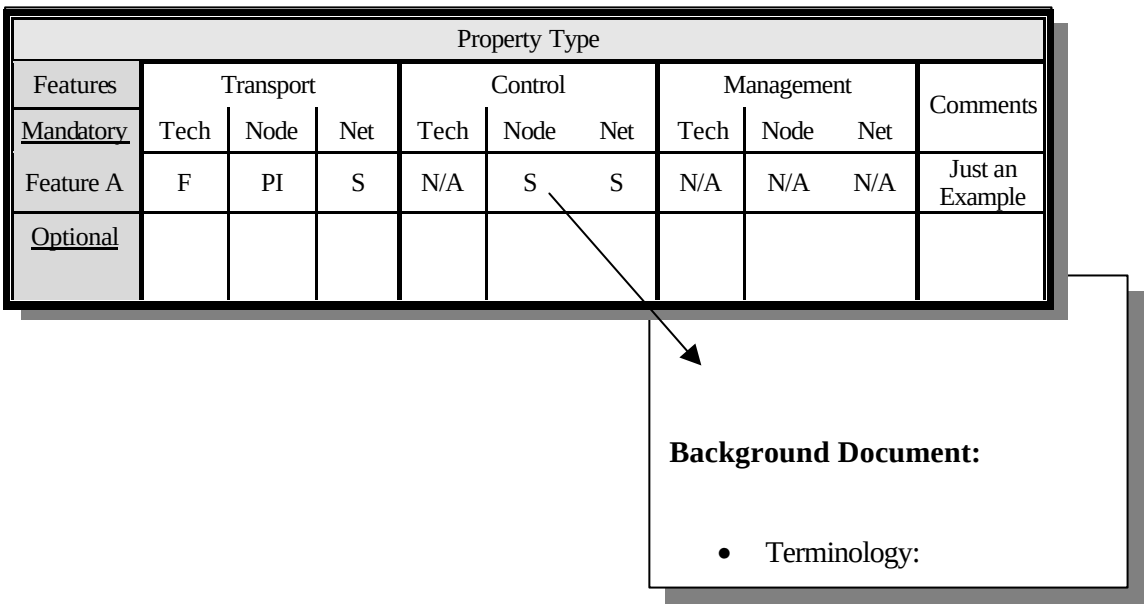


Figure 8-3: Two Level Evaluation Template for Property Types.

8.1.2 CLASSIFICATION OF THE FAIN COMPONENTS

In Table 8-1we propose a classification of the FAIN components according to the scheme proposed in Figure 8-2. 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 8-1:– Classification of the FAIN components

8.2 EVALUATION RESULTS

8.2.1 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 8-2:- 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 [33].
- **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. [35]
- **Network level:** at this level Composability is satisfied by the NMS [36]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 [33]
- **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 [37]). Regarding the Node ASP a deeper explanation of its extensibility is given in [38].
- **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 [36]. 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.

8.2.2 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, ([27] 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 8-3: - 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 158.

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 158.

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.

8.2.3 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 8-4:- - 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 [41]. 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 [41]. 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 [39].

8.2.4 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 8-5: - 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.

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 8-6: - 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.

8.2.5 Performance

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 8-4 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 8-4, 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.

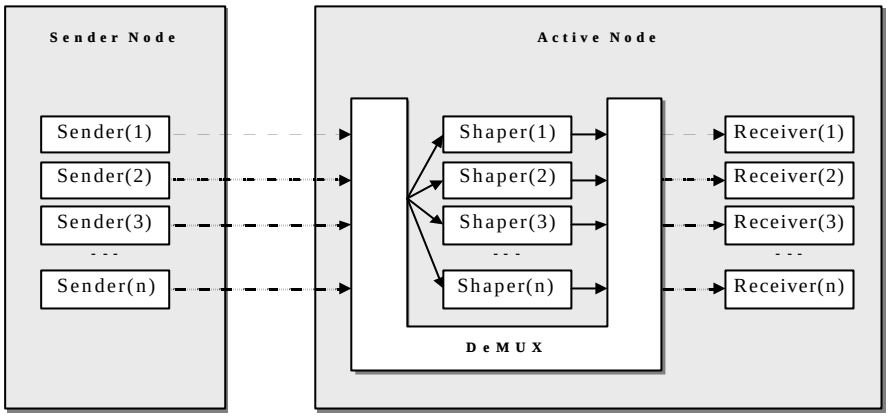


Figure 8-4: DeMUX Test System

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, 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 8-7: Specification of the Packet Sender

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

Table 8-8: 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 . 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.

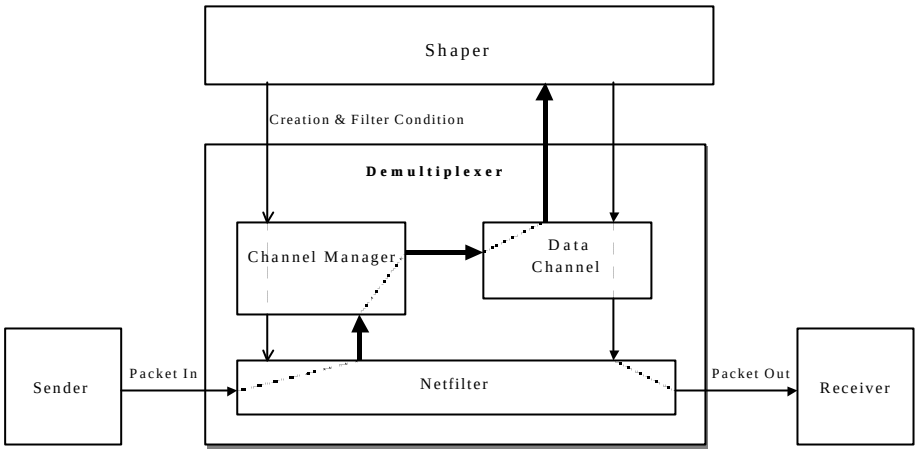


Figure 8-5: System Diagram for Demultiplexing Evaluation

Before starting with the evaluation of the DeMUX system the performance of the default system as depicted in has been evaluated. The specification of the active node is shown in the and the settings of the sender node are shown in the .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.

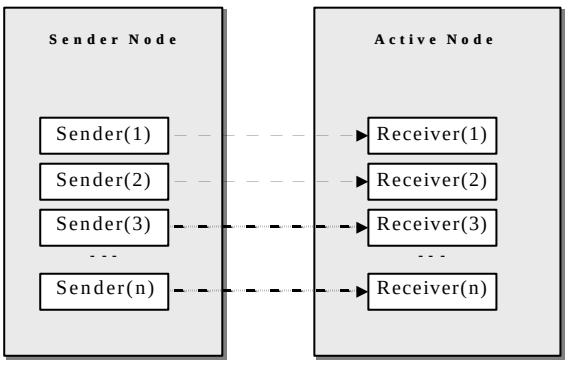


Figure 8-6: Block Diagram of the Default Data Transmission

Table 8-9: 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 . 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 8-10: 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 . 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 8-11: 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 . 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 8-12: 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.

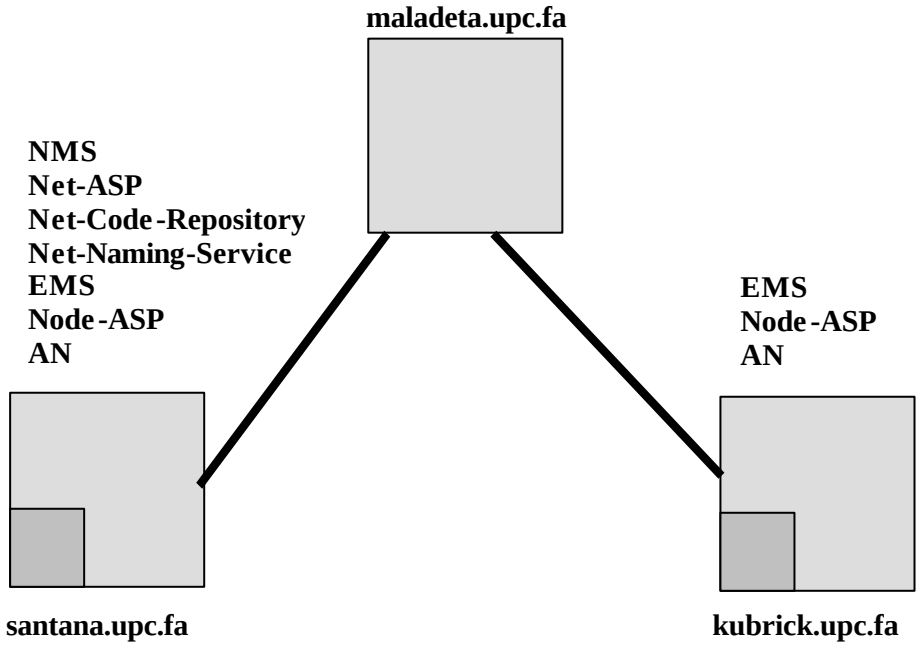


Figure 8-7: Topology for Evaluation Measurement

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)

Evaluation Results:

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

Table 8-13: Bootstrapping Measurements

Comments:

As can be seen in 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 8-14: 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[13]**	**Service PEP[14]**	**Dlg PDP**
19697	10776	6187	4594	167
18007	10375	5015	3567	149
1410	5097	0	0	0

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

[14] 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.

Table 8-15: EMS-santana Measurements

EMS – kubrick

Deploy Policy		**Deploy Functional Domain**		
QoS	**Dlg**	**QoS PDP**Error! Bookmark not defined.	**Service PEP**Error! Bookmark not defined.	**Dlg PDP**
10922	7734	6069	5293	200
9862	7906	5089	4338	185
1741	7326	0	0	0

Table 8-16: EMS-kubrick measurements

Comments:

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.

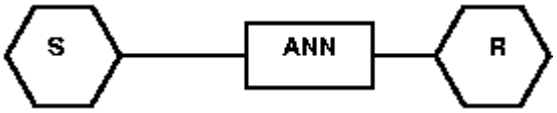


Figure 8-8: Network for performance measurements

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 (). 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 in Figure 8-9.

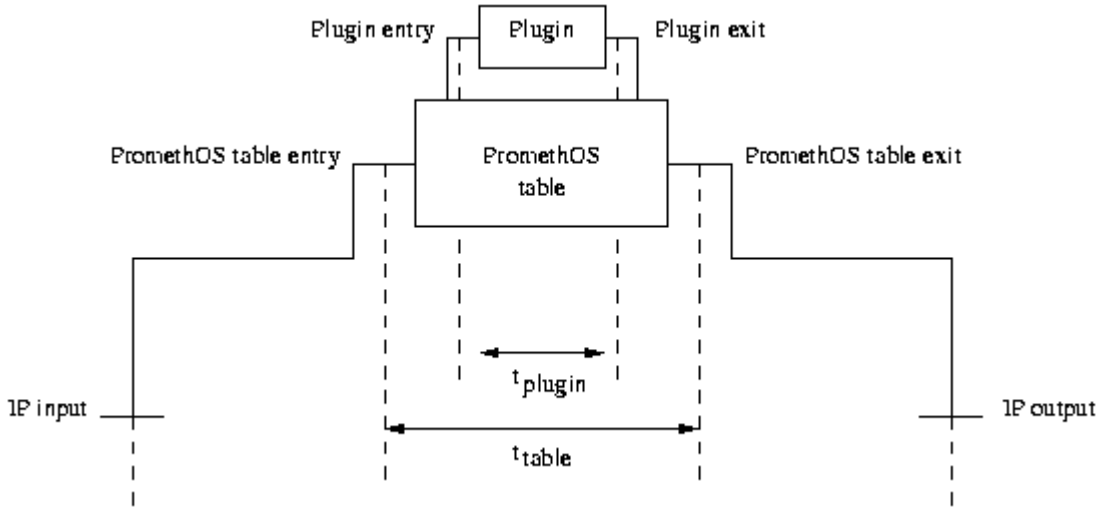


Figure 8-9: Evaluated Components

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 8-17: PromethOS Measurements

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 , 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.

9 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 8 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.

10 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
VAN	Virtual Active Network

VE	Virtual Environment
VEM	Virtual Environment Manager
VN	Virtual Network

11 REFERENCES

[1] S. Yoshizawa, A. Karlcut, “L-Interface for IP Router Flow & Output Queue Resource Abstraction and its application to Differentiated Services” IEEE P1520 IP Sub-working Group, July 1999.

[2] T. Suzuki, C. Kitahara, S. Denazis, “Data Path Creation Procedure”, WP5-HEL-049-DataPath-Intv0-2.doc

[3] B. Ghose, V. Jain, V. Gopal, “ Characterizing QoS Awareness in Multimedia Operating Systems”, 1999.

[4] M. Solarski, M. Bossardt, T. Becker, Component-based Deployment and Management of Services in Active Networks, In Proceedings of IWAN’02, December 2002.

[5] FAIN team. Active Node Architecture and Design. FAIN Public Deliverable D2, May 2001.

[6] FAIN team. Revised Active Node Architecture and Design. FAIN Public Deliverable D4, May 2002.

[7] H. Krawczyk, M.Bellare and R. Canetti. Hmac: Keyed-hashing for message authentication. RFC2014, Infromational Februrary 1997,

[8] A. Menezed, P. van Oorschot and S. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.

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

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

[11] IETF Web Replication and Caching (wrec) Working Group. http://www.ietf.org/html.charters/wrec-charter.html

[12] Building Bullet Proof Internet/intranet sites with IP Load Balancing - Scalability, Optimisation, Fault Tolerance and Availability with RADWARE. White Paper, Radware.com, http://www.radware.com/support/papers/bullet.pdf.

[13] cdcenter.com – content delivery and distribution resource centre. http://www.cddcenter.com/

[14] Digital Island – http://www.digisle.com/

[15] U. Legedza, D. Whetherall and John Guttag: Improving the Performance of Distributed Applications using Active Networks. IEEE Infocom, San Francisco. Proceedings, 1998

[16] Evaluation Framework, WP5-ETH-025028.doc76-D6-Int, FAIN project, Internal Report

[17] Evaluation Framework Questionnaire, WP5-ETH-025EvalQuest001.doc76-D6-Int, FAIN project, Internal Report

[18] C. Klein, L. Mandl: Web service Distribution - An application scenario of the FAIN architecture. FAIN project, Internal report. To be published.

[19] E. Pfeuffer, R. Schmid, C. Meyer, C. Niedermeier: FAIN Applications in Mobile/Wireless Networks. FAIN project, Internal Report. To be published.

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

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

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

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

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

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

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

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

[28] 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.

[29] 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.

[30] 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.

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

[32] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, "SIP: Session Initiation Protocol", RFC

[33] 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.

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

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

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

[37] 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

[38] 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.

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

[40] ASP, FAIN Deliverable n. 8

[41] Network ASP, FAIN Deliverable n. 8

12 APPENDIX – MOBILITY SCENARIO EVALUATION

12.1 Introduction

The main focus of the FAIN project is on traditional IP based data networks. Restricting oneself to wired networks does, however, exclude a technologically as well as commercially interesting type of networks: wireless networks.
As the concept of active networks shows improvements for fixed networks [see results of the Fain Demonstrator], an evaluation of the benefits of these concepts is certainly worthwhile taking into account the economic importance of these networks as well as the technological challenges. Therefore, it is in order to evaluate if and how active networking in general and FAIN concepts in particular may be useful if applied in the mobile domain. In this work package, supplementary to the main line of investigation in FAIN, we evaluate the extension of FAIN concepts to mobile networks taking WLAN as the prime example.

The setup and the basic mechanisms of a wireless extension of the FAIN network are sketched in sections 2 and 3. Chapter 4 addresses the scenarios of the mobile demonstrator in more detail. This is the basis for the evaluation of mobility concepts in FAIN.

The evaluation of the FAIN Applications in Mobile Wireless Networks differs from the FAIN evolution. The FAIN evaluation is focussed on properties of the core of the FAIN demonstrator, the execution environment, whereas the evaluation of the FAIN Applications in Mobile Wireless Networks aims at the benefits of active networking within a specific application domain: mobile applications.

Beyond the scope of the practical experience gained by the FAIN Mobility Demonstrator, the mobility issue deserves a broader scope to assess the benefits of active networking concept. Work in this direction is still in its infancy, so we restrict to a short overview detailing the current status of research and indication some main issues.

Only a few scientific papers discuss the application of active networking technology to future mobile networks. An overview is given in section 12.6.1. They argue that programmability is a central concept in future mobile networks, because flexibility is a key requirement for future mobile services for the following reasons:

- Data connections in Mobile networks are considerably more fragile than in wired networks, due to the wireless links and mobility. Hence they require sophisticated mechanisms on the network layer.

- Mobile networks are more expensive than wired networks; hence optimizations pay off more easily

- Multi-layer aspects are of growing importance in mobile networks (e.g. for Quality of Service, Service Adaptation, Reconfigurable Radios)

- Considerable innovation in air interfaces will require adaptation to new technologies

Finding a way to deal efficiently with the flexibility of networks and protocols required by these demands is therefore an important goal.

Flexible Quality of Service and adaptation of services is another important example of active networking already in wired networks (see for instance the FAIN show cases). Being even more relevant for wireless networks, one can expect a wide range of applications.

Especially because mobile networks are becoming more and more important, there is an increasing need for a novel network infrastructure, which enables the fast deployment of the new services. Typically, it is assumed that more flexibility is needed on the application and middleware layers than on the networking layer.

According to 0 for the mobile domain, some Programmable Network Applications are:

- Generic mobility support (e.g. MAO)
- Multicast, including multicast to mobile users
- Mobile Proxies (http, ftp, etc.)
- Mobile Firewalls
- 3G type "signalling" for user control/metering, but with "basic" SIP alone
- Management and enforcement of (mobile) user policies/profiles
- Mobile user metering& accounting & charging & (hot) billing & fraud control
- Management and enforcement of inter-operator policies (e.g. diffserv practices at network borders)
- Mobile Personal Network Services, context sensitive or not
- Mobile E-mail boxes
- QoS management for mobile users
- Mobile VPN, with QoS
- Detection of traffic anomalies and taking corrective actions (malicious user traffic, machines going berserk)
- Mobile app. converters (e.g. doc to pdf, test implementation described in 0)
- Mobile transport protocol converters
- Updating of router configurations & protocols without service interruption
- Flexible load balancing & diagnostic & fault recovery

Based on the principle mechanism implemented by the FAIN Demonstrator and taking into account the limited budget for the mobile scenarios, we focussed on dedicated scenarios (see 0 and 0) and the underlying concept for seamless handover. Experiences gained during the implementation and by evaluating the mobile demonstrator are documented in section 12.5. First, in section 12.5.1 the main criteria are mentioned and second, in section 12.5.2 our evaluation results are documented. In conclusion, the section 12.6 contains a short summary of the state of the art in active mobile networks research and development and highlights future directions in active mobile networks research and development.

12.2 FAIN mobile Network DEMONSTRATOR: principle Mechanisms

The FAIN mobile network is a wireless LAN, which is connected to the existing wired FAIN LAN as an extension. So the FAIN network is extended from wired network to WLAN. Additionally, active networking concepts are applied to mobile scenarios and their mobile applications. Most of the principle mechanisms (implemented in the Fain mobile network) are the same principles like in the fixed wired network of FAIN. So, a FAIN active mobile node is a special FAIN active node: a FAIN active node is configured by a WLAN access controller (see **Figure 10**).

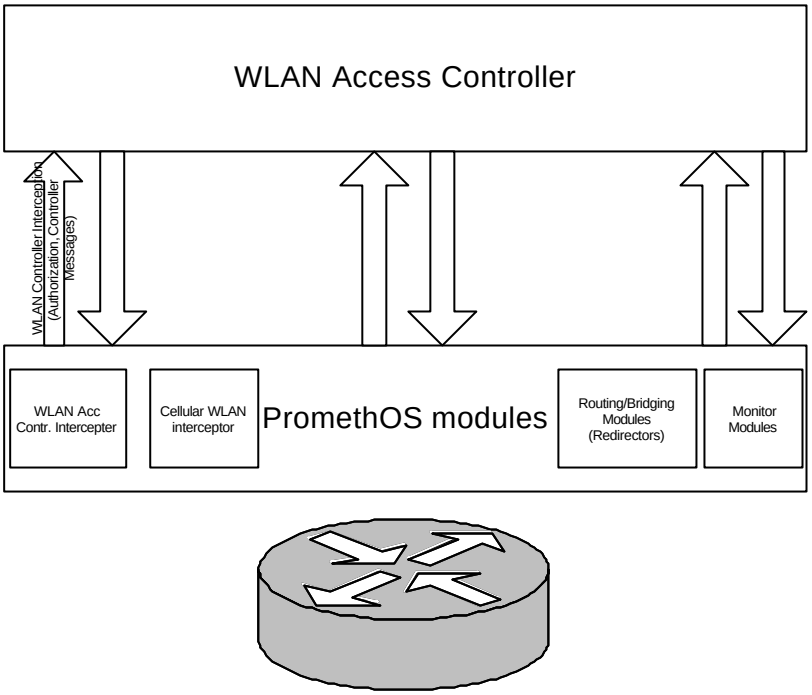


Figure 10: FAIN Active Node for Mobile Applications

The WLAN access controller keeps a log on usage and users of the WLAN network. Based on this information load distribution in the WLAN network is enforced. It directly interacts with the active node by installing and configuring modules on the active node.

The FAIN active node, which is configured by the WLAN controller, is responsible for IP layer decisions concerning the traffic in the WLAN network. It implements routing/bridging, as well as monitoring and interception functionality required for the WLAN controller. The functionality is realised by modules deployed at the active node.
Tasks and responsibilities of the active node are:

- Bridging between WLAN AP subnets and other networks
- Count data flow from/to AP per user
- Routing to simulate bridging functionality of active node
- Gather messages to WLAN access controller
- Message Flows

The message flows between the different entities are elaborated in the detailed design section.

Such a mobile Fain active node builds the transit from the fixed FAIN active network to the mobile Fain active network. So the general infrastructure of the mobile scenarios results in Figure 11, see section 12.3.

Comprising the following FAIN concepts are used for the mobile scenarios:

- Creating Virtual Environments as Part of the Virtual Networks Creation
    - This demo does not use Virtual Environments. Creating Virtual Environments is part of another demo described in 0.
- Resource Control for hard Resource Partitioning
    - Resource Control is not part of this demo.
- Deployment of different Types and Instances of EEs

Manipulation of PromethOS modules using an extended version of iptables provided by PromethOS. This provides an intermediate step towards a full integration of PromethOS plugins into FAIN execution environments. A next step will be described in 0.

- Creating and Operating Component-based EEs
  - This demo will use only one EE on one node.
- Tuning the Active Network for maximising Performance

  Using PromethOS kernel modules for processing HTTP traffic will be a step towards maximising the performance for processing web traffic.
- Simple fault management functionality

  There is an automatic reconfiguration capability, which will remove shut down web servers automatically from the list of allowed targets. When the server becomes available again, it will be reactivated automatically after some time. As long a there is at least one working web server, the end user will see a usable network (with maybe degraded performance of course).

  The interactive configurability of PromethOS plugins enables a service provider to react promptly to changing requirements (e. g., web servers which must be shut down, increasing load, etc.).
- Active Network Upgrades
  - PromethOS allows the dynamic loading and unloading of plugins. This is a first step towards a dynamic upgrade of an active node.
  - There are currently no provisions for a seamless upgrade without interrupting a running service. There will be at least a short break when the old plugin is unloaded until the new plugin is loaded and configured.

Additionally, there is one typical mobile active networking concept, which are used for the mobile scenarios:

- Resource Control for Access Points, especially for WLAN
  - Monitoring of Load on Access Points
  - Management of Active handover
  - Controlling of Active handover

12.3 The Fain mobile testbed

The FAIN mobile network is a wireless LAN, a WLAN, which is connected to the existing wired FAIN LAN as an extension. Therefore, the evaluation is restricted to WLAN, too.

The FAIN mobile test bed is shown in Figure 11. The WLAN Access controller builds the bridge to the wired FAIN network.

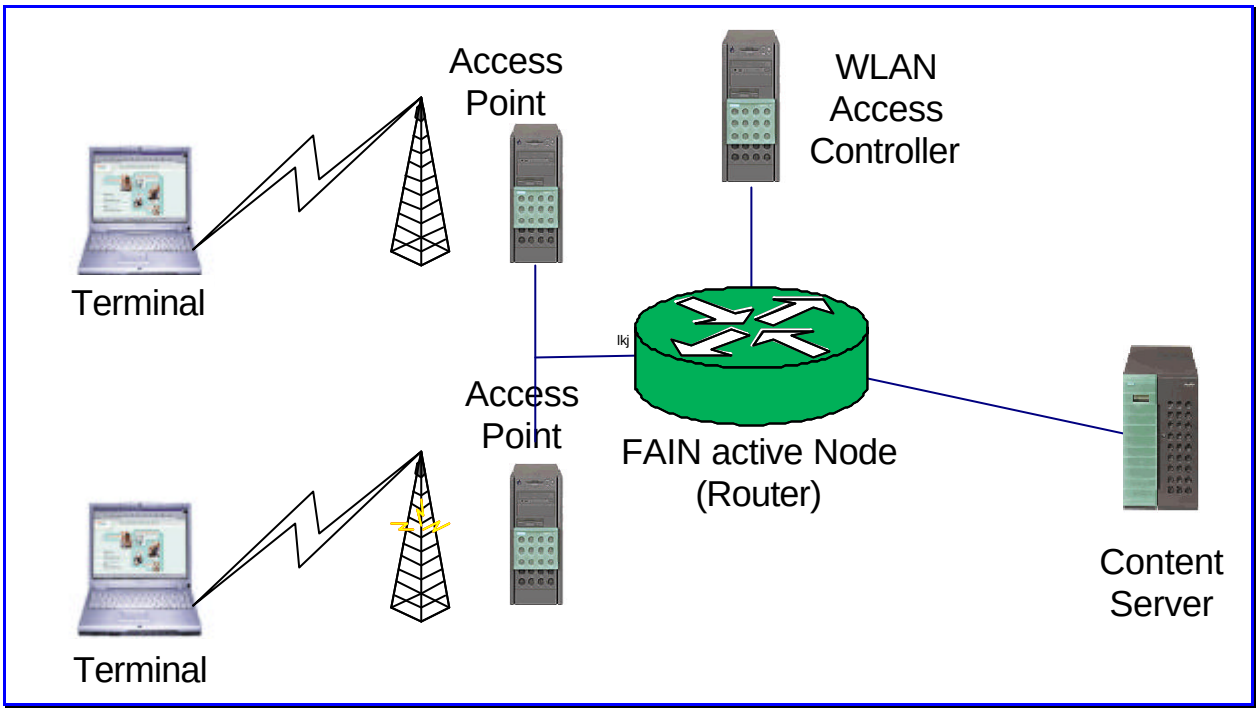


Figure 11: Generic Infrastructure of the Demos

The basic building block of the wireless LAN is the Cell. This is the area in which the wireless communication takes place. All radio communication in the cell is coordinated by a traffic management function. A unit called Access Point performs it. The access points connect wireless LAN cells to a wired Ethernet LAN via a simple cable. So the access point functions as a bridge between the cell and the wired LAN. Once connected to a wired LAN, the network management functions of the wired and the wireless LANs can also be integrated.

Several access points can be positioned in such a way that their coverage areas converge, thus creating a multi-cell. Stations inside the multi-cell area automatically "choose" the best Access Point to communicate with via evaluating signal strength. Overlapping coverage area is an important attribute of the wireless LAN setup, because it enables seamless roaming between the overlapping cells.

Users with portable stations can move freely between overlapping cells, continuously maintaining their network connection. This ability to move around the wireless campus is called "Roaming". Roaming is seamless, that is, a work session can be maintained when moving from cell to cell while the user experiences, depending on the traffic, only a momentary break in the data flow. A station implements its roaming capabilities by "choosing" the access point in its area that provides the clearest signal.

The FAIN mobile test bed is the typical infrastructure for the generic mobile demonstration scenario, the FAIN mobile scenarios, which are described in 0.

12.4 Fain mobile scenarios

The FAIN mobile scenarios are initiated by the 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 show their advantages. The Fain mobile scenarios are settled in the edutainment domain. They are part of the showcase "FAIN Dino Park" 0. They demonstrate how in a challenging wireless environment load-balancing and load reduction approaches succeed in avoiding bottlenecks and could improve edutainment concepts.

The focus of the demonstration is on load-balancing and load distribution in mobile networks. The implementation supports the two following demo cases for load balancing in WLAN 0:

a) on camp-on (when connecting to the system, a client is rejected and redirected to another AP if the tried one is overloaded.)

b) 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.)

Load balancing or distribution can be used to achieve non-functional requirements such as

- Reliability: deals with faults and shutdowns of particular servers by automatically redirecting traffic to servers, which are available.
- Performance: reduced response times even if several simultaneous users use a particular functionality.
- Scalability: by simply adding additional servers, the system can be configured to work with higher traffic loads.

12.5 Evaluation of the FAIN Mobile Demonstrator

The mobile FAIN Demonstrator contributes to the FAIN evaluation regarding the following points:

- Flexibility Property
  - Dynamic loading and unloading of PromethOS plugins
  - Interactive configurability of PromethOS plugins
- Security Property
  - Currently relies on the fact, that PromethOS plugins can only be loaded by root. For a better integration into the FAIN security model see [2].
- Portability Property
  - PromethOS plugins can be deployed only on active nodes running PromethOS
  - PromethOS itself is based on Linux
- Reliability Property
  - AN Concepts are used in this scenario to implement reliable mobile applications
- Performance
  - Throughput of web traffic to be measured
- Interoperability Property
  - For further study.
- Timeliness Property
  - End-to-end delay: no hard requirements, should be sufficient for interactive use.
  - Hard to measure anyways, as it depends on many factors (delay imposed by the web servers, delay on active nodes, delay on non active nodes passed by packets, etc.).
- Openness Property
  - Standard web protocols are used and therefore the approach is “open” to any web user or web service provider

These points essentially reflect the result of the PromethOS evaluation as the present demonstrator is based on it. However, these points are not in the focus of the FAIN mobile demonstrator evaluation.

To meet future requirements for ubiquitous communication, future wireless mobile systems require both high bandwidth wireless communication links and a very efficient and widely adaptable mobility control and management architecture. From these points of view, the a FAIN based mobile infrastructure is to be evaluated.

12.5.1 Evaluation Methodology

For the evaluation results the following criteria are interesting:

- Scalability: Nodes, Access Points, Clients
- Performance
- Architectural Concept
- Interoperability with other concepts (good, integration with fixed networks, supports heterogeneous networks)
- Future long-term suitability
- Mobility support
    - QoS management for mobile users
    - Updating of router configurations & protocols without service interruption
    - Flexible loadbalancing & diagnostic & fault recovery
- Adoption to other wireless networks

12.5.2 Evaluation Results

The experience with the design and development of the mobile demonstrator showed, that using FAIN architecture and PromethOS in particular the following goals could be achieved:

- rapid prototype development
- separation of hardware and software layer in routers
- high flexibility due to programmable cross layer interfaces
- peacemeal evolution
- run-time extensibility
- implementation based on standard Linux software

As regards the issues mentioned in the evaluation methodology, the experiences may be summarized as follows:

- Scalability: the use of active networking does not impose any additional scalability restrictions, which do not already occur without the use of PromethOS.
- Performance: not evaluated, as the implementation is restricted to a demonstrator
- Architectural Concept: the separation of kernel und user space has been used in a beneficial way to implement the demonstrator without having to do major kernel-programming work.
- Interoperability: is to established easily as it only requires the use of new PromethOS modules
- Future long-term suitability: by changing the PromethOS modules the system is easily adapted to evolving technologies and standards
- mobility support: see discussion of handover at the end of this sections.

- adoption to other wireless technologies: not evaluated, as this requires considerable extensions to other wireless technologies.

Handover is a potentially frequent event in a next generation picocellular environment. The resulting mobility management is rather difficult. To ensure that the mobile terminal performs a handover in an efficient and reliable way is a challenge for such mobile networks. The mobile FAIN demonstrator presents a seamless way to perform handover. Additionally, the implemented active handover provides an improved QoS management for mobile users and a better resource utilization concerning access points.

12.6 Conclusions and recommendations

12.6.1 Summary of the state of the art in active mobile networks research and development

Many scientific papers discuss the application of active networking technology to future mobile networks. They argue that programmability is a central concept in future mobile networks, because flexibility is a key requirement for future mobile services for the following reasons:

- Mobile networks are very fragile, due to wireless links and mobility

- Mobile networks are more expensive than wired networks; hence optimisations pay off more easily

- Considerable innovation in air interfaces will require adaptation for new technologies

So they deal with flexible networking protocols which will be needed in future mobile networks, and mobility management and hand-over optimization as a central service of mobile networks, which can be optimized in many ways.

Flexible Quality of Service and adaptation of services is a common example of active networking. For instance, in 0, filters are uploaded dynamically to adapt the capabilities of the networking environment.

At Siemens (CT SE 2) within the project MASA I 0 mobility management and seamless handover for an enhanced support of user-defined QoS requirements was investigated. The supported networks were UMTS FDD, WaveLAN and LAN. The focus of the project was on:

–Mapping of Quality of Service from user level (best, good, medium,..., best effort) to media stream, system resources, and network level

and

–Dynamic adaptation of media streams depending on network (bandwidth, quality), terminal and global policy.

The objectives of the MASA I project were the definition and implementation of a comprehensive QoS framework for 'Mobility and Service Adaptation in Heterogeneous Mobile Communication Networks' (MASA), in particular a Quality of Service (QoS) Framework for Audio and Video over IP services was implemented. Siemens's thesis is that, in order to provide high quality communication for mobile users, media processing facilities as well as mobility handling and handoff decision mechanisms should be closely integrated into a QoS framework. This allows, e.g., to base handoff decisions on all available QoS elements such as availability of transcoding units or local resource management.

The MASA framework is able to release applications of QoS-related work as much as possible and, in addition, hides the complexity of network QoS mechanisms from the applications. The MASA QoS framework is able to support users with the ability to continue ongoing sessions even during handoffs and device changes (session mobility).

Similar research work was done within a project at the Lancaster University, UK. They investigated in Component-based Active Networks for Mobile Multimedia Systems with detailed consideration of mobile Ipv6 0. Associated with mobile Ipv6 and active networking, most of the research activities discuss ad-hoc routing protocols and n table-driven routing protocol (proactive) e.g. destination-sequence distance-vector (DSDV) routing or clustered gateway switch routing (CGSR) and n source initiated on-demand routing protocol (reactive) e.g. ad hoc on-demand distance vector (AODV) routing, dynamic source routing (DSR) and Active Source Routing.

Other research activities take care about ad-hoc networks, in which all the network nodes are mobile terminals. The connectivity among them is continues changing. Therefore, it is difficult to find one best ad-hoc routing protocol suitable for all circumstances. The project, described in 0, uses AN to support customization of routing protocols, where most suitable routing protocol can be chosen from multiple routing protocols according to the QoS requirements, security concerns, link characteristics.

To sum up, many research projects (e.g. 0, 0, 0) show, that future mobile networks can best fit for the adoption of active networks. The main benefit of active networks is the added flexibility. This flexibility is require for at least two points 0:

- Applications will evolve rapidly. The adaptation of lower layer infrastructure will be needed to optimize these applications.
- New wireless technologies and ad-hoc networks will require continuous adaptation of the networking layer.

Active networks can help to evolve separately different networking layers. So it is not necessary to introduce completely new networking infrastructure.

Other research projects (e.g. 0) handle in particular solutions for handover. They regard handover as a high frequency event in future macro, micro and pico-cellular mixed wireless mobile over-layer internetworking environment. It is a very important issue to support handover efficiently and reliably is a very important issue, which will directly influence the whole system performance. In particular, the focus of the research activities is an efficient and reliable handover scheme. Unfortunately, complete implementations and hands-on experiences are still missing.

12.6.2 Future directions in active mobile networks research and development

As stated in the introduction this document just gives a short introduction into the mobility aspects of active networking. A lot of issues still require further investigation, to name just a few:

- Tests about load balancing, practical experiences about scalability (number of access points, number of clients, number of services)
- Use cases with utilization of different services belonging to different QoS-classes (others than videostreaming, which is the only demonstration service)
- Management application: Monitoring as bases for accounting and billing
- Hand-over between different wireless access technologies
- Extension to heterogeneous networks consisting of fixed networks (different technologies) and mobile networks (different technologies)
- Legacy integration: how are Clients without FAIN-specific Software handled ?
- How behave Clients with FAIN specific Software in not-FAIN-networks?
- Extension to personal mobile networks,
- Evaluation in not closed environments (other scenarios than DINO park, ….)

More concrete and directly related to the WLAN demonstrator, the present functionality may be extended with

- Mobility Support for Applications using a Software Proxy Concept
- Content Adaptation for Mobile Users
- Reconfiguration Support for Mobile Terminals
- Application dependent decisions
- multi-layer aspects in mobile networks (e.g. for Quality of Service, Service Adaptation, Reconfigurable Radios)

12.7 REFERENCES

[1] Mobile FAIN Demonstrator: Generic Demonstration Scenario FAIN project, E. Pfeuffer, R. Schmid, C. Meyer, FAIN Internal Report. To be published.

[2]FAIN Applications in Mobile/Wireless Networks, E. Pfeuffer, R. Schmid, C. Meyer, FAIN project, FAIN Internal Report. To be published.

[3]Web service Distribution - An application scenario of the FAIN architecture, C. Klein, L. Mandl, FAIN project, Internal report. To be published.

[4]Design Specification for Mobile FAIN Demonstrator, R. Schmid, Carsten Meyer, Siemens AG, FAIN Internal report.

[5]Mobile Controlled Handover in Wireless LAN, August 11, 2002, Attila Weyland Günther Stattenberger Torsten Braun, LANMAN 2002, Stockholm

[6]Component-based Active Networks for Mobile Multimedia Systems, S. Schmid, J. Finney, A.C. Scott, and W.D. Shepherd.

[7]Active Network Technology and Reconfigurability, Christian Prehofer, DoCoMo Communications Laboratories Europe.

[8]A Survey of Active Network Research, David L. Tennenhouse, Massachusetts Institute of Technologym Jonathan M. Smith, University of Pennsylvaniam W. David Sincoskie, Bell Communications Research, David J. Wetherall, Massachusetts Institute of Technology, Gary J. Minden, University of Kansas.

[9]Survivable Mobile Wireless Networks: Issues, Challenges, and Research Directions, James P.G. Sterbenz, Rajesh Krishnan, Regina Rosales Hain, Alden W. Jackson, David Levin, Ram Ramanathan, and John Zao, BBN Technologies 10 Moulton Street, Cambridge, MA 02138, USA

[10]Technolgies for Future Mobile Networks, Dr. Christian Prehofer, DoCoMo Eurolabs, München.

[11][Prehofer-sdr-colloq an-and-reconfig-2002-06.pdf]

[12]Christian Tschudin, Henrik Lundgren and Henrik Gulbrandsen, Active Routing for Ad Hoc Networks, IEEE Communications Magazine, April 2000

[13]A. T. Campbell, M. E. Kounavis, and R. R.-F. Liao: Programmable Mobile Networks. Computer Networks, Vol. 31, No. 7, pg. 741-765, April 1999.

[14]Active Networks: Applications, Security, Safety, And Architectures, Konstantinos Psounis, Stanford University. Surveys, I E E E Communications The Design And Evaluation Of Network Services In An Active Network Architectural Framework, Niraj Prabhavalkar- A thesis submitted to the Graduate School-New Brunswick Rutgers, The State University of New Jersey in partial fulfillment of the requirements for the degree of Master of Science Graduate Program in Electrical and Computer Engineering Written under the direction of And approved by Professor Manish Parashar New Brunswick, New Jersey October, 2000

[15]Active Networks, An Integrating Technology for Research, Peter T. Kirstein, 14/9/00 LCS Conference.

[16]A Network Architecture for highly integrated Access Points for use by Multimedia Mobile Terminals, Juntong Liu, Computer Communication System Laboratory, Telecommunication System Laboratory, Department of Teleinformatics, Royal Institute of Technology, Stockholm, Sweden, March 1998, A thesis

submitted to the Royal Institute of Technology in partial fulfillment of the requirements for the Licentiate of Technology degree

[17]G. Dommety, A. Yegin, C. Perkins, G. Tsirtsis, K. El-Malki, M. Khalil, Fast Handovers for Mobile IPv6, draft-ietf-mobileip-fast-mipv6-04.txt, March 2002

[18]New Technologies in Support of Mobility, 10. Sept. 2002 Arto Juhola, VTT TECHNICAL RESEARCH CENTRE OF FINLAND, http://akseli.tekes.fi/Resource.phx/tivi/nets/netsaiheryhma1aseminaarikutsu.htx.liite.liitteet.2.ppt

[19]High Quality Mobile Communication, KIVS 2001 - Kommunikation in verteilten Systemen (Tagungsband), 2001, H. Hartenstein, A. Schrader, A. Kassler, M. Krautgärtner, C. Niedermeier,

[20]Component-based Active Networks for Mobile Multimedia Systems, S. Schmid, J. Finney, A.C. Scott, and W.D. Shepherd Distributed Multimedia Research Group Computing Department Lancaster University, UK, The 10th International Workshop on Network and Operating System Support for Digital Audio and Video, June 26-28 2000, Chapel Hill, North Carolina, USA

[21]http://www.ist-mobydick.org/
Moby Dick - Mobility and Differentiated Services in a Future IP Network
Project Number: IST-2000-25394

