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

Initial Active Network and Active Node Architecture

CEC Deliverable Nr :	**D2**
Deliverable Type :	**PU**
Dissemination:	**PU**
Deliverable Nature :	**R**
Contractual date :	**31st January 2001**
Actual date :	**30th May 2001**

Editor :	**Spyros Denazis**
Workpackage(s) :	**WP3**
Abstract :	Active Node Architecture and Design
Keyword List :	**Active Networks, Node Components, Control Framework, Security Framework, Virtual Environments, Demultiplexing**

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

Initial Active Network and Active Node Architecture

Editor :	**Spyros Denazis**
Document No:	**D2**
File Name	**WP3-HEL-027-D2.doc**
Contributors :	

Version :	**1.0**
Date :	**Wednesday, 30 May 2001**
Distribution :	**WP3**

The FAIN Consortium consists of:

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

The FAIN Consortium

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

Project Management

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

Torrington Place
London WC1E 7JE
United Kingdom
Tel +44 (0) 207 458 4463
Fax +44 (0) 207 388 9325
E-mail: a.galis@ee.ucl.ac.uk

Authors

Thomas Becker (GMD)
Spyros Denazis (HEL)-Editor
Walter Eaves (UCL)
Ducan Gabrijelcic (JSIS)
Alex Galis (UCL)
Richard Gold (GMD)
George Karetsos (NTUA)
Stamatis Karnouskos (GMD)
Antonis Lazanakis (NTUA)
Franci Mocilar (JSIS)
Arso Savanovic (JSIS)
Toshiaki Suzuki (HEL)
Kiminori Sugauchi (HIT)
Lukas Ruf (ETH)
Alvin Than (UCL)

Change History

Ver.	Date	Authors	Comments
0.05	15.01.01 and 30.04.01	Lukas Ruf (ETH)- Editor Peter Graubmann, Cornel Klein (SAG) for GENESYS: Hui Guo, Georg Carle, Thomas Becker, Richard Gold (GMD) for BANG, ANON, M0, JanOS Dusan Gabrijelcic, Franci Mocilar, Arso Savanovic (JSIS) for NETSCRIPT Christos Tsarouchis, Spyros Denazis, Toshiaki Suzuki (HEL) and Kiminori Sugauchi, Chiho Kitahara (HIT) for Openet, Click Router, ANTS, CANES, TEMPEST, JanOS Jessica Kornblum (UPEN) for SwitchWare Alvin Tan, Alex Galis, Walter Eaves (UCL) for ABLE, ARP, DARWIN Lukas Ruf, Matthias Bossardt (ETH) for ANN, EXOKERNEL Julio Vivero, Epi Salamanca (UPC) for ABONE Odysseas Pyrovolakis, Christos Chatziathanasiou, Yiannis Nikolakis (NTUA) for ALAN, Click Router Jan H. Laarhuis, Jerry van der Leur, Herman Pals (KPN) for PRONTO	Appendix A – Review of Active Networks projects
0.06	15/01/01 and 30/04/01	Richard Gold (GMD) – Editor ANTS: Peter Graubmann (SAG) BANG: Hui Guo (GMD) JanOS: Dusan (JSIS) Openet: Christos Tsarouchis (HEL) SwitchWare: Jessica Kornblum (UPEN) ABLE: Walter Eaves & Alvin Tan & Alex Galis (UCL) ANN: Lukas Ruf & Matthias Bossardt (ETH)	Appendix B- Evaluation and analysis of Active Network platforms
0.1	17.05.2001	Spyros Denazis (HEL) Toshiaki Suzuki (HEL) Kiminori Sugauchi (HIT) George Karetsos (NTUA) Antonis Lazanakis (NTUA) Thomas Becker (GMD) Stamatis Karnouskos (GMD) Arso Savanovic (JSIS) Ducan Gabrijelcic (JSIS) Franci Mocilar (JSIS)	Integration of the following internal reports into D2: · D2 ToC: WP3-HEL-016-D2-ToC (Ver 0.5) · R23 Report (Ver. 1.0) · RCF: · Security: WP3-JSIS-004-D2-Int-sec-1_0.doc (Ver.1)

		Walter Eaves (UCL)	1_0.doc (Ver.1) · DeMUX: WP3-HEL-026-D2-DeMUX-Int (Ver.1)
0.2	28.05.2001	Spyros Denazis (HEL) Walter Eaves (UCL) Alvin Tan (UCL)	· Insertion of R23.1 & R23.2 as appendices · Systems architecture & Implementation issues added · Editorial changes submitted by JSIS
0.3	29.05.2001	Spyros Denazis (HEL) Alex Galis (UCL) Walter Eaves (UCL)	· Executive Summary added by Galis · Appendix D added on idl specifications by UCL · Editorial changes submitted by UCL, GMD and JSIS
1.0	30.05.2001	Spyros Denazis (HEL) Alex Galis (UCL) Walter Eaves (UCL)	· Final Editorial changes and conclusion

Executive Summary

FAIN NETWORK OVERVIEW

FAIN Active Networks

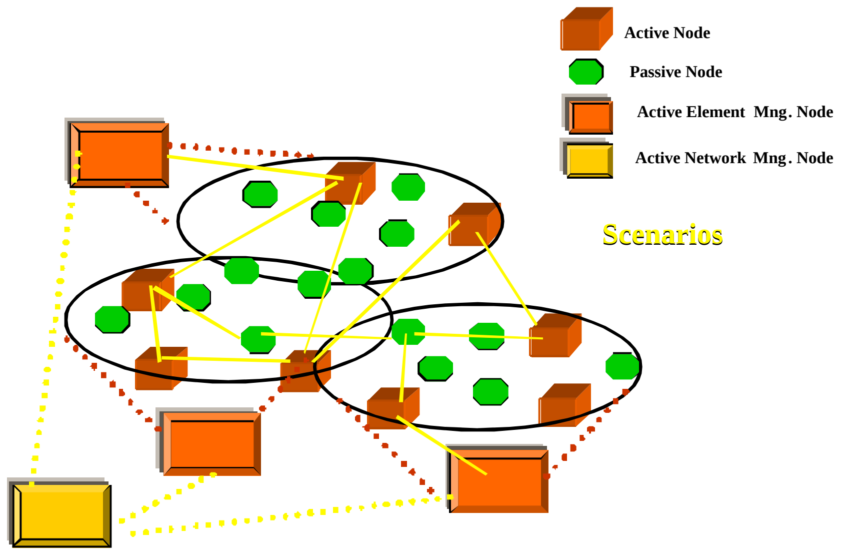


Figure 1 - Active Networks

The FAIN active network architecture defines active nodes, which provide full flexibility to the user to manage and provide active services.

The defining characteristic of an active node is the ability for users to load software components dynamically and offers dynamic programmability. An active node provides at least an execution environment, which provides the capability of running user-provided code. An important feature of active nodes is that any undesirable operation will have limited consequences. This can be achieved safely since each customer can in principle be provided with a dedicated active node and that customers who are sharing the same active node would be provided with a partition of the node resources via a private VPN.

Packets requiring active processing are marked in some way to allow correct handling by active routers. This allows the discrimination of active and conventional packets and the selection of an active node. Routing and node resources configuration in the active nodes could be achieved by setting policies at the network management level (element and network management nodes). Access to this functionality will be controlled and only possible via a strict API. Packets requiring active processing are marked in some way to allow correct handling by active routers. This allows the discrimination of active and conventional packets and the selection of an active node.

Figure 1 above exemplifies a configuration of an active network and its management nodes.

FAIN Active Nodes

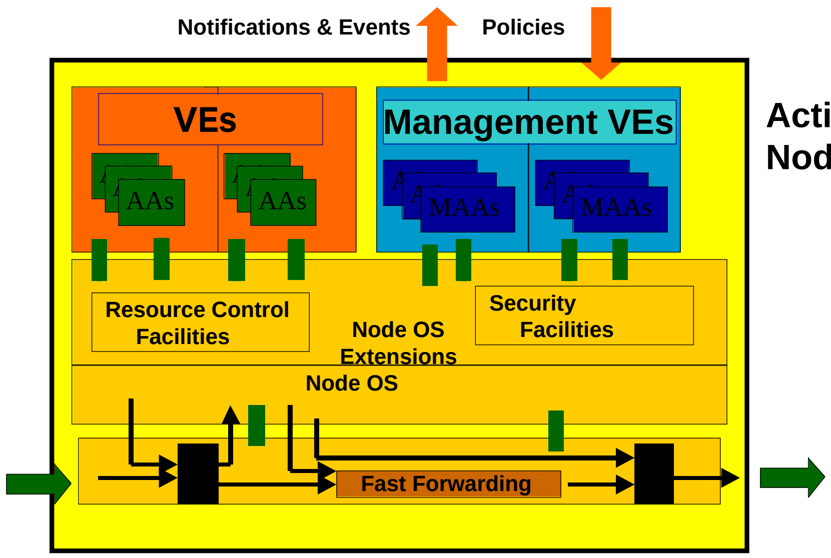


Figure 2 – FAIN Active Node

FAIN Reference Architecture consists mainly of AA, VE, EE and Node OS:

- Active Applications/Services (AA) are applications executed in Active Nodes.
- Execution Environments (EE) are environments where application code is executed. A privileged EE manages and controls the Active node and it provides the environment where network policies are executed. Multiple and different types of EE are envisaged in FAIN. EEs are classified into virtual environments VEs, where services can be found and interact with each other. VEs are interconnected to form a truly virtual network. In FAIN the interconnection of EEs through the use of the ANON protocol results into an overlay network which could be used for internodes signalling and control.
- NodeOS is operating system for active node and includes facilities for setting up and management of communications channels for inter –EEs and AA/EEs, manages the router resources and provides APIs for AA/EEs, isolates EEs from each other. Through its extensions the NodeOS offers:

- o Resource Control Facilities (RCF). Through resource control resource partitioning is provided. VEs are guaranteed that consumption stays within the agreed contract during an admission control phase static or dynamic.
- o Security Facilities- Main part about security is authentication and authorisation of using the resources and other objects of the node like interfaces and directories. Based on the policy profile of each VE security is enforced.
- o Application/Service code deployment facilities - As flexibility is one of the requirements for programmable networks partly realised as service deployment either on the fly or static, the NodeOS must support it.
- o Demultiplexing facilities - it filters, classify and divert active packets. Flows of packets arrive at the node and they should be delivered to the VE and consequently to the service inside the VE they are destined for.

Figure 2 describes the main design features of the FAIN nodes

In FAIN a number of node prototypes are under development as follows:

- o A high performance active node, with a target of 150 Mb/s
- o A range of flexible and very functional active nodes/servers, with the target on multiple VEs hosting difference EEs

The common part of the prototypes (the FAIN middleware) is the NodeOS with the relevant extensions.

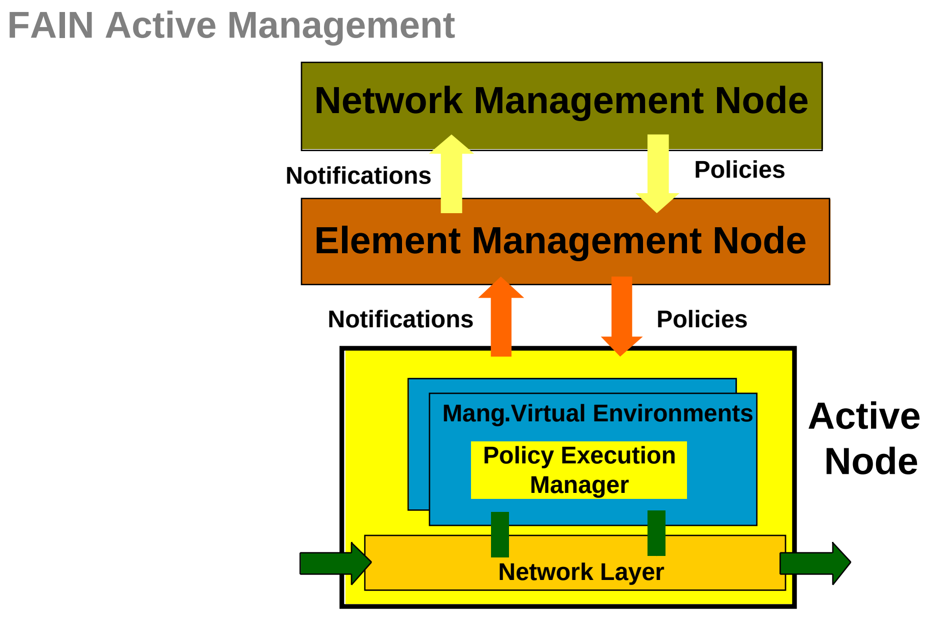


Figure 3 - Active Network Management

The management approach in the FAIN project is based on policies.

We envisage that the management of the active network will require:

- **Policies:** Design of management policies required to manage the active nodes and network

- **Management Components in the Active Nodes:** Design of management components for the active nodes, which will execute policies within an active node and which will monitor the node resources usage. The execution of policies means mapping target policies into node resource configurations
- **Management Nodes:** A set of management nodes that will mechanisms to enable network administrators to manage the active networks as a whole, including network policies set-up and processing.

The management components within the active node and the functions performed by these will determine the management capability of the active node and hence the extent to which management can be delegated to the node. The policy based management solution adopted by FAIN will serve to delegate management through the execution of policy. In this ways the policies will define the rules that will determine the behaviour of the active nodes. The execution of these policies will cause changes in the node and will determine how it delivers the services for users. The policies are defined and sent to the nodes by the management nodes, which deliver services. The management nodes will need to know what policies to send to which active nodes, when to send these and what results these have.

The network and service provider will need to by provided with management capability at the management nodes to monitor and control the whole and/or parts of the network.

The Network and/or Service providers will need to ensure the network resources are used efficiently and that enough resources are provided to meet users' demand. The Network and/or Service Providers will need to know if the active nodes are executing the right policies and that the active nodes are delivering the services correctly to users. As the delivery of services will require co-operation of a number of active nodes the network providers will need the means of managing the active nodes as a group of nodes and not individual nodes. They will need monitoring mechanisms for checking that correct policies are being defined and used in relation to the network before they are sent to the actual network. It will need to know what policies are currently loaded in the active nodes and what impact these are having on the network. It will also need to protect and monitor the security of the network. Therefore, the network/service provider needs a set of management mechanisms that will enable it to manage the network as a whole.

In FAIN we envisage that two types of management nodes provide these mechanisms:

- Element Management Nodes
- Network Management Nodes

The main difference in functionality provided by these two types of management nodes is in the policy types, which they could process and manage, in the sub-networks, which they cover and in the creation of management domains for different types of users as shown the figure 3.

Description of Deliverable

In the executive summary section we presented an overview of the FAIN system in order to connect this deliverable, D2, with the other ones, namely, D1 & D3. We have identified all the major components that provide a clear separation between the different deliverable and consequently the different workpackages these deliverables belong to.

D1 describes all the candidate applications and the operators' requirement while D3 deals with all management issues and dynamic service provisioning in the form of downloading code.

In contrast, the scope of D2 is to describe the design of the FAIN active nodes, the Resource Control, Security Facilities and the Demultiplexing Facilities.

More specifically, in section 2 we present a brief state of the art in programmable networks identifying the requirements and challenges that stem from achieving our objectives. In section 3, we provide an analysis of programmable networks by extracting all those aspects thereof that are relevant into defining the FAIN reference architecture. In section 4, we present our FAIN reference architecture model and its components that form the basis on which a detailed FAIN systems architecture is built. We argue that this model is sufficient to describe and explain other similar research efforts. In section 5, we further elaborate on the reference architecture by suggesting mappings of the “cube” model onto specific system architectures which in turn will be the basis for the FAIN implementation. Identifying the major components of the FAIN node architecture we continue to provide a detailed design for the resource control framework, security and demultiplexing in sections 6.1, 6.2, and 6.3, respectively. Although, most of the efforts described in D2 were focused on the design at this first stage of the project, in section 7, the first glimpse of implementation of the node design is discussed and some initial specifications are given in IDL. Finally, we conclude in section 8, where future steps are also discussed.

As a final note, D2 is mainly the result of merging the following internal reports as they are defined by the TA: R3 (R23) & R4. In addition, two more internal reports were produced, namely, R23.1 & R23.2 which raised the level of our understanding about AN that eventually resulted in the ideas of R23.

In particular the scope of R3 spans sections 1-5, while the scope of R4 spans sections 6 & 7.

Table of Contents

Table of Figures

Table of Tables

1 INTRODUCTION

It has been more than a decade since the introduction of programmability in the network elements (switches routers etc) as the basis for the rapid deployment and customisation of new services. It started in the field of telecommunications with the advent of Intelligent Networks (IN) [1] and later of Advanced IN [2] to eventually permeate data communications with the emergence of active networks [3] and open signalling [4] in the mid-nineties. As we are moving towards a global (common) network engineered to carry and integrate a variety of services programmability becomes more than ever the most important property of networks to the degree that now we talk about *programmable networks*[1] and the *programmable Internet.*

Advances in programmable networks are driven by a number of requirements that are profoundly impacted by an emerging enterprise model [5] as well as by new business actors and roles. More specifically, we are moving away from the vertical integrated approach of building products to a horizontal one composed of layers of different scope and focus.

The former model allowed only a few players to dominate the market while new services were difficult and time-consuming to introduce due to slow progress in standardisation thereof. In contrast, the horizontal model unleashes fresh market forces as it increases market granularity. New specialised players may concentrate on and contribute to specific layers of the model offering their services to layers above via a set of open interfaces while relying on (open) interfaces residing in lower layers. To this end, we can construct a truly open service platform that represents a marketplace wherein services and service providers compete while customers may select and customise services according to their needs.

[1] The term programmable networks is used by the Opensig community to characterise the networks built based on the principles they promote. In this document to distinguish between these networks and *active networks* we will use the term *opensig networks*. To refer to both active networks and Opensig networks we will use the term programmable networks.

However, such a scenario is not a trivial research exercise as it presents a series of technical challenges in designing and building a network infrastructure capable of supporting such a business model and meeting the new requirements. FAIN is such a research project that aspires to design and implement innovative network architecture. This part of the document (Sections 2-4) lays the foundations and the concepts involved in such an objective. The ideas and directions presented hereafter have been influenced by two major research initiatives; the active networks [3] and the Opensig community [4]. Approaching each one of them we argue that although they seem from the outset antagonistic to each other, they are rather convergent towards a common architecture model dealing with similar if not identical concepts. We will also argue that observed differences is due to different implementations mainly dictated by the choice of technologies as solution enablers for specific problems and trade-offs.

More specifically, in section 2 we present a brief state of the art in programmable networks identifying the requirements and challenges that stem from achieving our objectives. In section 3, we provide an analysis of programmable networks by extracting all those aspects thereof that are relevant into defining the FAIN reference architecture. In section 4, we present our FAIN reference architecture model and its components that form the basis on which a detailed FAIN systems architecture is built. We argue that this model is sufficient to describe and explain other similar research efforts. In section 5, we further elaborate on the reference architecture by suggesting mappings of the "cube" model onto specific system architectures which in turn will be the basis for the FAIN implementation. Identifying the major components of the FAIN node architecture we continue to provide a detailed design for the resource control framework, security and demultiplexing in sections 6.1, 6.2, and 6.3, respectively. In section 7, the first glimpse of implementation of the node design is discussed and some initial specifications are given in IDL. Finally, we conclude in section 8, where future steps are also provided.

2 REVIEW STATE OF THE ART IN PROGRAMMABLE NETWORKS

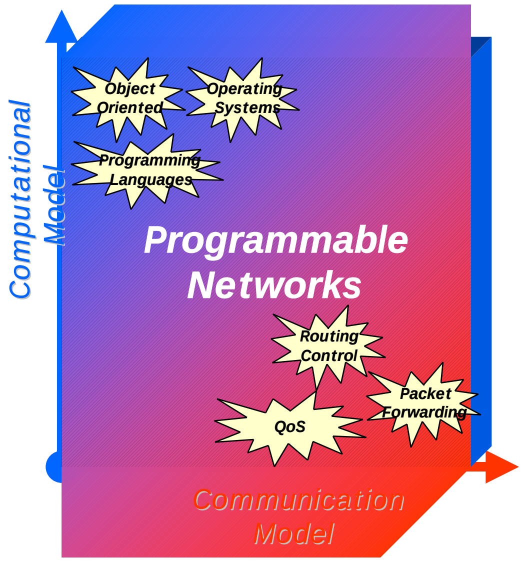


Figure 2-1: Programmable Networks

We are witnessing a networking paradigm shift from the flat, one-dimensional communication model to a two-dimensional model with the inclusion inside the network of the computational model (Figure 2-1). This two-dimensional model defines the problem space of what we call *programmable networks*. Traditionally, part of the communication model has been packet header processing and forwarding, quality of service and congestion control mechanisms. Programmability along this dimension has been exercised by introducing service models like ATM or Diffserv [6] operating at the transport plane and then using the control plane to customise them resulting in different forwarding behaviours as perceived by the users.

Emerging active technologies in the area of programming languages, object oriented and distributed programming, and operating systems made it possible to meet constant user demands for more network functionality and customisation that goes beyond the communication model. Programmability along this dimension is exerted by treating the network element (router, firewall, switch etc) as a place where advanced computations may take place.

Two distinct schools of thoughts can be identified out of research efforts dealing with this problem space. The first school of thought, emerged from the Opensig community, established through a series of international workshops while the second one, Active Networks, is the result of a series of projects under the auspices of DARPA. An exhaustive study of the state of the art whereby individual projects belonging to either school are reviewed and conclusions are drawn, may be found in Appendix A: and Appendix B: and the references therein.

2.1 The Opensig & The IEEE P1520

The motivation behind Opensig networks has been the observation that monolithic and complex control architectures are made up from basic components that can form a set of elementary network services. Modelling these services as individual objects by employing a high level programming language for the implementation and manipulation of the services-objects, results in a new generation of network architectures with some new desirable features: interoperability between different networking technologies, programmability for the creation of new services, and open interfaces across the control plane [7].

Eventually, a number of results out of the Opensig community were formalised by the IEEE Project 1520 standards initiative programmable network interfaces and its corresponding reference model [8]. P1520 advocates the need for open interfaces as the basis for service composition (creation). The interfaces are structured in a layered fashion offering their services to the layer above. Each layer defines what is termed as the *level*. Each level comprises a number of *entities* in the form of algorithms or objects representing logical or physical resources depending on the level's scope and functionality. This approach gives rise to the reference model (RM) depicted in the left part of Figure 2-2.

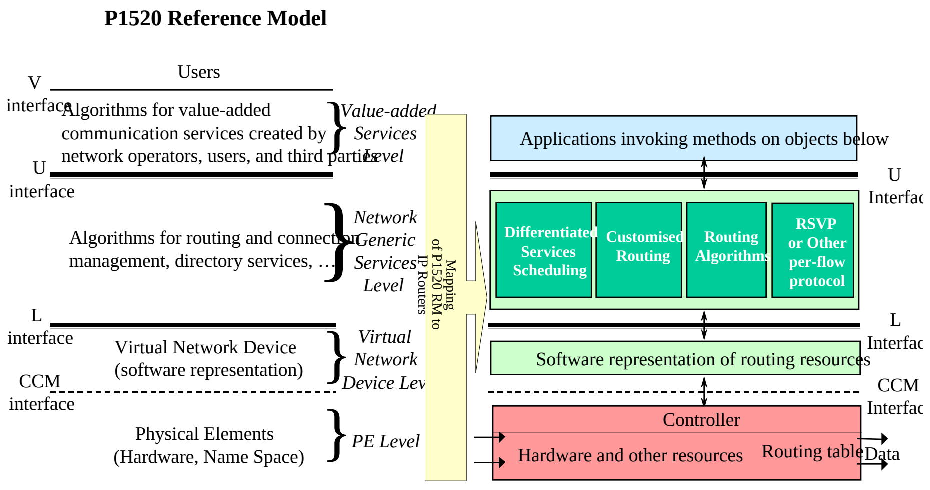

P1520 Reference Model
Users
V interface
Algorithms for value-added communication services created by network operators, users, and third parties
Value-added Services Level
U interface
Algorithms for routing and connection management, directory services, …
Network Generic Services Level
L interface
Virtual Network Device (software representation)
Virtual Network Device Level
CCM interface
Physical Elements (Hardware, Name Space)
PE Level
Mapping of P1520 RM to IP Routers
Applications invoking methods on objects below
U Interface
Differentiated Services Scheduling
Customised Routing
Routing Algorithms
RSVP or Other per-flow protocol
L Interface
Software representation of routing resources
CCM Interface
Controller
Hardware and other resources
Routing table
Data

Figure 2-2: The P1520 Reference Model and mapping to IP routers

More specifically, P1520 RM distinguishes among the following four levels:

- The *physical element* (PE) level consisting of entities such as hardware and the device architecture that actually reflects upon the supported capabilities;
- The *virtual network device level* (VNDL) which logically represents resources in the form of objects (entities); isolating the upper layers from hardware dependencies or other proprietary interfaces;

- The *network generic services level* (NGSL) consists of entities in the form of distributed algorithms that bind (interconnect) together the objects of the VNDL level according to specific network functionality, e.g., routing, connection set-up;
- Finally, the *value-added services level* (VASL) includes entities in the form of end-to-end algorithms that enhance the generic services of the NGSL level; providing user-oriented features and capabilities in the applications.

The four levels give rise to four interfaces, namely, CCM (Connection Control and Management), L (lower), U (upper), V (value-add) interfaces. The CCM interface is actually a collection of protocols that enable the exchange of state and control information at a very low level between the device and an external agent. The L-interface defines an API that consists of methods for manipulating local network resources abstracted as objects. CCM and L-interfaces fall under the category of node interfaces. The U-interface mainly provides an API that deals with connection set-up issues. As in the case of the L-interface, the U-interface isolates the diversity of connection set-up requests from the actual algorithms that implement them. Finally, the V-interface (not shown in Figure 1) provides a rich set of APIs to write highly customised software often in the form of value-added services. Additionally, U- and V-interfaces constitute network-wide interfaces.
The P1520 Reference Model (RM) provides a general framework for mapping programming interfaces and operations of networks, over any given networking technology. Mapping diverse network technologies and their corresponding functionality to the P1520 RM is essential.

Initial efforts focused on telecommunication networks based on ATM and introduced programmability in the control plane, later they extended these principles to IP networks and routers. The right side of Figure 2-2 illustrates a mapping of the P1520 RM to IP routers. Nowadays there is an effort to create a framework for designing interfaces not just for routers but for any network element (NE) like a router, or an optical switch etc that participates in forwarding traffic [9].

2.2 Active Networks

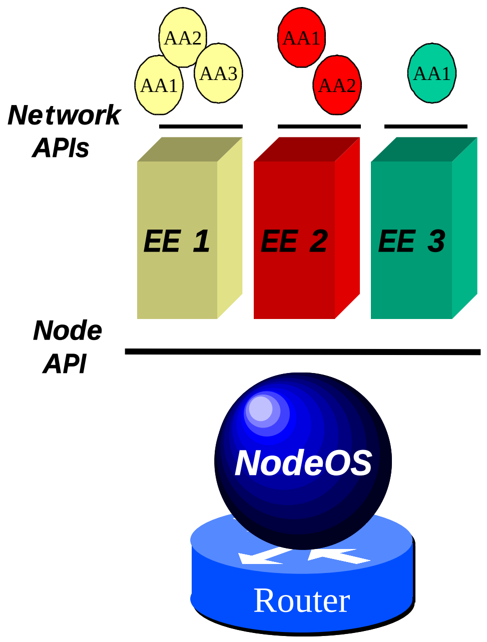


Figure 2-3: The Active Node Architecture

Active Networks transform the store-and-forward network into store-compute-and forward. The innovation here is that packets are no longer passive but rather active in the sense that they carry executable code together with their data payload. This code is dispatched and executed at designated (active) nodes performing operations on the packet data as well as changing the current state of the node to be found by the packets that follow.

In this context, two approaches can be identified based on whether programs and data are carried discretely, namely within separate packets (out of band) or in an integrated manner, i.e. in-band.

In the discrete case the job of injecting code into the node from the job of processing packets is separated. The user or network operator first injects his customised code into the routers along a path. Then the data packet arrives, its header is examined and the appropriate pre-installed code is loaded to operate on its contents [10], [11].

Separate mechanisms for loading and execution may be required for the control thereof. This separation enables network operators to dynamically download code to extend a node's facilities, which in turn they become available to customers through execution.

At the other extreme lies the integrated approach where code and data are carried by the same packet [12]. In this context, when a packet arrives in a node code and data are separated, code is loaded to operate on the packet's data or change the state of the node.

A hybrid approach has also been proposed [13].

As in the case of P1520 active networks have also proposed their own reference architecture model [14] depicted in Figure 2-3. According to it an Active Network is a mixture of active and legacy (non-active) nodes. The active nodes run the node operating system (NodeOS) –not necessarily the same- while a number of Execution Environments (EE) coexist at the same node. Finally a number of active applications (AA) make use of services offered by the EEs.

The NodeOS undertakes the task of simultaneously supporting multiple EEs. Accordingly, its major functionality is to provide isolation among EEs through resource allocation and control mechanisms, and providing security mechanisms to protect EEs from each other. It may also provide other basic facilities like caching or code distribution that EEs may use to build higher abstractions to be presented to their AAs. All these facilities are encapsulated by the Node interface through which EEs interact with the NodeOS. This is the minimal fixed point at which interoperability is achieved [15].

In contrast EEs implement a very broad definition of a Network API ranging from programming languages to virtual machines like the Spanner VM in Smart Packets and bytecodes, to static APIs in the form of a simple list of fixed-size parameters etc [16]. To this end, EE takes the form of a middleware toolkit for creating, composing and deploying services.

3 ANALYSIS OF PROGRAMMABLE / ACTIVE NETWORKS

This section is an attempt to extract all those aspects of programmable networks that are interrelated for defining the FAIN reference architecture. A detailed review and analysis of the Active Networks projects and platforms is described in Appendix A: and Appendix B:, respectively.

3.1 Requirements

The main objective of programmable networks is the rapid creation, deployment and prototyping of network services. Addressing this objective and making the network truly programmable the following general requirements need to be met first.

- Flexibility
  - Extensibility
- Performance
- Security
- Interoperability
  - Portability

The problem here is that these requirements are usually antagonistic with each other. For instance, the more flexibility is engineered in a network the greater the problems become in maintaining acceptable levels of performance. In contrast, increasing performance levels may require some proprietary solution that may not be available everywhere in the network resulting in a decrease of the level of the interoperability.

Accordingly, there is a trade-off among these requirements. The exact level is the result of customer demands and market needs[2]. It is this trade-off that eventually influences programmable network architecture and guides the engineer through design and implementation choices. This level of trade-off also defines the level of programmability built in a network. This is evident in all projects of the two schools of thought where their main differences can be traced in the level of programmability that each one of them selected to instil their network prototypes with [17], [18].

3.2 Architecture vs Implementation aspects

In so far we have seen that one of the components of the AN reference architecture is the EE. The question here is what exactly is an EE, what is it comprised from and are these elements part of the architecture or part of its chosen implementation? Scanning the literature we have found a variety of answers regarding the exact characteristics of an EE.

Conceptually an EE is the active network's programming environment [19] when instantiated it becomes the runtime environment of a process or a process itself [20]. This programming environment may be centred on a particular language and may export some API that encompasses elements like a Java Virtual Machine [19], [14], toolkits used for building AAs (services) [10], [20] or even interfaces to access generic services that AAs may customise building value added services. EEs have also been proposed as extensions of the NodeOS for those that are allowed to be extensible [15]. The latter has an impact on where to draw the boundary between EE and NodeOS known as the Node interface.

In contrast, the AN reference architecture [14] is designed for simultaneously supporting a multiplicity of EEs at a node. This implies that EEs are treated as principals based on which authentication, authorisation and resource control takes places. Services and users that use an EE are represented by this principal, which is the only valid entity allowed to access NodeOS facilities. Prototypes proposed in [21], [22] fit in this picture.

Finally, EEs are characterised not by the choice of technologies but rather by the services they offer and the architectural plane they operate at, namely, control, management, and transport [23], [24].

One of the conclusions that we may draw from this discussion is that it is very difficult to come up with a common architecture that encompasses most, if not all, approaches as in the majority of cases the boundaries between architecture and implementation are blurred. Moreover, there is always an ambiguity regarding the terminology whereby the same term is used with different connotations.

On the other hand, such reference architecture is valid only if it provides explanations to most research efforts and is still valid to account for different implementation approaches. This is the subject of the next section.

[2] In FAIN customer demands and market needs that will eventually dictate the trade-off level are expected to come from WP2

4 THE FAIN REFERENCE ARCHITECTURE

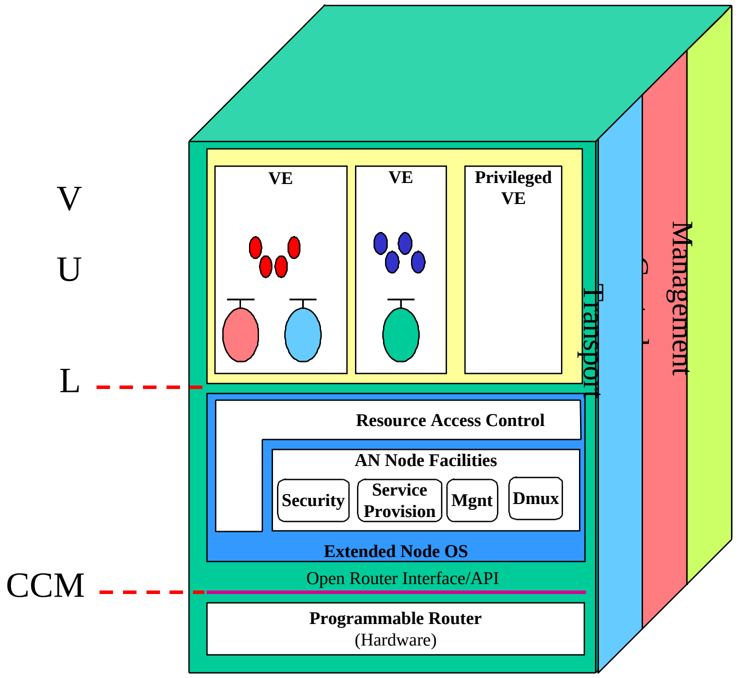


Figure 4-1: The FAIN reference architecture

In an attempt to establish a FAIN reference architecture independent of any implementation contexts we make the observation that EEs can be classified according to the following three categories:

- Technologies
- Services
- Virtual Environments

Technologies are all those toolkits like programming languages, specialised hardware, virtual machines e.g. JVM, that allow someone to implement his/her designs in order to achieve the desired level of the programmability.

Services are what the EE has to offer usually in the form of interfaces. Services may well be extensible in the sense that EEs offer another interface (service) through which the service under consideration can be extended. For example a service uses the code distribution mechanism to download code extensions. The extension then becomes part of the overall service interface.

Finally, EEs act as virtual environments. This is an abstraction in itself as it is used for resource management and control. Therein services may be found and interact with each other. From the NodeOS viewpoint is the principal responsible for the consumption and use of resources, the recipient of sanctions in the event of policy violations and the entity that is legal to receive authorisation.

We argue that from the viewpoint of customers and users all these three categories encapsulate what is considered as the Network API exported by the EE in [16] and this is the definition for the Network API that we will use hereafter.

Of course, an EE may be treated as a mixture of all these categories but for the purposes of creating a reference architecture we feel that this is confusing. Accordingly, in order to distinguish between EEs and VEs we treat every EE as a specific choice of a technology, e.g. JVM, and as such dependent on the particular choice of the implementation. To this end, any references to technologies and how these may be used to realise services or make the network programmable to achieve the desired level of programmability have been left out as we consider this an implementation issue. Figure 4-1 depicts our proposal as the FAIN reference architecture.

Our approach is from the service viewpoint in the terms of interfaces and how these may be combined to compose other services thereby creating a set of building blocks to create a truly service platform. However, the methodology to compose services falls outside the scope of FAIN although this capability is necessary to increase re-usability and service programmability. One way to go about is to adopt the P1520 approach and define U and V interfaces or use an “active” language like PLAN of Switchware. These services may solely lie in the control or management plane or may traverse all planes.

Services may be implemented anywhere in the node by any means of technology (EE) and they are invoked by their interfaces. To this end, the architecture is separated from implementation that follows.

In contrast, a virtual environment (VE) provides a place where services may be instantiated and used by a community of users and stay isolated from other communities. Within a VE many technologies, namely EEs, may be used to implement and/or instantiate a service. However, using a diversity of technologies within a VE has a detrimental effect on interoperability requirement but this may be avoided by choosing a common interface language or representation that acts as a wrapper to the proprietary one. To this end, many technologies (active or not) may co-exist in the same VE and used in a synergistic way. The details of this synergy are an implementation issue but some preliminary ideas may be found in [15] when the *concurrency model* is discussed. Finally, when VEs are connected together they form a true virtual network.

Another property of the reference architecture is that it makes no assumptions about how “thin” is a VE. It may take the form of an application, or a specialised service environment e.g. video on demand, or even a fully-fledged network architecture as proposed in [23], [24]. Moreover, a VE may coincide with an implementation that is based only on one technology e.g. Java technology. In either case this is a design decision dictated by customer requirements[3].

Out of all the VEs residing in a node there must be a privileged one that is instantiated automatically when the node is booted up and serves as a back door through which subsequent VEs may be created through the management plane. This privileged VE should be owned by the network provider who has access rights to instantiate the requested VE on behalf of a customer. From this viewpoint the creation of VEs becomes a kind of meta-service.

The other major and most important component of the reference architecture is the NodeOS. It offers all those facilities[4] that are necessary to keep all the other components together, and supports,

- Resource Control

  Through resource control resource partitioning is provided. VEs are guaranteed that consumption stays within the agreed contract during an admission control phase static or dynamic.

- Security

  Main part about security is authentication and authorisation of using the resources and other objects of the node like interfaces and directories. Based on the policy profile of each VE security is enforced.

- Management

  Management at the element level is provided through monitoring tools etc.

- Code Deployment

[3] D1 of WP2 provides these kind of requirements to justify certain design decisions

[4] We use here the word facilities to refer to services offered by the NodeOS to VEs and distinguish from services found inside EEs.

As flexibility is one of the requirements for programmable networks partly realised as service deployment either on the fly or static, the NodeOS must support it. Through code deployment mechanisms dynamic service provision is supported

- Demultiplexing

  Flows of packets arrive at the node and they should be delivered to the VE and consequently to the service inside the VE they are destined for. Implementation-wise, this means that this place may be hardware, a process in kernel space or in user space.

All these facilities in the NodeOS co-operate to deliver the overall functionality of the NodeOS to achieve its goals. One question that is relevant here is how much functionality should be put in the NodeOS and how much should be left to the VEs. We believe that this is related to the degree of extensibility a NodeOS is designed to achieve. For instance, a NodeOS may opt for a slim version whereby a sophisticated code deployment facility is not supported and leave this choice to the VE itself ending up with each VE using its own code deployment. The difference between the two approaches is that in the first case VEs do not have to start as empty boxes building everything from scratch while in the second case they can rely on NodeOS facilities to build on top of them.

Under any scenario, the exact choice has an effect on how the resource control is designed as the NodeOS needs to charge the VEs for the use of these facilities since their use even if it occurs at the OS level still consumes resources. Note that limited extensibility in the NodeOS does not necessarily means limited extensibility at the Active node as this may be realised within the VE possibly at the cost of the performance.

Between VEs and NodeOS lies the node interface that encapsulates all the facilities offered by it. Its objective is to create programmable abstractions of the underlying node resources, whereby third-party service providers, network administrators, network programmers or application developers can influence or extend node control through the use of higher-level API's. The requirements applicable to the node interface are exactly those of section 3.1. Meeting these requirements, it is essential to define a design framework that addresses all the issues involved in the node interface specification [25]. We believe that the design framework pursued by the P1520 IP sub-working group [9] points to the right direction while it creates opportunities for contributions by FAIN project.

Finally, between the NodeOS and the hardware NE there might be the open router interface, which may be a control protocol like GSMP. Its scope coincides with the scope of the CCM interface of P1520. Figure 4-1 also depicts a mapping of the P1520 RM onto the FAIN reference architecture as it was argued in the discussion above.

The FAIN reference architecture is the starting point from which a detail node architecture specification will follow. Accordingly, it should be complemented by the system architecture design & specification and the requirements. This, together with customer/user/application requirements will determine the degree of programmability to be built in the node and the choice of technologies. Section 5 provides the initial designs of a detailed FAIN node architecture while it discusses candidate configurations as mappings of the reference architecture onto actual systems. Based on this active node architecture, individual components thereof are detailed in subsequent sections.

5 FAIN SYSTEMS ARCHITECTURE

5.1 Overview

FAIN is faced with a wide variety of design and engineering options that is further complicated by the wide range of network environments in which an active node may operate. Different network environments impose different requirements that eventually influence the level of programmability delivered by the AN node.

In recent years we have seen a profound change in network technologies and topologies with IP being the common denominator in all of them. New hardware built around optical technologies has made the physical layer faster than ever thus creating a core network that is capable of carrying traffic at terabit rates.

In contrast, access networks are required nowadays to receive different types of traffic, interface with a wide range of link layer protocols and deal with varying service demands.

Furthermore, customers demand greater granularity of customisation to their services while service and network providers are looking for means for greater customer differentiation.

The end result of all this activity is there is a push for introducing more and more intelligence inside the network. However, distributing the right amounts of intelligence at specific areas of the network is a challenging task.

It is obvious that designing a system where one size fits all is very difficult if not impossible to achieve. Therefore system design should be carried out in relation to a specific context that eventually dictates different sets of requirements. In the next and subsequent sections section we elaborate on this subject by describing a network scenario that FAIN adopts.

5.2 Design Requirements

In FAIN we base our choices on a specific network scenario that is supported by the recent technological advances. In addition, we take into account the FAIN enterprise model complemented by the operators' AN requirements described in D1 [26]. According to this scenario, IP networks are expected to converge to a two-tier network comprised of few very fast all optical-based network cores surrounded by a multitude of intelligent access networks.

Fully featured AN nodes are most likely to be found at the edges of the core where the demand for intelligence is far greater than in the core. These edge devices will act collectively as intelligent multiplexing/demultiplexing entities delivering traffic to the proper environment residing at the same or an adjacent device for further processing or simply forward it to the core for fast delivery to the correct destination after performing the proper mapping between customer demands and certain QoS-enabled communication channels.

Programmability in this context ranges from the creation of the processing environment and the downloading and/or execution of the code if carried by the packet to the dynamic configuration of the behaviour of the access nodes to meet the requirement of the service.

In contrast, intelligence in the core devices is limited as it is restricted by the ability to match processing speeds with the high communication speeds. It likely to appear as highly sophisticated intelligent hardware dedicated to specialised tasks, e.g. video transcoding, and only if it has been proven that it is not at the cost of performance.

Programmability in this context will be realised as control and management functionality. Such functionality will include the ability to setup channels supporting certain level of QoS, sophisticated means of network monitoring, wavelength table manipulation etc.

The ability to customise the FAIN node with respect to the environment that is required to operate within implies an architectural design that is ***modular***. Active networks and the technologies thereof support modular architectures. To this end, the FAIN node should provide the necessary interfaces that allows its user(s) to select and download the right EE for the right task, to extend the OS by downloading components necessary to support high level network services etc.

In contrast, from the FAIN enterprise model we are able to extract a simplified model of the operation of the FAIN active node in the form of a use-case diagram depicted in Figure 5-1. The purpose of this model is to identify those activities that are the most important during the operation of the node.

More specifically, there are five actors, namely, a Service provider, Network Operator, Node Operator, Client and Server. No use-case is given for the interaction between the service provider and the server, but servers negotiate for a service level from the service provider, who negotiates in the “Service Provisioning” use-case with the network operator.

The network operator is in charge of the “Management” activity. His knowledge of the network comes from the node operators, who are responsible in the “Control” use-case for managing and monitoring the nodes they operate.

Finally, the client and server interact with one another through the “transport” use-case.

These, then, are the three planes of the cube model and a simplified statement of the relationship of the actors to them.

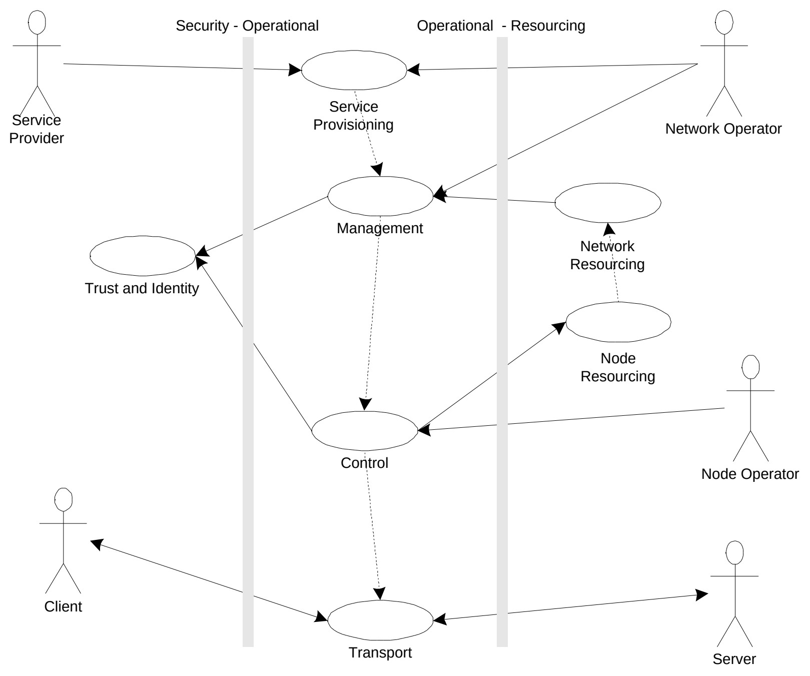


Figure 5-1: The Active Node Operation Use-Case

The point of this use-case diagram is to make explicit to the three activities: establishing “Trust and Identity” and “Node Resourcing” and “Network Resourcing”.

It is these three use-cases that require the most extensive engineering and the design of the architectural components that participate in these activities is elaborated in Section 5. Moreover, detailed requirements pertaining to each one of them are given separately and in conjunction with the description of each one of them.

In the next section we focus on certain examples of system and node architectures that take into account the design requirement for a modular architecture. We also link them with the FAIN reference architecture, namely, the cube model, as candidate realisations thereof.

5.3 System Architectures as realisations of the Cube Model

In this section, we are going to elaborate on a spectrum of design choices of how to realise the Cube Model. To achieve this we need to address the following two issues:

a) How to map overall AN functionality on a specific system architecture

b) How to map the Cube Model on a specific system architecture

These two issues provide complementary views of a systems architecture. The first is about ways of distributing overall FAIN node functionality in different systems, while the second one defines the relationship between VEs and EEs and how these may be mapped on to a node architecture.

5.3.1 FAIN AN functionality vs System Architecture

Active nodes must be able to:

- Intercept IP packets as they leave one network for another.
- Modify the packets, principally to inject code and remove code.
- Perform operations:
  - On the contents of the packets
  - On routers and network devices over which they have authority.
- Forward the packets to the next network

The ability to perform operations on packets and neighbouring network nodes requires that the code delivered by active packets be executed in a safe context: known as an execution environment, EE. This is a type of sandbox. It ensures only trusted code can be executed and that the operations are constrained in the use of resources within the active node. The active node must also constrain the code operations from accessing resources in the neighbouring network.

Active nodes will be used for many networking services, namely:

- Establishment of virtual private networks
- Preferential treatment of data flows
- Active caching mechanisms
- Implementation of new protocols
  - Adaptive multicast
  - Replicated networks
  - Secure transaction networks
  - Customer managed networks

Active nodes can operate in two modes. Either in a distributed fashion, where each node performs a specific function and the active nodes forward packets to one another to transform the packet payloads; or, in a centralised one, when the active node is capable of performing many functions, it can configure itself internally to provide a data path that transforms data as required.

For instance, we may decide that the reference architecture will not be implemented on the same physical node but on a physical configuration that comprises an access network router and a series of servers connected to it. This choice may have been dictated by the requirement whereby corporate customers or content providers require their own physical resources opting for a hard separation. Under this scenario, part of the VE that requires computational processing beyond packet forwarding is instantiated in their own server while the communication aspect that consists of communication resources is instantiated in the router by establishing a virtual path or joining a diffesrv based service.

5.3.1.1Distributed

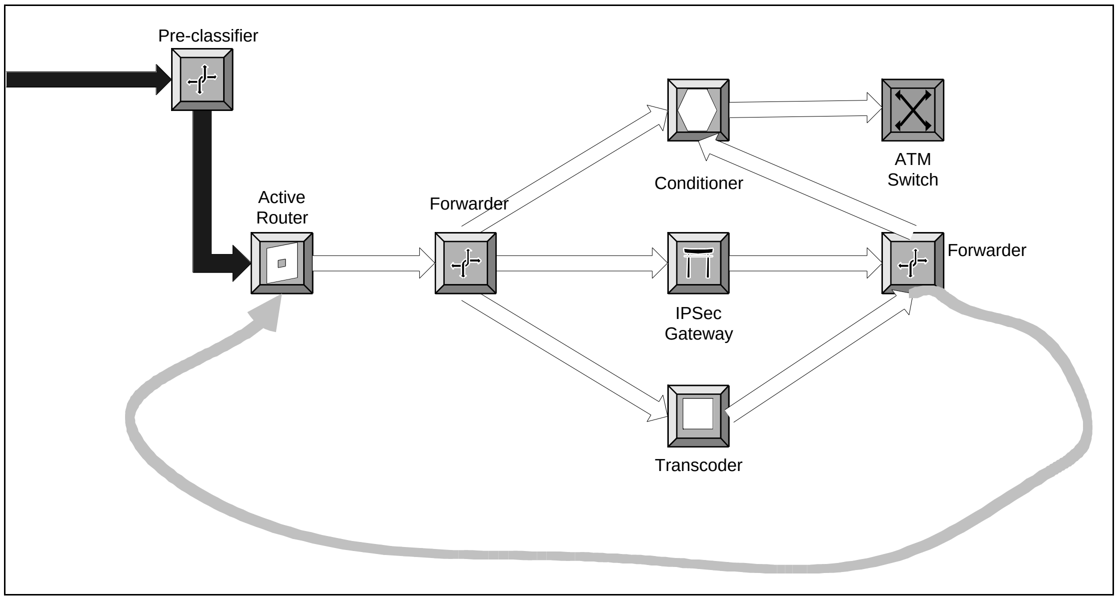


Figure 5-2 Dsitributed Operation of Active Node Facility

In Figure 5-2, there is an example of how an active router is used to forward packets onto a number of processing paths. There are a number of network elements, IPSec gateway, transcoder and traffic conditioner which are used together to transform the packet data.

The active router classifies packets and sends them on the right path. For example, IPSec packets would go through the first forwarder, then the IPSec gateway, then to the last forwarder, back to the active router on to the first forwarder and then to the traffic conditioner.

Another path might be to go through the transcoder and then to the conditioner.

5.3.1.2Centralised

More sophisticated active network nodes will be able to perform many network functions within their own address space and would not need to use IP packet addressing to forward packets from one data transformer to another.

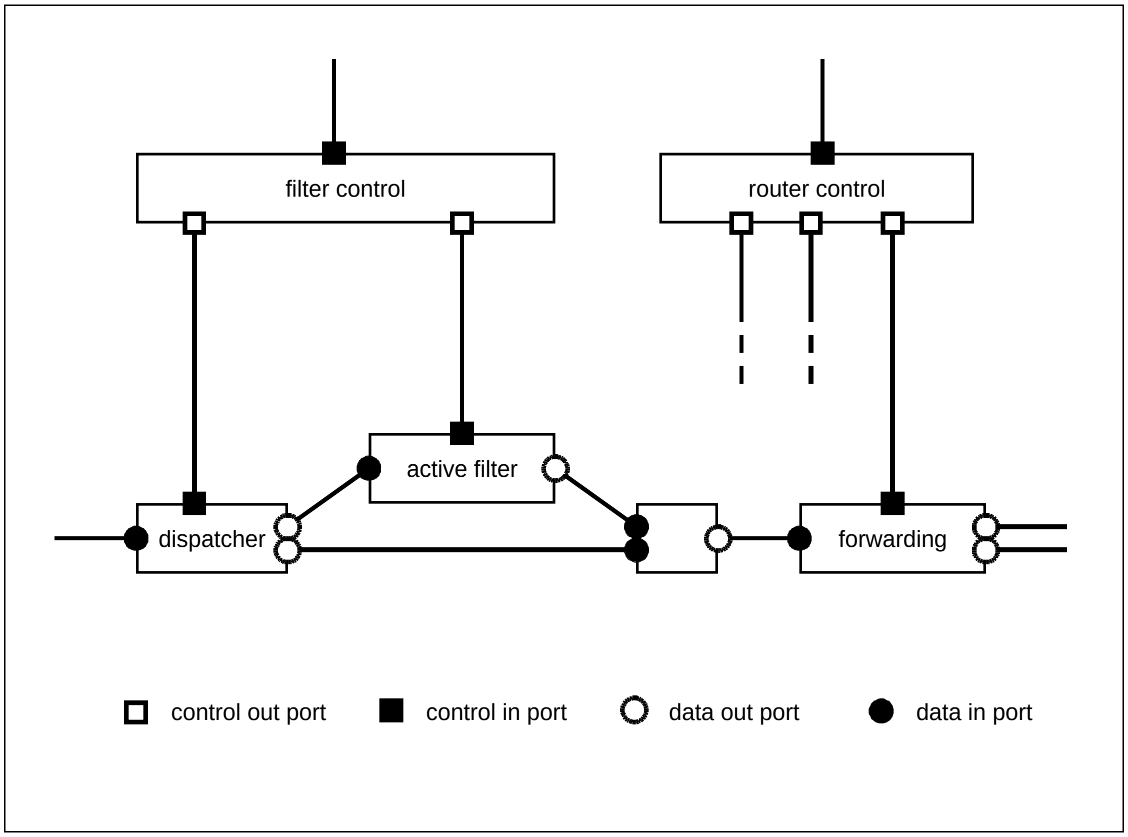


Figure 5-3 Active Node with many data processing components

Figure 5-3 is an illustration of an active node that contains an active filter.

5.3.2 Cube Model vs System Architecture

The realisation of the cube model may be carried out in a number of ways.

5.3.2.1Custom Packet Processor Systems

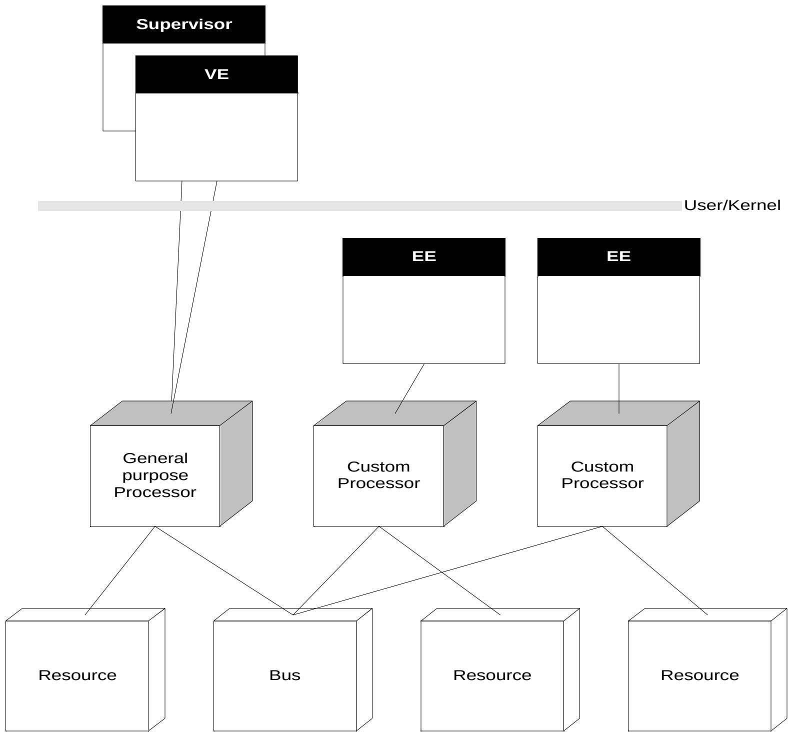


Figure 5-4 Custom Packet Processor

This type of system has autonomous packet processors, which are loaded by a general purpose processor. Each custom processor provides at least one execution environment. The Intel IX architecture is an example of this sort of technology. Such systems are suited for high-speed operation in the transport plane. The custom processors have a limited ability to interact with the general purpose processor, if they are damaged or corrupted the main processor should be able to recover.

5.3.2.2 Generic Operating System with Packet Processing Extensions

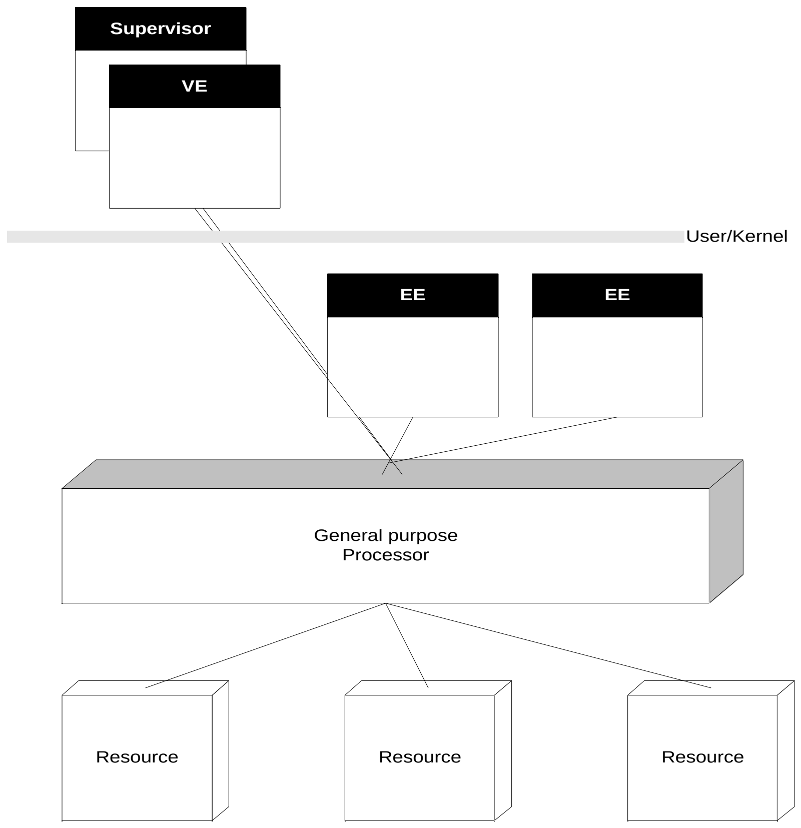


Figure 5-5 Generic Operating System with Packet Processing Extensions

This sort of system uses modules loaded in the operating system kernel to provide packet processing facilities. This is the way in which IPSec gateways have been implemented on Unix systems. The execution environments are kernel modules, which share address space with the general purpose processor and are easier to manage than an autonomous processors. There is little isolation between the packet processing extensions and the general processor, so any corruption of the extension will almost certainly corrupt the general processor.

5.3.2.3 Generic Operating System with Packet Processing Applications

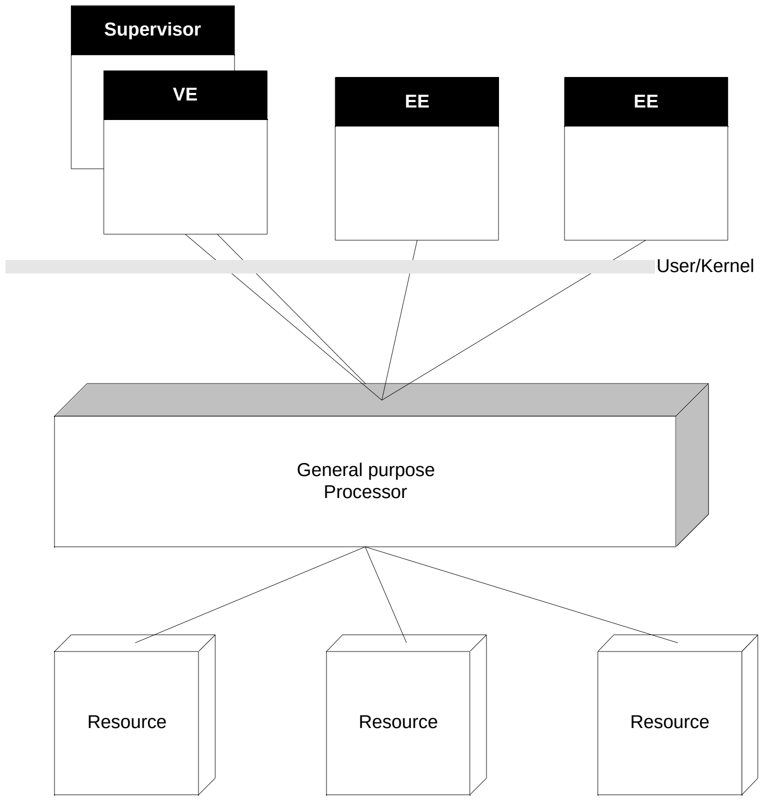


Figure 5-6 Generic Operating System with Packet Processing Applications

This sort of system forwards packets to an application running in user space, which is the execution environment. Such systems are very easy to manage. The application is easy to develop and, because it runs in user space, is already operating in a constrained environment. Performance is very poor and such systems are only effective in the control and management planes.

5.3.3 Summary

From this section, it should be clear that an active node can provide many services operated by a number of service providers in a competitive environment on a variety of technologies which need to be a very closely managed.

The next section looks in detail at providing engineering components that can provide facilities for:

- Resource Control
- Security
- Demultiplexing to forward traffic to the proper VE and executed by the proper EE

These are specifications for these facilities. After these two facilities have been described, an information model for the loading and activation of execution environments is given.

6 FAIN NODE ARCHITECTURE DESIGN

6.1 Resource Control Framework & Design

6.1.1 Introduction

In this document we present the design of the Resource Control Framework (RCF) that is going to be used in the FAIN Active Nodes as well as in the FAIN Active Network. However the emphasis of this work will be on the Active Node due to its complexity and importance. For the Active Network we will be based mostly on already available results from the research community for controlling networking resources. Resource control is quite important in such an open environment where the potential actors will struggle for taking their piece. It is apparent that these operations should be done as efficiently as possible. The resources are distinguished into two categories, namely Physical and Logical.

The resource control framework may apply throughout the four layers of the Active Nodes architecture namely Programmable Hardware, Node OS, Execution Environments and Active Applications.

6.1.1.1Motivation

The network that provides the dynamic deployment of the new services and gives users the capability to compute their data at the node is called as an active network. It seems to be very promising because the active network can introduce new services rapidly by dynamic deployment of function. Besides, it is possible to customise the network by data processing at each active node.

6.1.1.2Objectives

The main objective of Active Networking of FAIN is to provide the necessary infrastructure for the dynamic deployment of the new services and to give the user the ability to customise the network using his own code. This dynamic aspect of active networks has an impact on the requirements for resource management of active node. In comparison to a traditional network node, the active node offers the ability to dynamically inject code, which implements a new service. The code is injected in an execution environment that belongs to a VE, which presents an abstraction of the node to the active application. In order to be able to support these services the VE should have guaranteed access to the necessary resources of the node. In addition an active node may contain multiple VEs, which will have to compete for node resources. This requires the existence of a resource control framework in the node OS, which will be responsible for the management of the node resources and their allocation to the different VEs.

6.1.1.3Scope

The RCF will develop the mechanisms needed for the control of resources in the active node and export the corresponding interfaces to the resource consumers (VEs, active application) and to the network management system. The basic interfaces offered should support the allocation and monitoring of logical and physical resources. The VEs and applications will be interested in the allocation of the necessary resources, while a network management system will use the RCF interface to allocate resources using higher-level policies and to monitor the resource usage for accounting and performance management.

A service provider, who uses the node for network services, needs specific resources that are allocated to a Virtual Environment. The active node contains multiple VEs, which represent different abstractions of the node. Every VE could be assigned to a different service provider. The RCF will partition the resources of the node to the VEs after agreement between the service providers and the node operator or by a dynamic way, which will depend on the resource needs of VEs. The partition of the node resources should guarantee to VEs access to node and network resources that VEs need, or the node operator has agreed to provide to the corresponded service providers.

6.1.2 Definition of Resources

A resource of an Active Node is called the certain capability of the node (processing, data storage, forwarding, object creation etc.) that is offered at a certain amount to a specific entity, which we call Active Node resource consumer.

We distinguish two types of resources namely physical and logical. Physical resources refer to the hardware capabilities of the Active Node. Logical resources refer to the software capabilities of the Active Node for the creation of objects that handle the various data flows inside the node. Software resources depend of course on hardware resources but they depend also on the quality of the design that took place for their creation.

6.1.2.1Physical Resources (PR)

In Table 6-1 we list the identified Active Node's physical resources.

Table 6-1: List of Physical Resources

Name of Resource	Explanation
CPU time	It is the time required for the execution of an application, a process, a thread etc.
Data Bus Bandwidth	Refers to the capability for data transmission in the data bus being used for data exchanges among the various physical components of the Active Node.
Input Bandwidth of Network	Refers to the maximum rate at which the Active Node can handle incoming data from the network interface. It is controlled on the basis of an input bandwidth table.
Input Network Buffers	They are composed of hardware registers that are used for queuing the incoming packets before their transmission to the processing or forwarding engines of the Active Node
Memory	Memory is the set of physical devices used for the short-term storage of data like RAMs.
Output Bandwidth of Network	Refers to the maximum rate at which the Active Node can transmit outgoing data to the network interface. It is controlled on the basis of an output bandwidth table.
Output Network Buffers	They are composed of hardware registers that are used for queuing the outgoing packets after they have been handled from the processing or the forwarding engines of the Active Node.
Storage	Storage refers to the physical devices being used for storing data for long periods of time like a hard disk. Access to storage devices is usually slower than the access to memory.

6.1.2.2Logical Resource (LR)

Table 6-2 shows the list of logical resources.

Table 6-2: List of Logical Resources

Name of Resource	Explanation
Queue	It is a software component that is used for the queuing of packet data that are exchanged among the components of the Active Node.
Classifier Table	It is a software component which contains the information about identifying which data flow is assigned to which queue.
Computation Table	It is a software component which contains the information about which processing is executed to the packets per flow.
Filtering Table	It is a software component, which contains the information about which packets will be accepted for further processing within the node or discarded for each incoming data flow. It has also the ability to distinguish the active from the non-active packets. Active packets will be sent for further processing within the node whereas non-active packets will be forwarded (or routed) to the proper output port.
Forwarding Table (or Routing Table)	It is a software component which contains the information about which modifications should take place on the addresses that are carried on each non-active and active (after it has been processed or locally generated) packet and to which output port it has to be transmitted.
Thread	Thread is a logical resource of independent execution.

6.1.3 Requirements

6.1.3.1General Requirements

After the definition of resources that has took place in the previous section we should now try to answer the following question. *Who, when and how is allowed to consume node resources?* We have first to define who are the *consumers* of each resource and then to elaborate on the mechanisms that are required in order to answer the above question.

Consumers of resources

As it was described above, there are two kinds of resources:

· the physical resources, which each of them corresponds to a specific physical devise of the node (e.g. Storage corresponds to the hard discs)

· the logical resources, which each of them corresponds to a set of physical resources and occasionally to a set of logical resources, implemented in a programming way (software resources).

The consumers of the resources could be the VEs and the logical resources of the node as well. The logical resources are consumers of the physical resources of the node and when it is necessary, of some other logical resources. So the resources consumption chain is described in Figure 6-1:

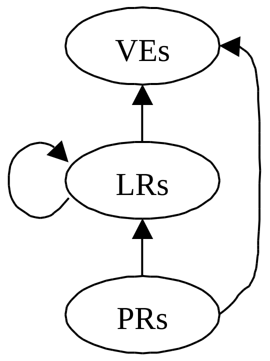


Figure 6-1: Resource Consumption Chain

Requirements for resource control

Major Requirement 1: Control the consumption of resources

· **Access Control**

Request for consumption of the resource X (**who** is allowed to consume resource X?) should be authenticated and authorised by security subsystem.

· **Admission Control**

Admission Control mechanisms for requesting and reserving resource X (**how** a consumer A requests reservation of resource X)

· **Isolation of resource consumption between the various consumers**

Mechanisms for resource consumption isolation (resource usage of resource X by a consumer A should not interfere with the use of the same resource by a consumer B)

· **Scheduling**

Mechanisms for dedicating portions of the total capacity of a resource to consumers (priorities, round robin, fair share etc.) should be selected carefully (**how** a consumer is guaranteed his portion of the assigned capacity of a resource).

Major Requirement 2: Monitor the consumption of resources

In order to control each resource we have first to monitor it. Thus there is a need for each resource to be associated with a monitoring facility that monitors resources like CPU, memory etc.

6.1.3.2Requirements from Security Block

RCF has a limited scope and, thus, requires certain services from other subsystems, which collectively compose FAIN ANN architecture. The same applies the other way round, i.e. other FAIN ANN subsystems delegate certain functionality to RCF, since RCF is most suitable for providing this functionality. Here we specify only two specific sets of requirements, those posed by RCF to security subsystem and vice-versa.

6.1.3.2.1 RCF requirements towards Security subsystem

6.1.3.2.1.1 Perform Request Authorisation

There is no need for RCF to perform RCF specific authorisation decisions, since a common authorisation engine can be used to make authorisation decisions on behalf of every subsystem in an ANN. The advantage of this is that there is no duplication of work within different ANN subsystems and RCF becomes simpler.

In order to meet this RCF requirement, security subsystem provides an authorisation service, which is accessible via its interface. In other words, RCF forwards every request to security subsystem by invoking its authorisation() method. Security subsystem returns either "AUTHORISED" or "NOT AUTHORISED" to RCF.

6.1.3.2.1.2 Log RCF related events

Keeping a log of events in the ANN is important for auditing, but also for accounting. Logging provides a chronological trace of events, which can be later, analysed or processed, for different purposes.

In order to meet this RCF requirement, security subsystem provides a logging service, which is accessible via its interface. Upon every critical event RCF adds a new log entry that describes the event.

6.1.3.2.2 Security requirements for RCF

6.1.3.2.2.1 Monitor current consumption of resources per-entity

In many cases an entity is requesting *additional* resources, i.e. it is already using a specific amount of node resources. One of the factors that influence the authorisation decision is the amount of resources that this entity is currently occupying. RCF is tightly coupled with node resources, so it is the natural place for resource usage monitor, which maintains an updated table of consumed resources vs. entities.

In order to meet this requirement, RCF provides a monitoring service, which is accessible via its interface. Whenever security subsystem needs information on current resource consumption for particular entity, it will be requested from RCF.

6.1.3.2.2.2 Enforce authorisation decision

Authorisation engine only *decides*, whether a request should be allowed or not. Thus, security subsystem depends on RCF to enforce this decision. If, for example, security subsystem returns a negative decision, such as ``request [to allocate X bytes of memory to entity Y] not authorised," then RCF must reject the request altogether. On the other hand, even if security subsystem returns a positive decision, such as ``request [to allocate K bytes of memory to entity L] authorised," RCF must prevent entity L from using more than K bytes of memory.

Thus, in order to meet this requirement, RCF must provide policy enforcement mechanisms at two levels. At the ``higher level" enforcement is a simple logic, which asks external security system for authorisation and simply discards the request in case it is unauthorised. At the ``lower level" enforcement is embodied in a more complex policing algorithm(s), which can control the scheduler(s) for specific resource and thus impose limits on resource usage by an entity. For example, ATM policing mechanisms fall into this category.

6.1.3.3Requirements from Management Block

6.1.3.3.1 Fault Management

The RCF must support the PBNM in order to develop features (flexibility, extensibility) and combine them with the active node capabilities in order to provide an autonomous fault management system, which favours a quick solution to the problems, avoiding the failure to progress in its importance.

RCF components should be able to inject events asynchronously in the event detection layer. Through these messages could be implemented an appropriate mechanisms for fault notification and local decision dissemination. In this way it is possible to modify the global policies that could be affected as a consequence of the local node anomalous behaviour.

Furthermore, the RCF must provide the facility for the element manager in order to access the MIB variables.

6.1.3.3.2 Accounting

The RCF through Monitoring could inform the management system about the resource usage per service in order to be possible to do the accounting. The RCF should monitor the consumption of node resources, in order to provide the necessary accounting information. The monitoring can be done either per-VE or per-EE or per-application.

6.1.3.3.3 Configuration Management

The configuration management functionality implemented in the Management system will be basically that offered by RCF as it controls and offers the resources of the managed node. When we say resources we mean both the communication and the computational as well as physical and logical.

Using the PBNM system we want to assure that certain flows get the necessary quality of service, by allocating sufficient bandwidth and using priorities for the queues. For this reason the PBNM system will have to support the necessary mechanisms for QoS provision (IntServ, DiffServ, RSVP). This means that the PBNM system must be able to identify different flows based on certain fields of the packet header (source/destination IP addresses, protocol, source/destination ports) and make the required resource reservations.

The computational resources of the node are the processing power (CPU cycles), the memory and the disk storage. An active node may have multiple Execution Environments residing in a VE. It is the VE that is competing for the resources of the node with other VEs and therefore the administrator must set up the corresponding policies to partition the computational resources of the node as needed. There is also a need to assign packet flows to VEs. The PBNM system is also responsible for the creation, deletion and modification of VEs. When the creation of a new VE is requested, the PBNM system should check existing policies, to see if the person that made the request has the necessary privileges and if there are sufficient resources in the node for this VE. There should also be priorities for different VEs, to ensure that the most critical VEs always have enough resources to execute. These priorities also depend on Service Level Agreements made between the Network Provider and the Consumers.

To be able to provide all the above services the PBNM system must use the interface offered by the RCF.

First of all, the RCF must identify packet flows according to packet fields. The incoming packets will pass through a filter. This filter will identify for each packet the flow that belongs to and it will process it to the appropriate queue.

Secondly the RCF must Reserve (and free) bandwidth resources to flows in order to limit the congestions both to the node and to the network layer. Every flow will have access to a specific upper limit of bandwidth, which is computed to satisfy its needs.

Thirdly it will allow access to the computational and forwarding resources to VEs. This depends from the level in which the RCF will do the allocation of the resources. The central node RCF will allocate resource per VE basis. Every VE could contain multiple EEs. So the resource control sub-component inside a VE will be responsible to allocate resources to the corresponding EEs. Furthermore it will assign flows to EEs for a better processing procedure.

Fourthly it will determine the node OS functionality a VE is able to access since every VE will have specific privileges to the resources. These privileges will be assigned to the VEs from the RCF after the verification of the Authentication Engine. It will have the ability to install and remove code within a VE in order to satisfy new requirements by the application running to the multiple EEs. The multiple VEs will have the same management interface something that extents the simplicity of the system.

6.1.3.3.4 Performance Management

The RCF must collect from the active node all the information regarding the status and the resources of the node. This information includes computing resources status, in general and per flow, like CPU use percentage, used memory, free memory, free hard disk, number of threads in the node, number of threads per execution environment, etc. Information about node forwarding resources status, on per interface and per flow basis, like bandwidth used and free bandwidth, should also be provided. This information will be the base in which the PBANEM system performs the management functionality.

6.1.4 Resource Control Framework Architecture

In this chapter it described the RCF element-based architecture.

6.1.4.1Architecture

The Figure 4-1 shows the element-based architecture of the RCF.

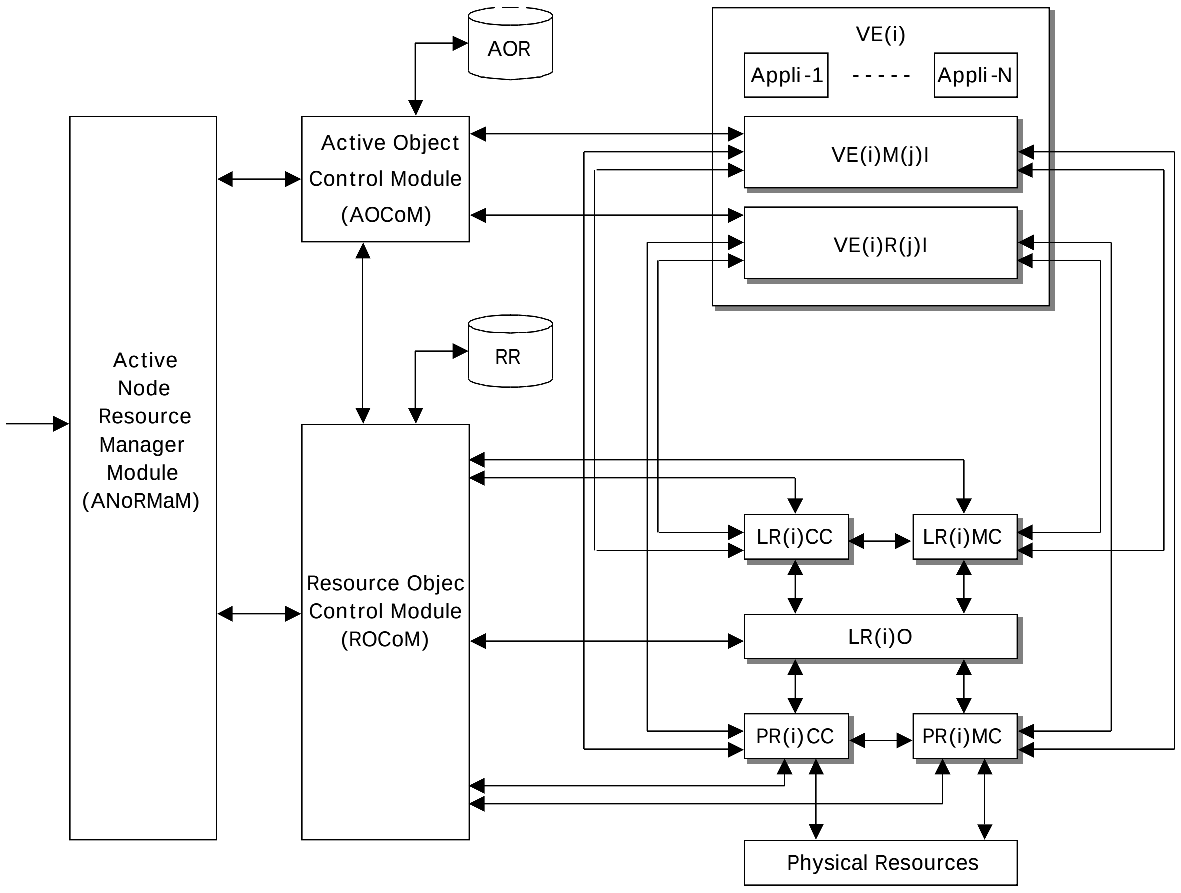

AOR
VE(i)
Appli-1
- - - - -
Appli-N
Active Object Control Module (AOCoM)
VE(i)M(j)I
VE(i)R(j)I
RR
Active Node Resource Manager Module (ANoRMaM)
LR(i)CC
LR(i)MC
Resource Objec Control Module (ROCoM)
LR(i)O
PR(i)CC
PR(i)MC
Physical Resources

Figure 6-2: Resource Control Module Architecture

6.1.4.2Architectural Elements

6.1.4.2.1 Active Node Resource Manager Module (ANoRMaM)

Being in close collaboration with the Resource Objects Control Module and the Active Objects Control Module, it manipulates all the external requests, and undertakes the communication between the RCF Block and the other Blocks of the AN (e.g. the management and the security block). It decides which action should take place in order to satisfy an external request, and replies to the requested entity for the result.

6.1.4.2.2 Active Object Control Module (AOCoM)

The role of this component is the instantiation of the resource control mechanisms in every VE and the handling of these objects' instances. It works in close collaboration with the AOR.

This module is responsible for:

- the creation of the VEs
- the instantiation of the active objects in the VEs (Resources Instances and Resource control mechanisms instances.)
- the control of the active objects. (e.g. when a request for allocating more resources to an instance, has been approved as valid, it is configuring this instance in order to be able to use the allocated resource.)
- keeping up to date the Active Objects Repository
- retrieving all the information that have been stored in the AOR

6.1.4.2.3 Active Object Repository (AOR)

The AOR contains all the details of the resources and VE's RCF mechanism instances, which are used by the VEs.

It keeps information about:

- the VE's IDs
- the Active Objects IDs
- the amount of the node resources, which are allocated to each object
- the VE in which each Object belongs to

6.1.4.2.4 Resource Object Control Module (ROCoM)

This module is responsible for the instantiation of the Logical Resources Objects' Instances and it is in close collaboration with the RR. In more detail this module is responsible for:

- the initialisation of the control and monitoring components of the Physical resources of the node
- the initialisation of the Logical Resources Objects
- the initialisation of the control and monitoring components for the Logical resources
- the control of all these elements, e.g. it manipulates all the requests for extra allocation of a resource to an existing consumer or for allocation of an amount of a resource to a new consumer. It enforces these actions by configuring the control components.
- keeping up to date the Resource Repository
- retrieving all the information that has been stored in the RR

6.1.4.2.5 Resource Repository (RR)

It contains all the details of the instances of the resources, which are internal to the RCF block.

It keeps information about:

- the total amount of each resource (physical or logical)
- the total consumed amount of each resource
- the total free amount of each resource
- the VE’s IDs
- the amount of allocated resources per VE.

6.1.4.2.6 Logical Resources Objects

Every object of this kind represents a specific Logical Resource of the AN. It contains the corresponding Instances of this resource in the VEs, and every part of its capacity is allocated to a specific VE’s Instance. These Objects are consumers of the Physical Resources, and the consumption is controlled by accessing the corresponding Control and Monitoring Modules. The available quantity of each resource is changing dynamically by allocating more physical resources or by releasing some of the already allocated.

6.1.4.2.6.1 Queue

A queue is an object that is created whenever a requirement for the queuing of data between two software components of the RCF exists. The number of the queue objects and their size are decided during the configuration of the corresponding communicating components and according to their needs (i.e. dynamically).

6.1.4.2.6.2 Classifier Table

It is a classifier table to be used by every consumer. It is a software component which contains the information about identifying which data flow is assigned to which queue. The size of the classifier table is set during configuration.

6.1.4.2.6.3 Computation Table

It is a computation table to be used by every consumer. It is a software component which contains the information about which processing is executed to the packets per flow. The size of the computation table is set during configuration.

6.1.4.2.6.4 Filtering Table

It is a filtering table to be used by every consumer. It is a software component, which contains the information about which packets will be accepted for further processing within the node or discarded for each incoming data flow. It has also the ability to distinguish the active from the non-active packets. Active packets will be sent for further processing within the node whereas non active packets will be forwarded (or routed) to the proper output port. The size of the filtering table is set during configuration.

6.1.4.2.6.5 Forwarding or Routing Table

It is a forwarding table object for being used by every consumer. It is a software component which contains the information about which modifications should take place on the addresses that are carried on each non-active and active (after it has been processed or locally generated) packet and to which output port it has to be transmitted. The size of the forwarding table is set during configuration.

6.1.4.2.6.6 Threads

Threads are available to be used by every consumer. Their mission is to ensure the independent execution of each of the Active Node’s software components. Each component takes a certain number of threads during execution. The details of the threads allocation per component are set during configuration.

6.1.4.2.7 Logical Resources Control Components

One component of this kind exists for each LR. Every one of them is scheduling the requests for accessing the corresponding Logical Resource, and allocates a specific amount to every consumer.

6.1.4.2.7.1 Queue Control Component

This component is responsible for controlling the amount of packet data in a queue. For example, if the amount of packet data in the queue exceeds a certain limit that has been previously configured, some of the packet data will be discarded and a request for queue length reconfiguration may take place.

6.1.4.2.7.2 Classifier Table Control Component

This component is responsible for controlling the data entries in the classifier table (i.e. initialise, write, delete, modify).

6.1.4.2.7.3 Computation Table Control Component

This component is responsible for controlling the data entries in the computation table (i.e. initialise, write, delete, modify). For example, it writes the data for identifying the data flow and for identifying which processing is executed to the flow. Besides it removes the data in the computation table.

6.1.4.2.7.4 Filtering Table Control Component

This component is responsible for controlling the data entries in the filtering table (i.e. initialise, write, delete, modify).

6.1.4.2.7.5 Forwarding or Routing Table Control Component

This component is responsible for controlling the data entries in the forwarding table (i.e. initialise, write, delete, modify).

6.1.4.2.7.6 Threads Control Component

This component is responsible for controlling the threads. For example, it controls the allocation of threads per consumer.

6.1.4.2.8 Logical Resources Monitoring Components

One component of this kind exists for each LR. Every one of them monitors the usage of the corresponding Logical Resource per consumer.

6.1.4.2.8.1 Queue Monitoring Component

This component is responsible for monitoring the amount of data in a queue. For example, if the amount of data in the queue exceeds the limit that is configured, a callback is issued.

6.1.4.2.8.2 Classifier Table Monitoring Component

This component is responsible for monitoring the data of classifier table. For example, it allows retrieving statistics for particular classes.

6.1.4.2.8.3 Computation Table Monitoring Component

This component is responsible for monitoring the data of a computation table. For example, it allows retrieving statistics for the computation entities.

6.1.4.2.8.4 Filtering Table Monitoring Component

This component is responsible for monitoring the data of a filtering table. For example, it allows retrieving statistics for the filters.

6.1.4.2.8.5 Forwarding or Routing Table Monitoring Component

This component is responsible for monitoring the data of forwarding table. For example, it allows retrieving statistics for the forwarding entries.

6.1.4.2.8.6 Thread Monitoring Component

This component is responsible for controlling the thread. For example, it allows to retrieves statistics for threads.

6.1.4.2.9 Physical Resources Control Components

One component of this kind exists for each Physical Resource. It is scheduling the requests for accessing the corresponding Physical Resource, and allocates a specific amount of the resource for every consumer.

6.1.4.2.9.1 CPU Control Component

This component is responsible for controlling the CPU time based on the CPU time schedule table. The schedule table contains the information between the CPU time and the consumer of it. The CPU control component assigns the CPU time to each consumer of it. For example, the CPU time is allocated in each thread or VE. Besides, it enforces the real CPU time to match the CPU time that is stated in the CPU time schedule table.

6.1.4.2.9.2 Data Bus Control Component

This component is responsible for controlling the data bus bandwidth based on the data bus bandwidth schedule table. The schedule table contains the information between the data bus bandwidth and the consumer of it. The data bus control component assigns the data bus bandwidth to each consumer of it. For example, the data bus bandwidth is allocated in each thread or VE. Besides, it enforces the real data bus bandwidth to match the data bus bandwidth that is stated in the data bus bandwidth schedule table.

6.1.4.2.9.3 Input Bandwidth Control Component

This component is responsible for controlling the input bandwidth based on the input bandwidth table. The table contains the information between the input bandwidth and the consumer of it. The input bandwidth control component assigns the input bandwidth to each consumer of it. For example, the input bandwidth is allocated in each flow or VE. Besides, it enforces the real input bandwidth to match the input bandwidth that is stated in the input bandwidth table.

6.1.4.2.9.4 Memory Control Component

This component is responsible for controlling memory based on the memory table. The table contains the information between the size of memory and the consumer of it. The memory control component assigns the memory to each consumer of it. For example, the memory is allocated in each thread or VE. Besides, it enforces the real memory size to match the memory size that is stated in the memory table.

6.1.4.2.9.5 Output Bandwidth Control Component

This component is responsible for controlling the output bandwidth based on the output bandwidth table. The table contains the information between the output bandwidth and the consumer of it. The output bandwidth control component assigns the output bandwidth to each consumer of it. For example, the output bandwidth is allocated in each flow or VE. Besides, it enforces the real output bandwidth to match the output bandwidth that is stated in the output bandwidth table.

6.1.4.2.9.6 Storage Control Component

This component is responsible for controlling storage based on the storage table. The table contains the information between the size of storage and the consumer of it. The storage control component assigns the storage to each consumer of it. For example, the storage is allocated in each thread or VE. Besides, it enforces the real storage size to match the storage size that is stated in the storage table.

6.1.4.2.10 Physical Resources Monitoring Components

Every component of this kind exists for each PR. It is monitoring the usage of the corresponding Physical Resource per consumer.

6.1.4.2.10.1 CPU Control Monitoring

This component is responsible for monitoring the CPU time usage. It allows to retrieve statistics for the CPU usage and to invoke callbacks when high or low watermarks are reached or quotas are violated.

6.1.4.2.10.2 Data Bus Monitoring Component

This component is responsible for monitoring the data bus bandwidth usage. It allows to retrieve statistics for the bus bandwidth usage and to invoke calbacks when high or low watermarks are reached or quotas are violated.

6.1.4.2.10.3 Input Bandwidth Monitoring Component

This component is responsible for monitoring the input bandwidth usage. It allows to retrieve statistics for the input bandwidth usage and to invoke calbacks when high or low watermarks are reached or quotas are violated.

6.1.4.2.10.4 Memory Monitoring Component

This component is responsible for monitoring memory usage. It allows to retrieve statistics for the memory usage and to invoke calbacks when high or low watermarks are reached or quotas are violated.

6.1.4.2.10.5 Output Bandwidth Monitoring Component

This component is responsible for monitoring the output bandwidth usage. It allows to retrieve statistics for the output bandwidth usage and to invoke calbacks when high or low watermarks are reached or quotas are violated.

6.1.4.2.10.6 Storage Monitoring Component

This component is responsible for monitoring storage usage. It allows to retrieve statistics for the storage usage and to invoke callbacks when high or low watermarks are reached or quotas are violated.

6.1.4.2.11 VE's Resources Instances

The Resources are allocated by the instantiation and the proper configuration of the instances of these resources, by the AOCC. Every instance represents a Physical or a Logical Resource in the VE and it is the consumer of a specific amount of the corresponding resource of the Node. The consumption is taking place through the corresponding control module of the Node's RCF. The capacity of the instance is shared to the different Applications, which are running in the VE.

The VE's creation request includes the set of resources, which are needed for the new VE. The AOCoM, which is the responsible module for the creation of every VE, initialises a resource instance for each resource and configures it in order to have access to a specific amount of the corresponding node resource. During the lifecycle of each VE, when it is needed for the allocated amount of a resource to be changed, the AOCoM has only to re-configure the proper resource instance.

The set of the instances of the resources in a VE constitutes in a way a virtual node. The active applications use these virtual resources in order to be realised (implemented), but actually they use the allocated actual resources of the node for that purpose. Every access of a Resource Instance within a VE is transformed to an access to the corresponding node resource. For example, when an application wants to make a new entry in the routing table makes a new entry to the virtual routing table of the VE by accessing the Routing table Instance of the VE. The Routing Table Instance forwards this entry to the Routing Table of the node, in the part of it, which is allocated to that Instance. The Figure 4-1 shows the way that the consumption of the node resources achieved by using the instance of the resources.

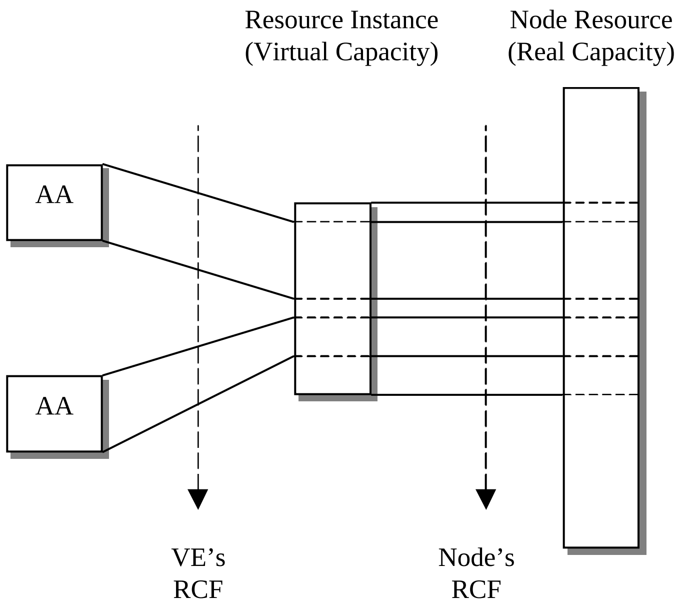


Figure 6-3: Resourse consumption by using the instance of the resource

6.1.4.2.12 VE’s RCF Mechanism Instances

Every Object of this category implements a specific Module of the VE’s Resource Control Framework, for example the Control Module of the Routing Table Instance of the VE. The set of all the VE’s RCF Mechanisms Instances constitutes the VE’s RCF. The AOCoM initialises these instances during the creation of the VE. Which mechanism is going to be used in order to control a specific resource in the VE is a choice of the owner of the VE (e.g. the service provider) or of a higher-level functional block (e.g. Management) and included in the VE creation request. The RCF should give the opportunity to the entity, which is doing the request to choose among a set of mechanisms for each resource. For example to schedule the CPU usage it can be chosen to have a simple FIFO queue or a multilevel queue with different priorities to each level.

These modules need resources to operate, so they consume the resources of the node by accessing the corresponding Control Modules. During the initialisation of these instances, the node RCF allocates the set the node resources, which are needed in order for the instances to be able to function.

6.1.5 Design of Modules

Details of the design of the previously described RCF basic components are described in this chapter.

6.1.5.1Design of ANoRMaM

The ANoRMaM has as main mission the handling of all the external requests and the communication between the RCF block and the other blocks (e.g. Security, Management). For the handling of the external requests, the ANoRMaM is in close collaboration with the AOCoM and the ROCoM. Thus there exist the following Interfaces:

A. INTERNAL I/Fs

i. Int I/F 1: Between ANoRMaM and AOCoM

Functions of Int I/F 1	Description
VE_creation_result(result,VE_ID)	Called after the creation attempt. Gives the result (SUCCESS/ FAILURE) and in the success case the ID of the new VE.
removal_ confirmation(VE_ID)	Called after the removal and confirming the action.
VE_parameters(VE_ID, parameters)	Gives the parameters of the VE (set of allocated resources, etc.)

ii. Int I/F 2: Between ANoRMaM and ROCoM

Functions of Int I/F 2	Description
Export_status_of_resource(resource,status)	Export the current status of a specific resource (total capacity, allocated capacity, consumed capacity)
Export_Consumers_of_resource(kind of resourse)	Export the list of consumers of a specific resource (VE_ids, allocated capacity to every consumer, consumed capacity from every consumer)

6.1.5.2Design of AOCoM

The AOCoM is responsible for the VEs and the Active Objects in them and it is in close collaboration with the AOR for that purpose. It also collaborates with the ANoRMaM basically to reply to their requests, and with the ROCoM. The AOCoM doesn't communicate with entities from other blocks, so it doesn't have any external I/Fs. The following internal I/F exists for the AOCoM:

A. INTERNAL I/Fs

i. I/F 3: Between AOCoM and VE's Resource Instances

Functions of Int I/F 3	Description
Configure(capacity, node resource control component ID)	Called during the initialisation process of the VE, or during the reconfiguration process of the VE. Gives to the instance the capacity which is allocated to it, and the ID of the component that is responsible for this resource
remove()	Remove the instance

ii. I/F 4: Between AOCoM and VE's Mechanisms Instances

Functions of Int I/F 4	Description
Configure(set or resources capacity, node resource control components ID, other configuration parameters)	Called during the initialisation process of the VE, or during the reconfiguration process of the VE. Gives to the instance the capacity which is allocated to it, and the ID of the component that is responsible for this resource
Remove()	Remove the instance

iii. I/F 5: Between AOCoM and ANoRMaM

Functions of Int I/F 1	Description
create_New_VE(set of resources, set of mechanisms)	Request for creation of a new VE
remove_VE(VE_ID)	Request for removing of an existent VE
Reconfigure_VE(VE_ID, new parameters)	Request for reconfiguration of an existent VE
Export_VE_parameters(VE_ID)	Asking for the parameters of a specific VE

iv. I/F 6: Between AOCoM and ROCoM

Functions of Int I/F 6	Description
confirm_allocation_resources (VE_ID, capacity of resources, instances IDs)	Confirmation of the allocation
confirm_releasing_resources (VE_ID, instances IDs)	Confirmation of the releasing
confirm_reallocation _resources (VE_ID, new capacity, instances IDs)	Confirmation of the reallocation

v. I/F 7: Between AOCoM and AOR

Functions of Int I/F 7	Description
query_results (results)	AOR export the results of a specific query

6.1.5.3Design of ROCoM

The ROCoM is responsible for the Control of the Logical Resources and the Control and the Monitoring Components of the Physical and the Logical Resources of the node. For that purpose is in close collaboration with the RR. It also collaborates with the ANoRMaM basically to reply to their requests, and with the AOCoM. The following internal I/F exists for the ROCoM:

A. INTERNAL I/Fs

i. I/F 8: Between ROCoM and Logical Resource Objects

Functions of Int I/F 8	Description
configure (configuration parameters)	Configure or reconfigure the logical resource object

ii. I/F 9: Between ROCoM and Logical Resources Control Components

Functions of Int I/F 9	Description
control_configuration (configuration parameters)	Configure or reconfigure the logical resource control component

iii. I/F 10: Between ROCoM and Logical Resources Monitoring Components

Functions of Int I/F 10	Description
total_consumption(total consumed capacity)	Gives the total consumption of the resource
consumer_consumption(consumer ID, consumed capacity)	Gives the consumption of the resource by a specific consumer

iv. I/F 11: Between ROCoM and Physical Resources Control Components

Functions of Int I/F 11	Description
confirm_allocation(instance ID, capacity)	Confirmation of the allocation
control_configuration (configuration parameters)	Configure or reconfigure the physical resource control component

v. I/F 12: Between ROCoM and Physical Resources Monitoring Components

Functions of Int I/F 10	Description
total_consumption(total consumed capacity)	Gives the total consumption of the resource
consumer_consumption(consumer ID,	Gives the consumption of the resource by a specific

consumed capacity)	consumer

vi. I/F 13: Between ROCoM and RR

Functions of Int I/F 7	Description
query_results (results)	RR export the results of a specific query

vii. I/F 14: Between ROCoM and ANoRMaM

Functions of Int I/F 14	Description
status_of_resource(kind of resource)	Asking for the current status of a specific resource (total capacity, allocated capacity, consumed capacity)
Consumers_of_resource(kind of resourse)	Asking for the list of consumers of a specific resource (VE_ids, allocated capacity to every consumer, consumed capacity from every consumer)
block_VE(VE_id)	Block the any request of consumption from the VE, without release the resources which have been allocated to it.
Unblock_VE(VE_id)	Unblock the blocked VE
Confirm_VE_privilege(VE_id, set of privilages)	Confirmation of the privileges of the VE (set of resources that have been authorised to be allocated). This function is the reply of a confirmation request from the ROCoM.

viii. I/F 15: Between ROCoM and AOCoM

Functions of Int I/F 6	Description
allocate_resources (VE_ID, capacity of resources, instances IDs)	Request for allocation of resources to the VE's instances
release_resources (VE_ID, instances IDs)	Request for releasing of all the resources that have been allocated to a specific VE.
reallocate_resources(VE_ID, new capacity, instances IDs)	Request for reallocation of all the resources that have been allocated to a specific VE.

6.1.5.4Design of LRs Control Components

The design of each logical resource's control component is described in this section.

6.1.5.4.1 Design of Queue Control Component

Figure 6-4 shows the model of queue control. This queue control component controls the queue not to be overflow. At first, the queue controller receives the borderline for controlling the queue from the outside component and stores it to the queue control table that is shown in the Table 6-3. Then the queue monitor observes the amount of data in the queue. When the amount of data in the queue exceeds the borderline that is stored in the queue control table, the queue monitor informs the queue controller exceeding the borderline in the queue. After receiving the information about exceeding the borderline, the queue controller requests the input controller to discard the input data. When the amount of data in the queue becomes under the, the queue monitor informs the queue controller being under the borderline in the queue. After receiving the information about being under the borderline, the queue controller requests the input controller not to discard the input data.

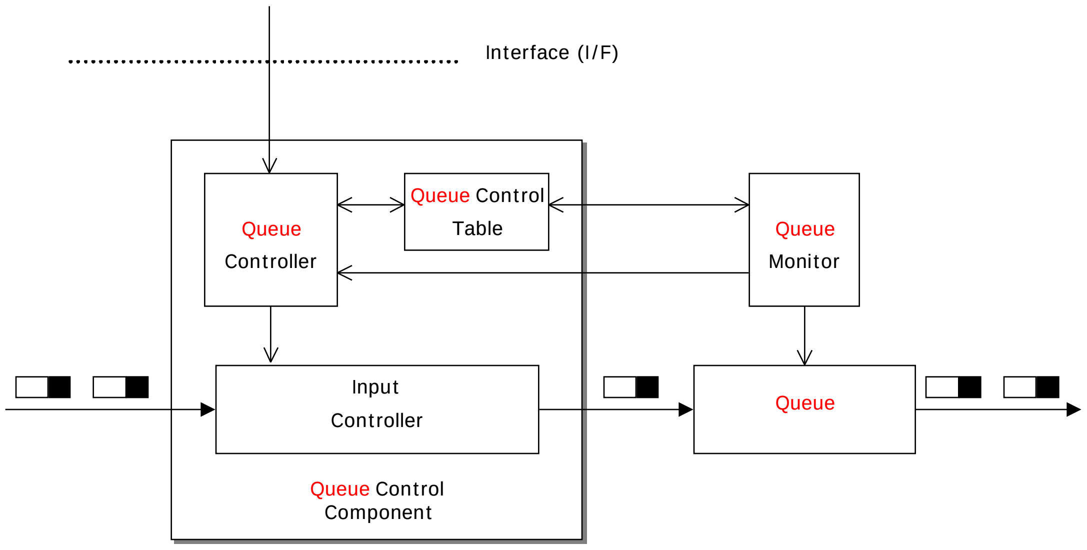


Figure 6-4: Queue Control Model

Table 6-3: Example of Queue Control Table

No of Queue	Control Point (Borderline)
B-1	Half Full
B-2	Half Full
- - -	- - -
B-N	Half Full

6.1.5.4.2 Design of Classifier Table Control Component

Figure 6-5 shows the mode of packet classifying. This classifier table control component controls the data of classifier table that is shown in the Table 6-4. In this model, the flow is identified by SIP, DIP, Port of SIP, Port of DIP and Protocol No etc. At first, the classifier table control component receives the data for classifying the flow from outside component and stores it to the classifier table. Then the packet classifier retransmits the received packets to the proper queue based on the classifier table.

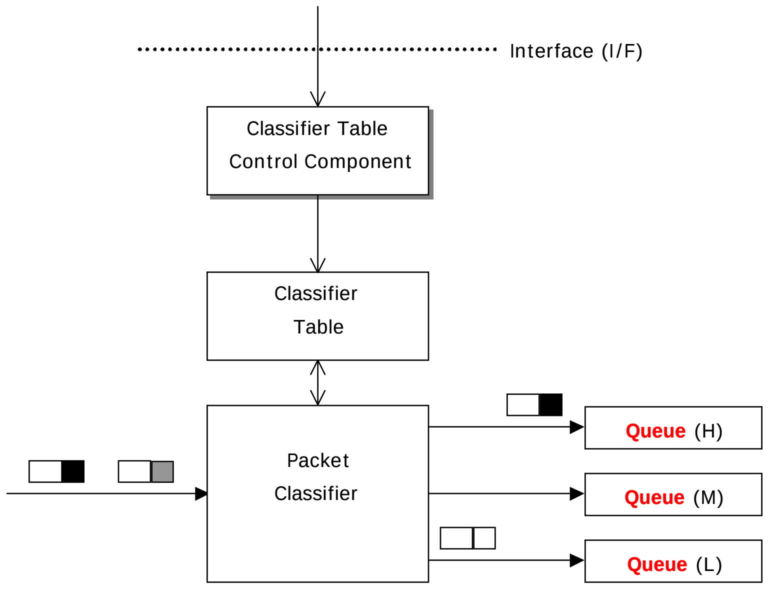


Figure 6-5: Data Classifying Model

Table 6-4: Example of Classifier Table

Flow						Destination
SIP	**DIP**	**Port of SIP**	**Port of DIP**	**Protocol No**	**etc.**	**of Queue**
S-1	D-1	PoS-1	PoD-1	No-1		**Queue** (High)
S-2	D-2	PoS-2	PoD-2	No-2		**Queue** (Low)
- - -	- - -	- - -	- - -	- - -		- - -
S-N	D-N	PoS-N	PoD-N	No-N		**Queue** (Middle)

6.1.5.4.3 Design of Computation Table Control Component

Figure 6-6 shows the mode of packet processing. This computation table control component controls the data of computation table that is shown in the Table 6-5. At first, the computation table control component receives the data for which processing is assigned to the flow from outside component and store it to the computation table. Then the dispatcher retransmits the received packets to the proper processing based on the computation table.

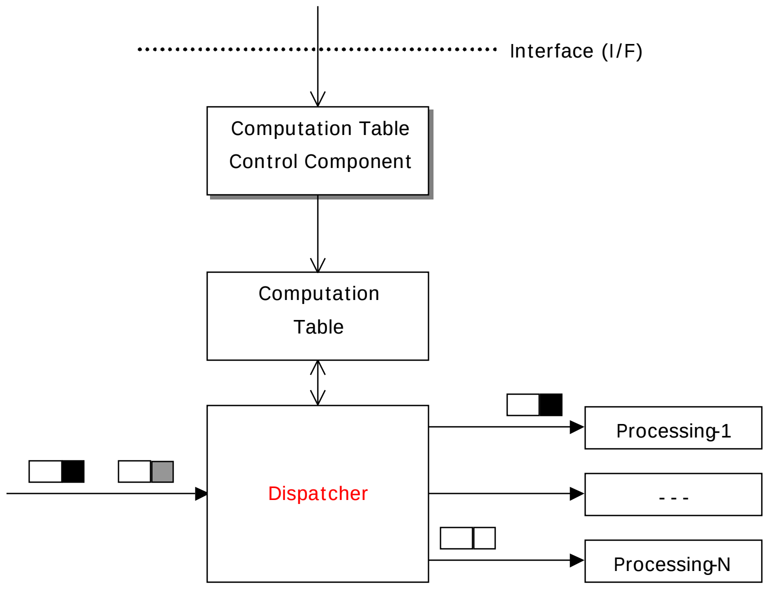


Figure 6-6: Packet Processing Model

Table 6-5: Example of Computation Table

Flow						Destination of Processing
SIP	**DIP**	**Port of SIP**	**Port of DIP**	**Protocol No**	**etc.**	
S-1	D-1	PoS-1	PoD-1	No-1		Processing-1
S-2	D-2	PoS-2	PoD-2	No-2		Processing-2
- - -	- - -	- - -	- - -	- - -		- - -
S-N	D-N	PoS-N	PoD-N	No-N		Processing-N

6.1.5.4.4 Design of Filtering Table Control Component

Figure 6-7 shows the mode of data filtering. This filter table control component controls the data of filter table that is shown in the Table 6-6. At first, the filter table control component receives the data for filtering the flow from outside component and stores it to the filter table. Then the packet filter decides to retransmit or not the received packets based on the filter table.

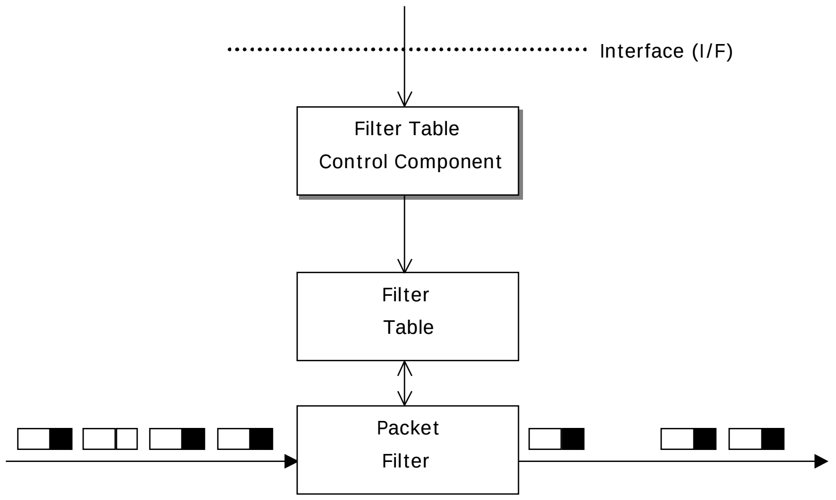


Figure 6-7: Data Filtering Model

Table 6-6: Example of Filter Table

Flow						Action
SIP	**DIP**	**Port of SIP**	**Port of DIP**	**Protocol No**	**etc.**	
S-1	D-1	PoS-1	PoD-1	No-1		Receive
S-2	D-2	PoS-2	PoD-2	No-2		Discard
- - -	- - -	- - -	- - -	- - -		- - -
S-N	D-N	PoS-N	PoD-N	No-N		Receive

6.1.5.4.5 Design of Forwarding Table Control Component

Figure 6-8 shows the design of the data forwarding component. Forwarding takes place on the basis of a forward table an example of which is shown in the Table 6-7. The incoming active and non-active packets are received by the forwarding engine. Active packets are directed to their specific Execution Environment for processing whereas non active packets are forwarded to the appropriate output port as indicated by their source-destination address pair and the corresponding entry in the forwarding table for transmission either to a local host or to a next node.

.

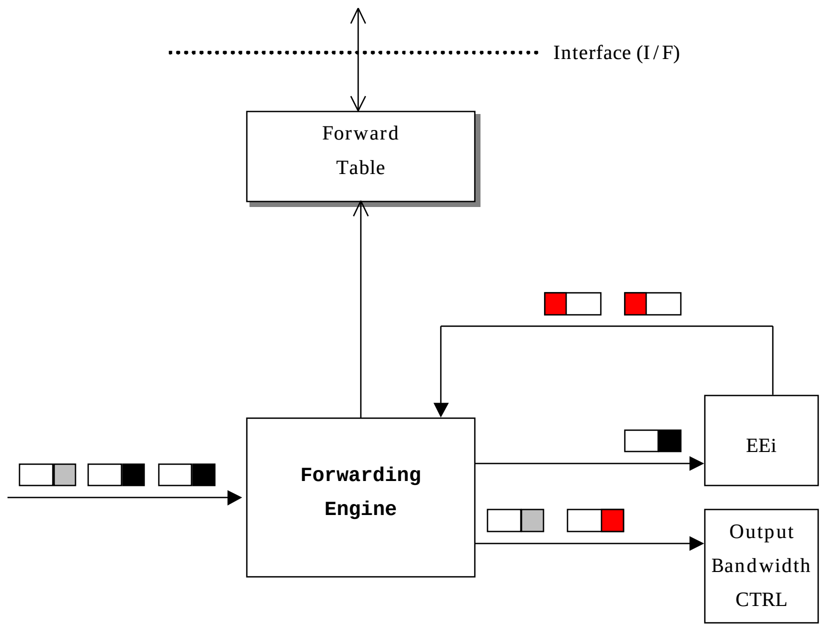


Figure 6-8: Data Forwarding Model

Table 6-7: Example of Forwarding Table

Flow						Action
SIP	**DIP**	**Port of SIP**	**Port of DIP**	**Protocol No**	**etc.**	
S-1	D-1	PoS-1	PoD-1	No-1		EEi then FWD
S-2	D-2	PoS-2	PoD-2	No-2		Forwarding
- - -	- - -	- - -	- - -	- - -		- - -
S-N	D-N	PoS-N	PoD-N	No-N		EEj then FWD

6.1.5.4.6 Design of Threads Control Component

Figure 6-9 shows the model of thread control. This thread control component controls the thread not to exceed using allowed resources. At first, the thread controller receives the limit of resources for controlling the thread from the outside component and stores it to the thread control table that is shown in the Table 6-8. Then the thread monitor observes the amount of resources that are consumed by a thread. When the amount of resources consumed by the thread exceeds the limit that is stored in the thread control table, the thread monitor informs the thread controller exceeding the limit for using the resources by the thread. After receiving the information about exceeded using of the resources, the thread controller enforces the usage of resources under the limit to the thread.

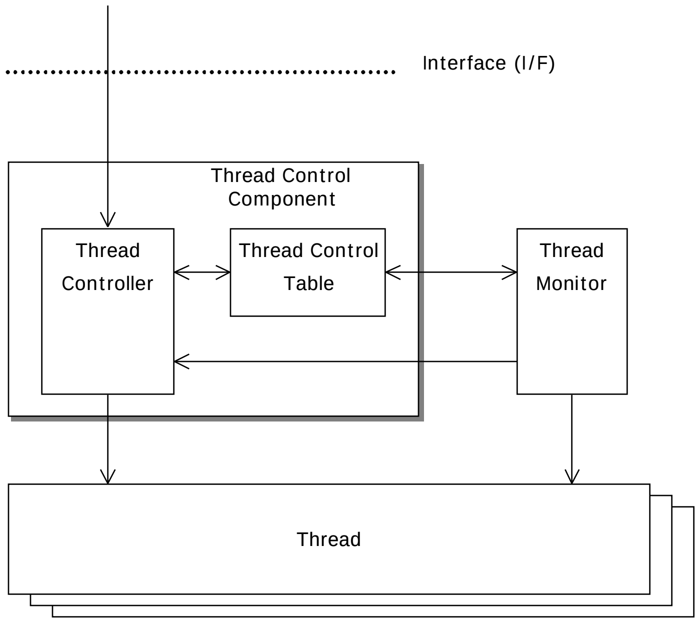


Figure 6-9: Thread Control Model

Table 6-8: Example of Thread Control Table

Identifier of Thread	**Limit of Resource**			
	CPU	**Memory**	**Storage**	**etc.**
Thread-1	time-1	mem-1	Storage-1	- - -
Thread-2	time-2	mem-2	Storage-2	- - -
- - -	- - -	- - -	- - -	- - -
Thread-N	time-N	mem-N	Storage-N	- - -

6.1.5.5Design of LRs Monitoring Components

The design of each logical resource’s monitoring component is described in this section.

6.1.5.5.1 Design of Queue Monitoring Component [GMD]

Figure 6-10 shows the model of queue monitoring. This queue monitoring component monitors the queues and interacts with the queue control component. The information about queue quotas is stored in the queue control table. The queue control table is used by both the control and the monitoring component. The monitoring component may also interact with an external component by invoking callbacks.

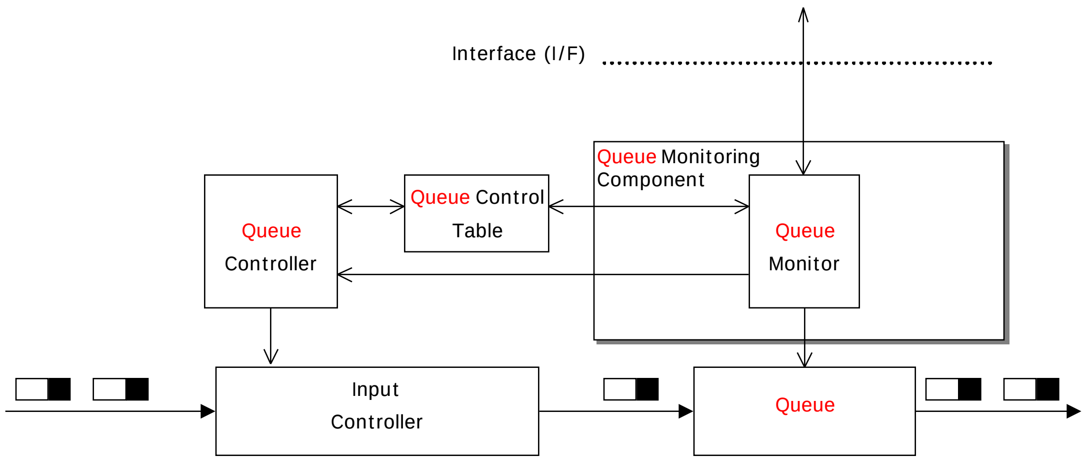


Figure 6-10: Queue Monitoring Model

6.1.5.5.2 Design of Classifier Table Monitoring Component

Figure 6-11 shows the mode of monitoring the data classifying. The monitoring component keeps statistics for the classes defined in the classifier table. This is done by receiving events from the classifier or by examining in counters held by the classifier. If the classifier doesn't support notifications on classifications or internal counters the monitoring component cannot provide statistics but only the entries in the classifier table. The monitoring component may also interact with an external component by invoking callbacks.

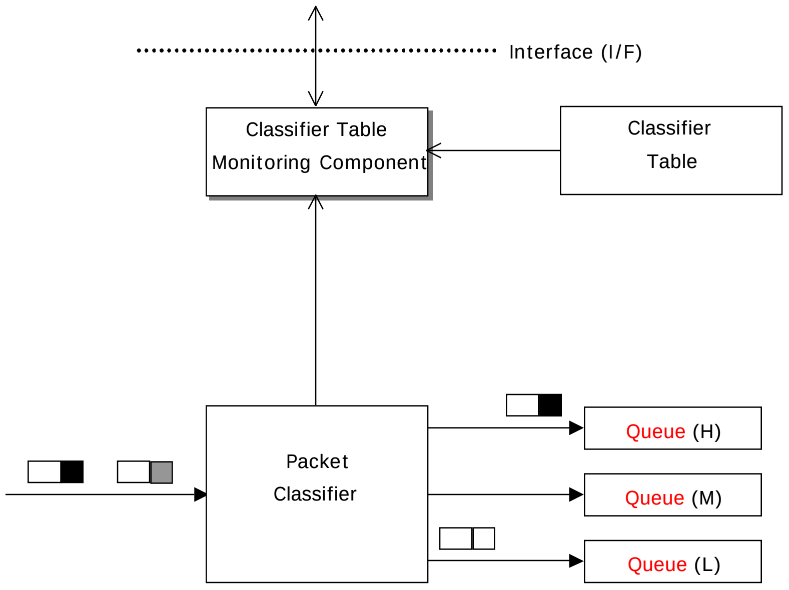


Figure 6-11: Packet Classifying Monitoring Model

6.1.5.5.3 Design of Computation Table Monitoring Component

Figure 6-12 shows the model of monitoring packet processing. The monitoring component keeps statistics for the computational entities defined in the computation table. This is done by receiving events from the CPU or by examining counters held by the CPU. If the CPU doesn't support notifications on dispatches or internal counters the monitoring component cannot provide statistics but only the entries in the computation table. The monitoring component may also interact with an external component by invoking callbacks.

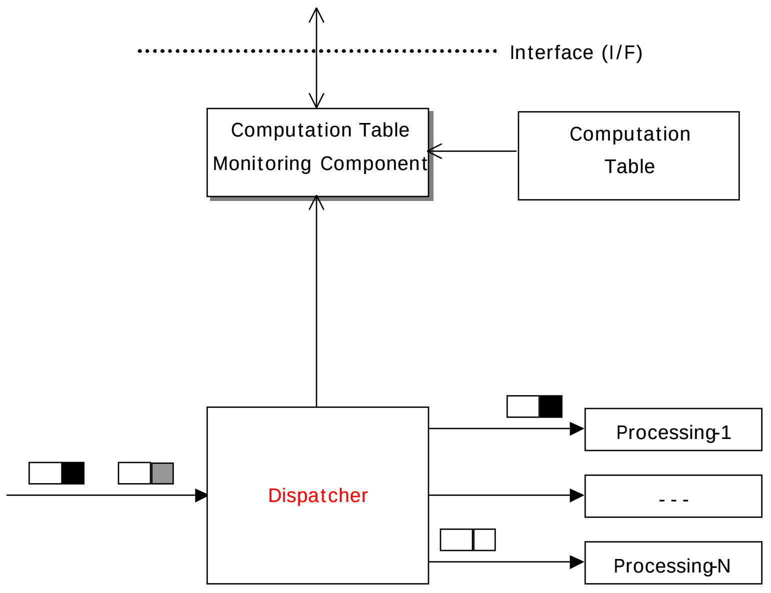


Figure 6-12: Packet Processing Monitoring Model

6.1.5.5.4 Design of Filtering Table Monitoring Component

Figure 6-13 shows the model of monitoring the packet filtering. The monitoring component keeps statistics for the filters defined in the filter table. This is done by receiving events from the packet filter or by examining counters held by the filter. If the filter doesn't support notifications on drops etc. or internal counters the monitoring component cannot provide statistics but only the entries in the filter table. The monitoring component may also interact with an external component by invoking callbacks.

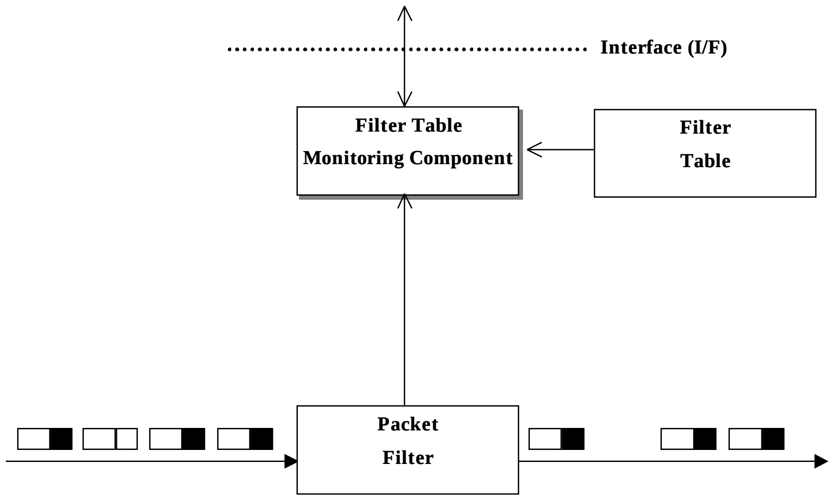


Figure 6-13: Data Filtering Monitoring Model

6.1.5.5.5 Design of Forwarding Table Monitoring Component

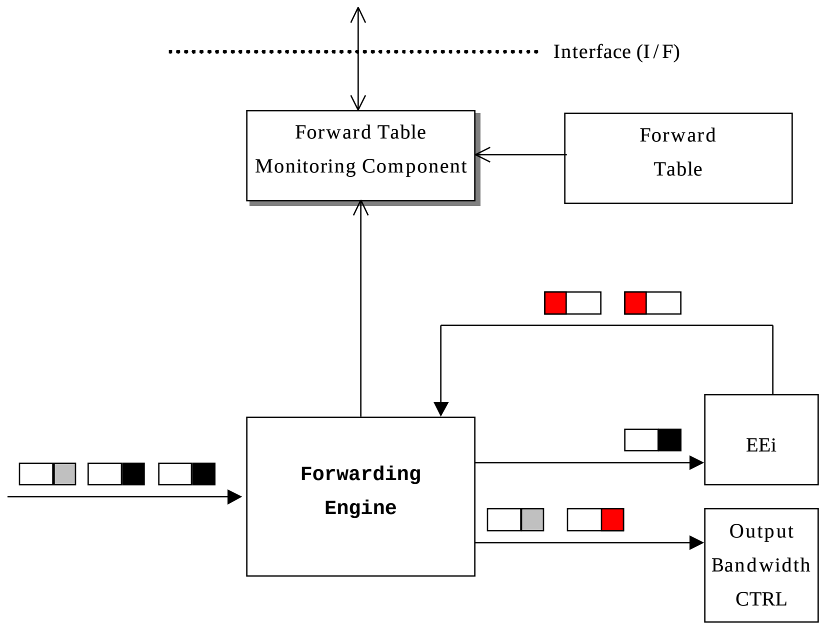


Figure 6-14: Data Forwarding Engine Monitoring Model

Figure 6-14 shows the model of monitoring the packet forwarding. The monitoring component keeps statistics for the destinations defined in the forwarding table. This is done by receiving events from the packet forwarder or by examining counters held by the forwarder. If the forwarder doesn't support notifications on drops etc. or internal counters the monitoring component cannot provide statistics but only the entries in the forwarding table. The monitoring component may also interact with an external component by invoking callbacks.

6.1.5.5.6 Design of Thread Monitoring Component

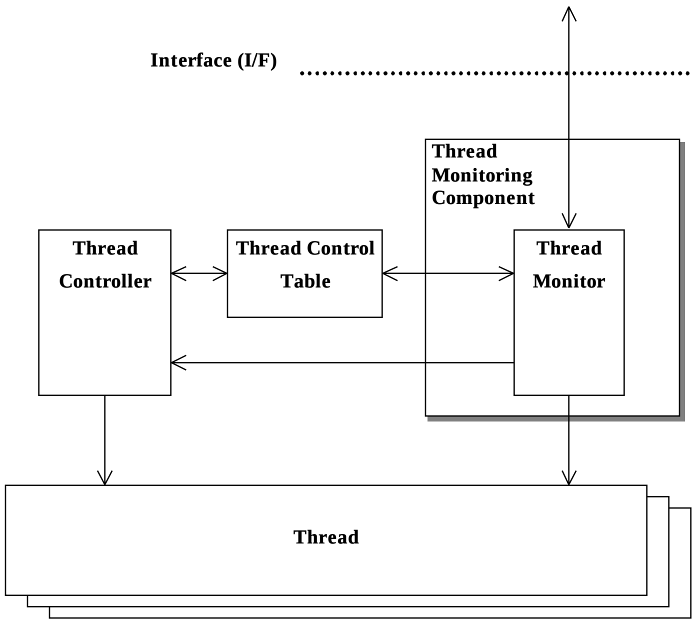


Figure 6-15: Thread Monitoring Model

Figure 6-15 shows the model of thread monitoring. This thread monitoring component monitors the threads and interacts with the thread control component. The information about thread quotas is stored in the thread control table. The thread control table is used by both the control and the monitoring component. The monitoring component may also interact with an external component by invoking callbacks.

6.1.5.6Design of Physical Resources Control Components

The design of each physical resource’s control component is described in this section.

6.1.5.6.1 Design of CPU Control Component

Figure 6-16 shows the model of CPU control. This CPU control component controls the CPU time. At first, the CPU controller receives the data for controlling the CPU time from outside component and stores it to the CPU control table. Then CPU controller controls the CPU time for each VE/service based on the CPU control table that is shown in the Table 6-9.

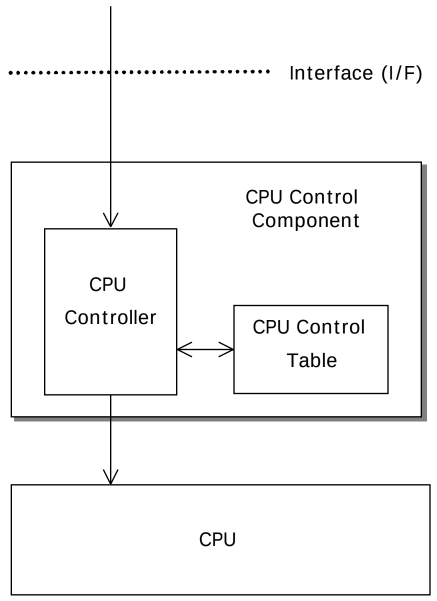


Figure 6-16: CPU Control Model

Table 6-9: Example of CPU Control Table

CPU Capacity	Allocation in VE	Allocation in Service
100%	VE-1	VE-1_Service1
		VE-1_Service2
		- - -
		VE-1_ServiceN
	- - -	- - -
	VE-N	VE-N_Service-1
		VE-N_Service-2
		- - -
		VE-N_Service-N

6.1.5.6.2 Design of Data Bus Control Component

Figure 6-17 shows the model of bus bandwidth control. This data bus control component controls the bus bandwidth. At first, the data bus controller receives the data for controlling the bus bandwidth from outside component and stores it to the bus control table. Then data bus controller controls the bus bandwidth for each VE/service based on the data bus control table that is shown in the Table 6-10.

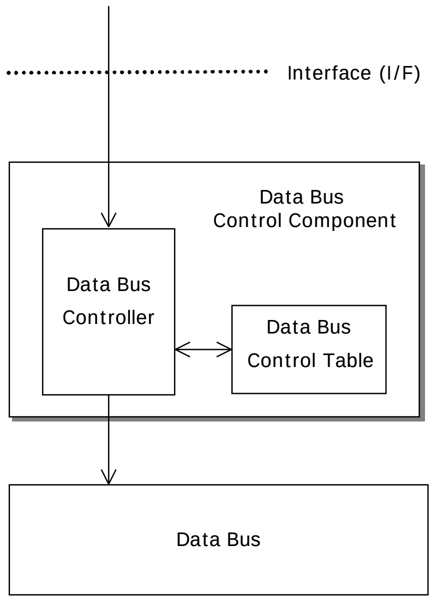


Figure 6-17: Data Bus Control Model

Table 6-10: Example of Data Bus Control Table

Bus Bandwidth Capacity	Allocation in VE	Allocation in Service
100%	VE-1	VE-1_Service1
		VE-1_Service2
		- - -
		VE-1_ServiceN
	- - -	- - -
	VE-N	VE-N_Service-1
		VE-N_Service-2
		- - -
		VE-N_Service-N

6.1.5.6.3 Design of Input Bandwidth Control Component

Figure 6-18 shows the model of input bandwidth control. This input bandwidth control component controls the input bandwidth. At first, the input bandwidth controller receives the data of bandwidth allocation from outside component and stores it to the input bandwidth control table that is shown in the Table 6-11. In addition, the input bandwidth controller receives the monitoring bandwidth of flow from input monitor and stores it in the table. Then the input bandwidth controller calculates the action that enforces the real input bandwidth to match the allocated bandwidth. For example, if the input bandwidth is larger than allocated bandwidth, discarding ratio is calculated. On the other hand, if the input bandwidth is not larger than allocated bandwidth, no action is assigned. After that, the input bandwidth controller controls the input bandwidth based on the action that is shown in the Table 6-11.

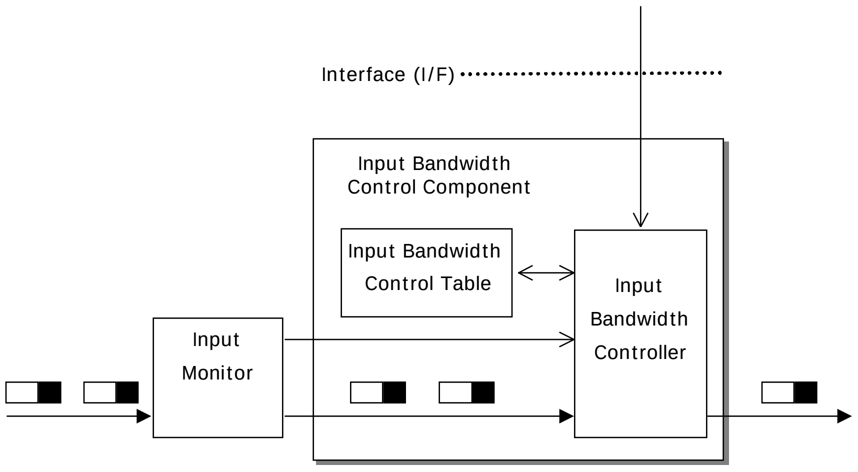

Interface (I/F)
Input Bandwidth Control Component
Input Bandwidth Control Table
Input Bandwidth Controller
Input Monitor

Figure 6-18: Input Bandwidth Control Model

Table 6-11:Example of Input Bandwidth Control Table

Input Bandwidth Capacity	Allocation in VE	Allocation in flow	Monitor in flow	Action
100%	VE-1	VE-1_Flow-1	Monitor_ VE-1_Flow-1	NO
		VE-1_ Flow-2	Monitor_ VE-1_Flow-2	-10%
		- - -	- - -	- - -
		VE-1_ Flow-N	Monitor_ VE-1_Flow-N	NO
	- - -	- - -	- - -	- - -
	VE-N	VE-N_ Flow-1	Monitor_ VE-N_Flow-1	NO
		VE-N_ Flow-2	Monitor_ VE-N_Flow-2	-20%
		- - -	- - -	- - -
		VE-N_ Flow-N	Monitor_ VE-N_Flow-N	NO

6.1.5.6.4 Design of Memory Control Component

Figure 6-19 shows the model of memory control. This memory control component controls the memory allocation and access etc. At first, the memory controller receives the request for allocation of memory per VE/service and stores it in the Table 6-12. Then memory controller controls the memory access for each VE/service based on the memory control table.

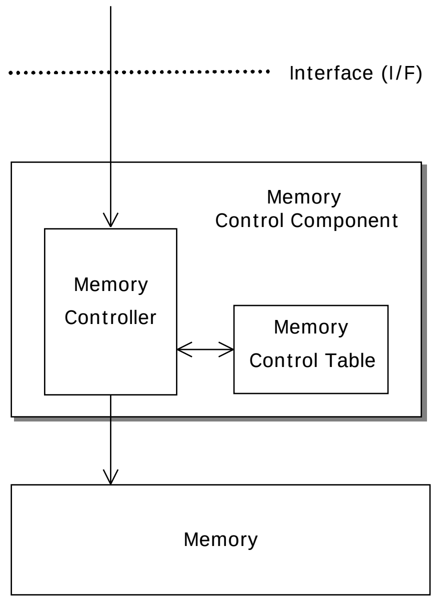


Figure 6-19: Memory Control Model

Table 6-12:Example of Memory Control Table

Memory Capacity	Allocation in VE	Allocation in Service
100%	VE-1	VE-1_Service1
		VE-1_Service2
		- - -
		VE-1_ServiceN
	- - -	- - -
	VE-N	VE-N_Service-1
		VE-N_Service-2
		- - -
		VE-N_Service-N

6.1.5.6.5 Design of Output Bandwidth Control Component

Figure 6-20 shows the model of output bandwidth control. This output bandwidth control component controls the output bandwidth. At first, the output bandwidth controller receives the data of bandwidth allocation from outside component and stores it to the output bandwidth control table that is shown in the Table 6-13. In addition, the output bandwidth controller receives the monitoring bandwidth of flow from output monitor and stores it in the table. Then the output bandwidth controller calculates the action that enforces the real output bandwidth to match the allocated bandwidth. For example, if the output bandwidth is larger than allocated bandwidth, discarding ratio is calculated. On the other hand, if the output bandwidth is not larger than allocated bandwidth, no action is assigned. After that, the output bandwidth controller controls the output bandwidth based on the action that is shown in the Table 6-13.

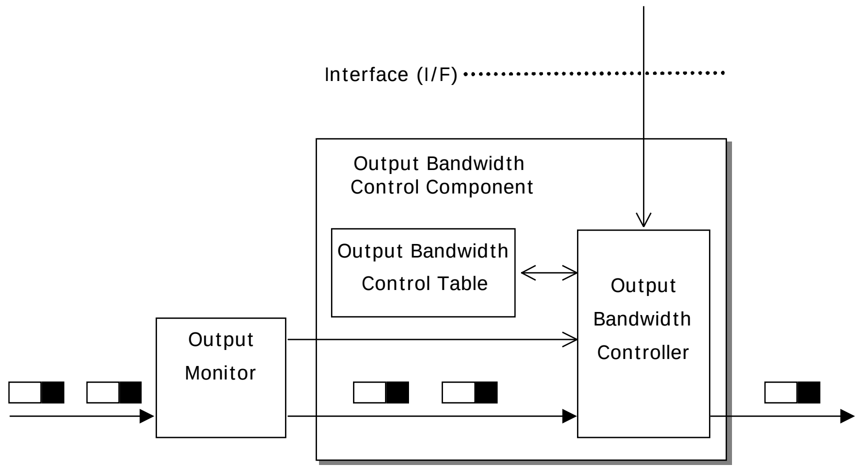

Interface (I/F)
Output Bandwidth Control Component
Output Bandwidth Control Table
Output Bandwidth Controller
Output Monitor

Figure 6-20: Output Bandwidth Control Model

Table 6-13: Example of Output Bandwidth Control Table

Output Bandwidth Capacity	Allocation in VE	Allocation in flow	Monitor in flow	Action
100%	VE-1	VE-1_Flow-1	Monitor_ VE-1_Flow-1	NO
		VE-1_ Flow-2	Monitor_ VE-1_Flow-2	-10%
		---	---	---
		VE-1_ Flow-N	Monitor_ VE-1_Flow-N	NO
	---	---	---	---
	VE-N	VE-N_ Flow-1	Monitor_ VE-N_Flow-1	NO
		VE-N_ Flow-2	Monitor_ VE-N_Flow-2	-20%
		---	---	---
		VE-N_ Flow-N	Monitor_ VE-N_Flow-N	NO

6.1.5.6.6 Design of Storage Control Component

Figure 6-21 shows the model of storage control. This storage control component controls the storage allocation and access etc. At first, the storage controller receives the request for allocation of storage per VE/service and stores it in the Table 6-14. Then storage controller controls the storage access for each VE/service based on the storage control table.

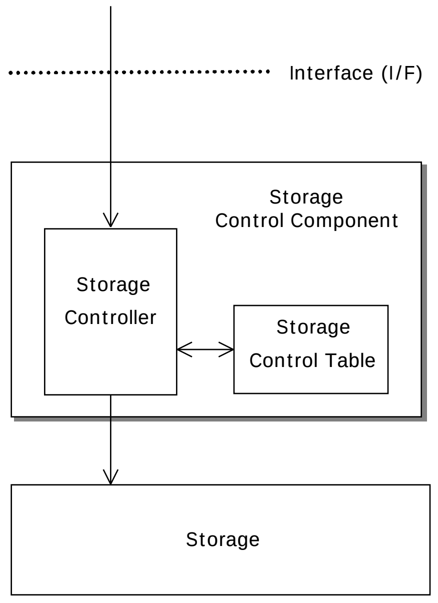


Figure 6-21: Storage Control Model

Table 6-14: Example of Storage Control Table

Storage Capacity	Allocation in VE	Allocation in Service
100%	VE-1	VE-1_Service1
		VE-1_Service2
		- - -
		VE-1_ServiceN
	- - -	- - -
	VE-N	VE-N_Service-1
		VE-N_Service-2
		- - -
		VE-N_Service-N

6.1.5.7Design of Physical Resources Monitoring Components

The design of each physical resource’s monitoring component is described in this section.

6.1.5.7.1 Design of CPU Control Monitoring

Figure 6-22 shows the model of CPU monitoring. The CPU monitoring component monitors the allocation of CPU time to a VE/service. It allows to retrieve statistics from the CPU and to examine the CPU control table. The monitoring component may also interact with an external component by invoking callbacks.

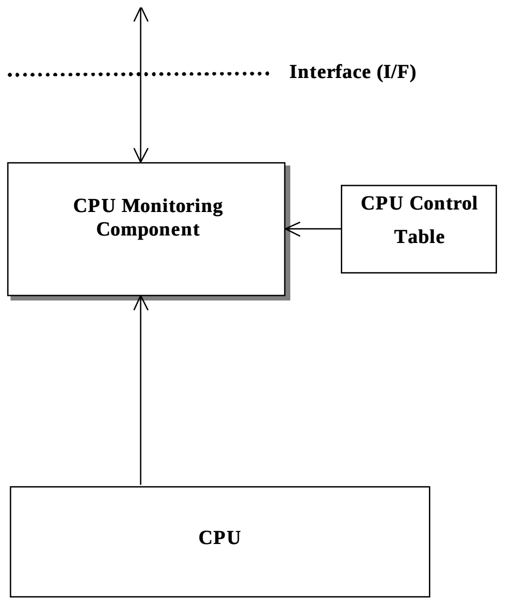


Figure 6-22: CPU Monitoring Model

6.1.5.7.2 Design of Data Bus Monitoring Component

Figure 6-23 shows the model of data bus bandwidth monitoring. The data bus monitoring component monitors the allocation of bandwidth to a VE/service. It allows to retrieve statistics from the data bus and to examine the data bus control table. The monitoring component may also interact with an external component by invoking callbacks.

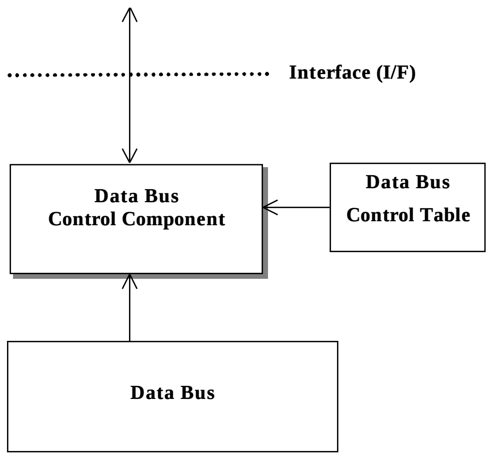


Figure 6-23: Data Bus Monitoring Model

6.1.5.7.3 Design of Input Bandwidth Monitoring Component

Figure 6-24 shows the model of input bandwidth monitoring. The input bandwidth monitoring component monitors the bandwidth used by a flow and interacts with the input bandwidth controller. It allows to retrieve statistics for the flows and to examine the input bandwidth control table. The monitoring component may also interact with an external component by invoking callbacks.

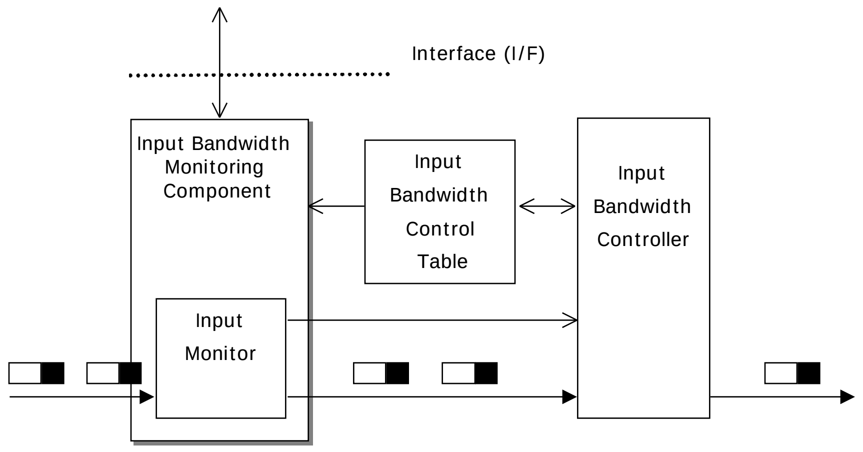

Interface (I/F)
Input Bandwidth Monitoring Component
Input Bandwidth Control Table
Input Bandwidth Controller
Input Monitor

Figure 6-24: Input Bandwidth Control Model

6.1.5.7.4 Design of Memory Monitoring Component

Figure 6-25 shows the model of memory monitoring. The memory monitoring component monitors the allocation of memory to a VE/service. It allows to retrieve statistics from the memory and to examine the memory control table. The monitoring component may also interact with an external component by invoking callbacks.

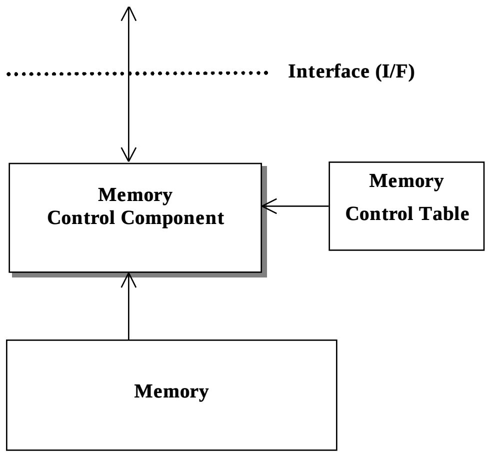


Figure 6-25: Memory Monitoring Model

6.1.5.7.5 Design of Output Bandwidth Monitoring Component

Figure 6-26 shows the model of output bandwidth monitoring. The output bandwidth monitoring component monitors the bandwidth used by a flow and interacts with the output bandwidth controller. It allows to retrieve statistics for the flows and to examine the output bandwidth control table. The monitoring component may also interact with an external component by invoking callbacks.

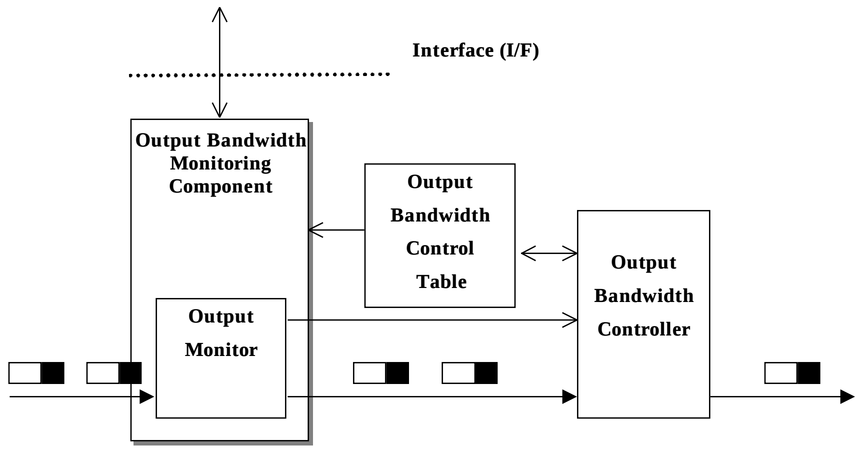

Interface (I/F)
Output Bandwidth Monitoring Component
Output Bandwidth Control Table
Output Bandwidth Controller
Output Monitor

Figure 6-26: Output Bandwidth Control Model

6.1.5.7.6 Design of Storage Monitoring Component

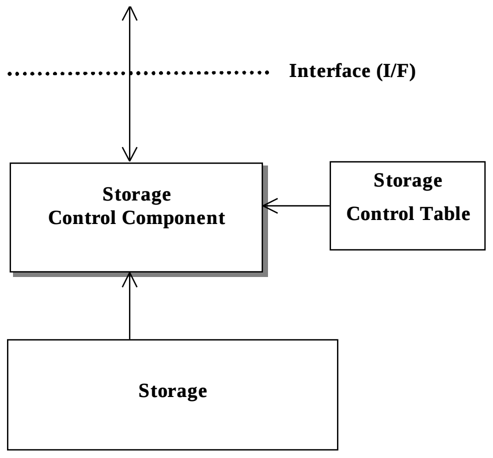


Figure 6-27: Storage Monitoring Model

Figure 6-27 shows the model of storage monitoring. The storage monitoring component monitors the allocation of storage to a VE/service. It allows to retrieve statistics from the storage and to examine the storage control table. The monitoring component may also interact with an external component by invoking callbacks.

6.2 Security Framework & Design

6.2.1 Introduction

6.2.1.1Motivation and scope

Basic FAIN AN principles, have serious consequences for the operation of an AN. The possibility of loading and executing active code in ANNs imposes considerable threats to expected operation of AN/ANN due to flaws in active code, malicious attacks by unauthorized users, and conflicted code execution. Thus, security in AN deals mainly with protecting system (AN infrastructure) from malicious (unauthorized) and erroneous use. The central objective of FAIN security architecture is to guarantee robust/secure operation of AN infrastructure despite the unintentional or intentional misbehaving of users, i.e. their respective active code/active packets.

Fulfilling these objectives is fundamental for the usability of ANs. Clearly, if it is trivial for any user to intentionally degrade the performance of an AN or any single ANN, or to bring down the AN/ANN, then ANs are not really usable. Note the difference between degrading performance of (disabling) an ANN and that of an AN. It is possible that specific network service, i.e. the respective active code, is consistent with a local security policy of an ANN; however, due to global, network wide behaviour of the protocol, it can degrade the performance of (part of) network or even completely disable it.

Furthermore, if an unintentional error in the design of a new network service, its implementation (active code), or its configuration can degrade performance of an AN/ANN or disable an AN/ANN, then ANs are not really usable.

Finally, if a malicious or unintentional misbehaving of any user can severely degrade or even disable the network services perceived by other user(s) of an AN but with no affect on the ANN/AN, then again, ANs are not really usable.

The threat model for active networks (see section 6.2.2.3) covers three broad classes of security issues: protecting AN infrastructure from users and active code, protecting users and active code from other active code, and protecting users and active code from AN infrastructure. However, the scope of initial FAIN security architecture is limited mainly to the first class, i.e. protecting AN infrastructure from users and active code.

6.2.1.2Definitions

An *Active Network (AN)* consists of arbitrary number of interconnected nodes, which can be active or passive. Interconnection between nodes is not physically secure and not dependent on any link layer technologies. The network is not administrated by a single authority.

An Active Network Node (ANN) is a computer system capable of running or interpreting active code. An ANN connected to the AN can communicate with other nodes in the AN. An ANN has possession over its resources, which can be offered to the users.

An *Active Packet (AP*) is a packet consisting from packet headers, active code related part and data. Active code can be either included or referenced in the packet.

An *Active Code (AC*) is a piece of program code that can be loaded to an ANN and run. While running, code can access, delete or modify data, use, request or release ANN or network resources and make request to use ANN or other AC services all in the security context in which the code is executed. Code is injected into the network by explicit or implicit request by a user and can be executed once or can run for some time. By executing or running AC provides new functionality, service or protocol in the network that is sometimes called active application in end-to-end terms.

An *active network user* is any user that can inject active code into the network. Users can be differentiated by their role in an AN. Typical users in AN can be anonymous user, user in a domain, administrator, service provider …

6.2.2 Generalized AN model and threats analysis

The fundamental property of FAIN AN is the possibility to dynamically inject active code, that implements new functionality, into the network. This code is executed or interpreted by ANNs and this way it provides end-user applications with application specific network services. We assume that FAIN AN consists of unlimited number of network nodes and some of them are active (ANN).

Active code is injected into the network via active packets, which carry active code itself or its reference to the repository, where the code can be fetched. Code can be executed in the nodes within the packet path. Execution provides new functionality in the network, which can be temporary or permanent. It can also produce new packets. Each execution use some of the ANN and AN resources, like CPU, storage and bandwidth, again temporary or permanent. Code in the ANN in AN can be injected, removed or replaced by explicit or implicit user or network node request. Intended usage of active network technologies is worldwide. Additionally, the following properties apply to generalized AN model [14]:

- an AN is a distributed system
- an AN is a packet-switched network, as opposed to circuit-switched
- not all nodes in an AN need to be active
- an AN explicitly provides for computation inside the network, but
- the primary goal of active networks is communication, not computation
- the contents of an active packet can legally change inside the ANNs[5]
- not all packets are active
- an AN consists of multiple domains, each controlled by a different administration

6.2.2.1Elements of the AN Infrastructure

Generally we can distinguish three key elements in the AN infrastructure:

Execution Environment (EE). This is the place where the active code executes. The EE offers access to the core node resources via a policy-controlled scheme. This can be for instance a mobile agent system that takes care of the execution of an agent.

Active Code (AC). This is the code that is actually executed in the EE of the node. The code could be written in any general-purpose language e.g. Java, C etc as long as the EE supports it, or even contain references to code already installed in the active node. By execution in the EE, the code programs the node according to user's preferences.

Active Code Carrier (ACC). The active code is carried from the source host to the destination host. There are two ways of actually moving the code to the target node known as the in-band and the out-of-band programming methods.

6.2.2.2AN Programming Methods

In the ***in-band programming*** method the active code is integrated into every packet of data sent to the AN node (also known as the capsule approach). The EE on the node executes the program and adopts the functionality of the node for the specific packet or the specific flow.

[5] In ANNs the payload (data part) can be changed also, not just header fields.

On the contrary in the ***out-of-band programming*** method the active code is injected in the AN node in a different session from the actual data packets that it affects. The user could install any time on the node the desired code. This code would then execute based on internal (e.g. according to node's EE schedule) or external (e.g. user activation command) events and program the node to process the desired data selectively. The data is recognized by specific tags or even by categorization e.g. all data coming from a specific node.

6.2.2.3Threat Model of Active Networks

Active networking supplies the users with the ability to download and execute code within a node. That is by its nature a security critical activity. In such an infrastructure the security implications are far more complex than in current static environments. In AN the author of the active code, the user who deploys it, the owner of the node hardware, the owner of the execution platform can be different entities governed by different security policies In such a heterogeneous environment security becomes an extremely sensitive issue. The possibility of loading and executing active code in ANNs imposes considerable threats to expected operation of AN/ANN due to flaws in active code, malicious attacks by unauthorized users, conflicted code execution etc. Thus, security in AN deals mainly with protecting system (AN infrastructure) from malicious (unauthorized) and erroneous use. The central objective of FAIN security architecture is to guarantee robust/secure operation of AN infrastructure despite the unintentional or intentional misbehaving of users, i.e. their respective active code/active packets.

AC is transferred to the node or is itself mobile e.g. in the form of a mobile agent. Therefore the attacks that AC and also the EE are susceptible to are more than those in current passive networks.

In general we can have:

- Misuse of an active network node by the active code
- Misuse of active code by other active code.
- Misuse of active code by an active network node.
- Misuse of active code and/or execution environment by the underlying network infrastructure.
- Misuse of the Active Network as an entity

Finally a combination of the above categories is possible. These kinds of attacks (the complex and collaborative ones) are very difficult to be detected not to mention prevented or effectively tackled. Classical examples include the co-operation of various hosts and ACs against another EE or AC.

Threats can also be analysed from the perspective of a single ANN, and from the network-wide perspective. Of course, threats to a single ANN apply also to the whole AN (domain). However, network-wide threats can be more subtle and harder to combat, since they are based on the global, distributed nature of network protocols, and thus, their respective active codes.

6.2.2.3.1 Misuse of an Active Network Node by the Active Code

Malicious AC while executing in an ANN can exploit security weaknesses in the node. It can perform attacks such as:

Masquerading. An AC may claim the identity of a trusted AC and therefore be granted access to resources that is not entitled to. Such AC besides being objected to false security schemes can also damage badly the reputation of a legitimate AC in the community.

Denial of Service. Malicious AC may overuse intentionally all resources and services provided by an ANN and degrade system performance. As a result the platform can not satisfy legitimate requests from other ACs. Furthermore if e.g. the security service is blocked further security breakouts could be introduced.

Unauthorized access. An AC that manages to bypass the authentication stage can harm an ANN. With various tricks or false language implementations an AC can bypass authorization and authentication stages and obtain access to private data.

Complex attacks. Here more than one ACs co-operate in order to attack a host. These are the most difficult attacks as they are strategically planned and can be event triggered. These collaborative kinds of attacks are very difficult to identify not to mention to deal with.

6.2.2.3.2 Misuse of AC by Other AC

These sets of threats are also very critical because of their variety of victims. An AC can perform various attacks against another AC including:

Repudiation. An AC can deny participation in a transaction or a communication although this has actually taken place. An ANN cannot prevent such an action but can provide sufficient evidence to assist with the resolution of such cases.

Masquerading. In case of an AC to AC communication one of the parties may try to disguise its identity and deceive the other one. Masquerading harms both the victim AC and the AC whose identity is being assumed.

Denial of Service. An AC can send spamming messages to another AC or false requests to keep it busy and degrade the CPU power, disk space or AC's response time.

Unauthorized Access. An AC not properly authenticated and authorized can directly interfere with another AC and perform various attacks e.g. change AC’s internal state, access/change its data, trap an AC and modify its configuration parameters or internal goals, steal info etc

6.2.2.3.3 Misuse of Active Code by an Active Network Node

An ANN has complete control over the execution of an AC. Therefore it can perform attacks such as:

Masquerading. An ANN can disguise as another ANN in order to deceive an AC and obtain its sensitive information. If the AC cannot reliably verify the EE and it trusts the false environment it is given, it will surely be an easy target.

Denial of Service. A malicious ANN may ignore AC requests or introduce unacceptable delays to services. Deleting an AC or suspending it for enough time so that the operations the AC wanted to perform are not valid any more or have no meaning is an example.

Eavesdropping. An ANN can monitor external/internal communications of the AC including every instruction executed by the AC. Therefore it has access to all unencrypted or public data the AC carries. So, it can for instance, invade its privacy by fully accessing AC's memory space and acquiring info like electronic money, secret keys etc.

Data and state manipulation. A malicious ANN can manipulate data and/or state of the AC and therefore interfere with AC’s normal execution or even worse control and guide AC’s execution based on falsely given perspective of environment. Furthermore, an ANN can interfere with AC's communications and alter them.

Cloning. Cloning an AC and then using the clone to analyse the original AC and its objectives is another type of attack.

We mentioned above the main threats that exist in an AN infrastructure. Of course, a combination of them makes it even more difficult to prevent or deal successfully with it. All the abovementioned security breakouts are performed when an AC is interacting with an ANN. The AC relies on an ANN to transport its code safe and secure to the desired host or execute it correctly, and that places an amount of trust to an ANN anyway.

6.2.2.3.4 Misuse of AC and/or ANN by external parties

Threats exist also while the AC traverses the network from host to host. One external attacker could perform all kind of attacks such as masquerade, denial of service, unauthorized access, copy and replay, alteration etc. A not so superficial scenario is the following: EEs are run by a user e.g. in a Unix host. By misconfiguration the user that runs the EE allows other users to access and modify the files that are stored on disk e.g. the policy files. Then another user (local or even remote via compromised www scripts) could easily change the policy file and allow his AC to execute with full access rights. The difficulty with these kinds of attacks is that they cannot be dealt with at all, as they use resources not controlled directly by the specific product (in this case the EE). A product that runs in a Unix environment is vulnerable to all kind of attacks via the security holes of the Unix system. Such kind of attacks cannot be predicted by the designer of the EE or the AC and are also out of the scope of this work.

6.2.2.3.5 Misuse of the Active Network as an entity

Badly designed AC can use AN in unattended or malicious manner but mainly on users behalf. Stale ACs are not revoked in a whole network.

By exploiting architectural or implementation flaws in AC or AN one could be using some part of the network as amplifier to bring down ANN or part of the network. Active Code that is node-safe is not necessarily and network-safe.

Within an ANN data, resources and services can be used in unauthorized way, modified or exposed. Stale code can be in the node cache or the node hasn't restarted stale ACs service. Furthermore heterogeneity issues such as different policy mapping and inter-domain differences complicate further trust and security enforcement decisions within an active network.

6.2.3 Security requirements

The AN infrastructures come with a double status; that of a legacy networks (e.g. data transportation) and that of a highly programmable network model adjustable on the fly to application’s specific requirements. Thus, the spectrum of threats for such a new network model is extended. It includes not only the threat models of the legacy node and network systems, but also those of general purpose computing engines (e.g. safeness). The basic requirements for a secure active network are:

6.2.3.1Authentication

Authentication is an action of securely identifying the user that is requesting a particular service or resource within an ANN. Authentication is crucial since other security critical decisions depend on it.

6.2.3.2Policy based code invocation/authorization in ANNs

Each request to perform specification(s) within ANN should be subject to authorization checks, performed by a respective ANN. Local security policy dictates who is allowed to do what in an ANN. In essence, authorization decides whether user's request is compliant with local security policy and if so, grants appropriate privileges to the requester.

Once the AC is installed in the node database, we need a policy scheme to say who can access this piece of code and under which policy conditions. Pre-installed pieces of code in the node could also be seen as extended libraries or services. Other AC may require results from these components in order to achieve its goals. In that case, we have to specify which code has the rights to execute pre-installed node components and make use of their feedback. The policy could be set by the node or EE administrator, or even by the user who originally installed the code there. By being able to invoke other components we have more lightweight active code (since parts of services can be found dynamically at runtime and not be implemented in one big program) and we promote security (the preinstalled code could be set there by the node admin who has tested it thoroughly).

6.2.3.3Policy enforcement

Policy enforcement prevents users that have been granted privileges to perform certain actions from abusing those privileges. In other words, it prevents users that have been granted access to the system from abusing granted privileges, e.g. performing granted actions in an unauthorized way.

6.2.3.4Integrity

Active code/packet integrity: It has to be assured that the code/packet has not been tampered with while in transit over the network. This prevents malicious attacks, such as forging and replaying of active code and packets in order to obtain privileges.

ANN system integrity: This requirement addresses the situations when an ANN system itself has been remotely compromised rather than its services having been degraded. Detecting a compromise of crucial system components is beneficial to AN security, since a compromised ANN can simplify many kinds of serious security breaches not only at the compromised system but network-wide.

6.2.3.5Interference free active networking

Several types of interference between users, i.e. their respective active codes can occur in an ANN:

- Conflicts at modification of ANN state
- Conflicts at modification of ANN configuration
- Conflicts in resource consumption/availability
- Conflicts in interpretation of security policies

Unless prevented or resolved, these conflicts can cause exceptions with unpredictable consequences for the ANN operation.

Furthermore because within the FAIN architecture multiple EEs of the same or different type can coexist special countermeasures have to be taken. We should make sure that these EEs do not affect each other and that the code executed in each of them does not result in unauthorized interactions with the code executed in another EE. This is the sandbox idea but this time between EEs. Each EE should have its own resources and manage them according to its needs. Preventing unauthorized interactions between various ACs that execute within an EE is EE’s responsibility.

6.2.3.6Common, cross plane security architecture

Security architecture should cover transport plane, control plane, and management plane as defined in FAIN reference model (see section 4). From the security perspective all the planes should be treated equally, i.e. there are common security mechanisms protecting all planes.

6.2.3.7Dynamic Management of Security

Active code/packets bring along their respective security policies, which have to be added to the ANN policy database and enforced by an ANN. It must be possible to dynamically add, remove or change security policies in the ANN policy database and on the whole network with minimal human interference. This calls for mechanisms that support a distributed way of propagating policy or even support for a centralized policy. We have here two extremes. In the distributed scheme, each node has its own policy while in the centralized scheme each node pulls the policy from a central server. Of course there are other possible schemes that take advantage of both ways e.g. co-existence of policies that could be pulled from or pushed to a third trusted node (not the central server). In that scheme, one could have policy references from node to node (a www connection style of policies). This eases the segmentation of the network to policy domains and simplifies the creation of virtual private networks with common policy schemes. In any case, this should be transparent to the network administrator.

6.2.3.8Verification of active code

An active code is verified prior to execution. If it fails verification test it is discarded without being executed. Failing the verification test means that code cannot be trusted to perform its declared function, which could lead to the abuse of granted privileges. Verification complements, but also partly overlaps policy enforcement.

6.2.3.9Verification of the EE

The AC should have the ability to verify the environment where it executes. EEs should also be able to perform such verification. The verification process could take place before the AC is installed in the node (using trusted services in another node) or even during runtime. The AC may contain private data or may even be environment sensitive and release info based on its internal list of goals. Therefore, the verification of the EE is necessary for building secure infrastructures.

6.2.3.10 Active Code Revocation

We should be able to maintain locally and network wide a list with revoked ACs. If a piece of AC behaves maliciously then the node administrator would forbid it to execute although its credentials (the user that signed it) may be valid. This could be done for various other reasons e.g. we forbid execution of ACs coming from competing companies. These black lists could be pulled/pushed also in a central point within the network for network-wide usage. A credential server in combination with credential expiry as the technique could be applied , as this can scale easily while inducing a controllable "window of risk".

6.2.3.11 Persistence

This service should exist within the node. During node shutdown the node should suspend all ACs and then, after rebooting, reactivate them and continue its normal operation. Otherwise, due to an accidental node reboot the AC could be lost. Persistence service provides a more reliable and fault tolerant active network infrastructure.

6.2.3.12 Secure Auditing

We should audit events occurring in the base of active node services as well as the events occurring because of actions taken by the AC. Decomposing auditing activity in this way allows the active node base code to be simpler as it doesn’t have to implement complex handling of audit messages. Audit logs should be securely stored possibly in a distributed scheme (better survivability to attacks) and access to them should be policy based. Apart from the node audit, the active code may perform its own auditing and possibly report it via an interface to the node’s audit facilities.

6.2.3.13 Predefined node manipulation

Many network operators are very much concerned with the idea of executing code within a node, mainly because of the obvious or hidden drawbacks such an action carries. Thus, there is a need that specific interfaces are provided to the users via which they can interact with the node in predefined ways. The network operator installs itself the necessary code and services in the node and allows the user to call this code with predefined (and well tested) parameters. Although again we have code executing, we can predict the result of this execution since the node’s status will change to one of the predefined ones. In order to make this idea more clear we can think of the following scenario: A user wants to install a compression filter for video transmission. Instead of providing his own implementation, the user calls the already installed by the administrator code with parameters that satisfy his goal.

E.g. VideoCompression <algorithm> < final_format> <time>.

Now the node administrator knows that the user has programmed the node in a specific and already known way. This can be seen as a hybrid approach since active code is executed (active network) but actually the node is manipulated via predefined interfaces (programmable network).

6.2.3.14 Safeness and global restrictions on active code

This is a requirement in order to protect the AN as a whole. It is possible that specific protocol, i.e. the respective active code, is consistent with a local security policy of an ANN. Thus, it passes local authorization test and is granted execution. However, due to global, network wide behaviour of the protocol, it can degrade the performance of the network or even completely disable it. For example, consider a novel routing protocol that creates routing loops and does not decrement hop-count (TTL) value of the looping packets. This scenario would eventually bring parts of the network to a halt.

AC safeness is a difficult goal to achieve. We have to make sure that the code that executes in a node executes correctly. There could be code that comes from trusted users, but that does not execute safely and compromises intentionally or unintentionally node’s security. Mechanisms that guarantee safe AC execution have to be used.

6.2.3.15 Explicit management of trust relationships between pairs of domains

This is a requirement in order to protect the AN as a whole. In a closed environment, users of a system are known in advance. In this case it is possible for a Source of Authority to issue credentials to the users, which are required for authorization. However, in an open, multi domain environment, users residing in one domain are generally not known to the system in another domain. Two main issues that have to be resolved by inter-domain trust management are: how are appropriate credentials assigned to users and how are these credentials mapped to granted privileges by the authorization system.

6.2.4 Meeting Security Requirements

In the following we give an overview of secure operation scenario for the following security requirements:

- authentication
- policy based authorization in ANNs
- policy enforcement
- active packet/code integrity
- interference free active networking
- common cross plane security facilities
- dynamic security policy management
- verification of active code
- ANN system integrity
- restrictions on global behaviour of active code
- explicit management of trust relationships between pairs of domains

This enables us to identify the necessary security functional blocks and their interrelations, which we then combine in a High-level security framework for active networks, which is depicted in section 6.2.4.12. This framework is a conceptual view of security protections required in FAIN AN and is used as a basis for the design of the initial FAIN security architecture (see section 6.2.5.3).

6.2.4.1Authentication

Authentication is a process of verifying an identity of an entity. Usually it is based on public key cryptography, which means that each entity presenting a request to an ANN has to have a public/private key pair and a public key certificate.

Overview of secure operation scenario:

- User employs its private key to digitally sign the active packet it sends and adds a signature to the packet
- ANN uses the public key certificate to verify the validity of the user's public key.
- If valid, ANN employs user public key to verify the digital signature of the packet

It follows from the secure operation scenario that following functional blocks must exist and co-operate in every ANN in order to provide authentication:

- crypto engine
- authentication engine

It also follows from secure operation scenario that these facilities are required for authentication:

- public key cryptography
- PKI infrastructure

6.2.4.2Policy based authorization

There are two options for authorization:

- it is based on user identity
- it is not based on user attributes, not his identity

Overview of secure operation scenario:

- ANN administrator specifies security policy, which governs the operation of an ANN, via the ANN management system
- along with a request an ANN is presented with a set of respective user's credentials; here, a request is general---possible actions include: installation/execution of active code, reservation of (HW) resources, access to (other) packet flows, management of security policies, modification of configuration data, etc.
  a credential is for example an attribute certificate, which is used for aggregation, i.e. for mapping of a specific user to larger group of users for simple and scalable authorization
- ANN performs authentication and verifies the ownership of credentials
- ANN evaluates the request:
  - if authorization is identity based:
    ANN evaluates the combination of request and user ID against
    - the respective set of local security policies (ANN policy, other code's policy) and
    - environment variables (e.g. time of day, system load, user generated load, ...)

    and grants or denies the request
  - if authorization is not based on user ID:
    ANN checks the combination of request and the presented set of credentials against the security policies and environment variables and grants or denies the request
- if necessary, ANN searches the AN and fetches the credentials (e.g. public key certificates, attribute certificates), that are missing, but are required for authorization

- based on all relevant data, ANN decides whether to grant a request; this decision is used for policy enforcement

It follows from the secure operation scenario that following functional blocks must exist and co-operate in every ANN in order to provide policy based authorization:

- authorization engine
- policy database
- policy manager
- credentials database
- credential manager
- source(s) of environment variable(s) (e.g. total system load, and per “user" loads)

It also follows that policy based authorization requires mechanisms for measurement of current resource consumption (i.e. environment variables).

6.2.4.3Policy enforcement

Policy enforcement operates on running “user" processes in an ANN. These processes perform the actions from the users' request, which have been granted by authorization. While authorization is only a decision making process, enforcement is the process, which actually intercepts requests and either blocks them or allows their execution, based on the authorization decision.

Overview of secure operation scenario for policy enforcement:

- ANN intercepts user's request
- ANN checks the request against authorization decision (granted privileges)
- if request adheres to the granted privileges, it is processed accordingly
- request is discarded, if it is not compliant with granted privileges; further actions can be taken, such as killing the offending process

Note that there are two types of resources in an ANN that policy enforcement must protect, in conjunction with authorization. Obviously, there are (scarce) hardware resources, such as processor computing power, memory, and link bandwidth. However, there are also *functional resources*, such as specific files, methods (functions) etc. These have to be protected as well, which means that enforcement mediates all function calls.

It follows from the secure operation scenario that the following functional block must exist in every ANN in order to provide policy enforcement:

- policy enforcement engine

It also follows that following general mechanisms are required for policy enforcement:

- mechanisms for enforcement of hardware resource usage
- mechanisms for intercepting function calls

6.2.4.4Active code and packet integrity

Active packets and code can be tampered with while in transit over the network: a malicious user can modify, replay, or even forge packets and code with the intent to gain higher privileges. Packet/code Integrity addresses all these issues.

It is important to note, that in general contents of an active packet comprise a static part, which is not changed while in transit (e.g. user data), and a dynamic part, which can legally change within the ANNs en route (e.g. parameters and data used by active code). Thus, “end-to-end"[6] integrity does not suffice in ANs. It has to be augmented with hop-by-hop integrity, which leads us to the following prerequisites for integrity:

- user has a public key pair and a valid PK certificate
- each ANN maintains a (non-compromised) secret key with each of his neighbours

Overview of secure operation scenario for integrity:

- each packet contains:
    - digital signature of static part of packet
    - a MAC[7] of dynamic part of packet
    - a special value for protection against replay
- upon receiving a packet, ANN checks:
    - the contents of the packet against its MAC and digital signature values and
    - validity of the anti-replay value
- integrity is provided if all checks are successful; ANN processes the active packet, optionally modifies it, generates a new MAC value of the dynamic part, and sends it to the next-hop ANN

It follows from the secure scenario that following functional blocks must exist and co-operate in every ANN in order to provide active packet/code integrity:

- integrity engine
- “private" security environment (holding ANN cryptographic material)
- authentication engine
- crypto engine

It also follows that these general mechanisms are required for active packet/code integrity:

- secure ANN key exchange mechanism
- digital signature mechanism
- symmetric encryption
- asymmetric encryption
- one way hash functions
- mechanisms required by specific replay protection scheme

[6] Quotes are used because end-to-end is not a proper naming, since it is the association between an end host and an ANN that is relevant, and not the association between two end-hosts. Better term would be “end-to-node" security.

[7] Message Authentication Code, known also as Data Authentication Code---DAC. Basically, this is a symmetrically encrypted hash.

6.2.4.5 Interference free active networking

ANN executes the active code of many users' simultaneously. Different conflicts can arise among these codes during their execution: conflicts at ANN state change, ANN configuration modification, consumption of resources, and security policy conflicts.

Here, there is no exact scenario for secure operation. Instead, we identify the mechanisms which can be employed for avoiding/resolving conflicts in an ANN. Currently, we identify the following mechanisms:

- priority, which can be used for active code and for policies:
  - active codes (policies) are assigned priority
  - when conflict arises, the code (policy) with higher priority takes precedence
- dynamically managed and global namespace for active codes

It follows from the scenario that following functional block is required for interference-free active networking:

- priority manager

Additionally, an active code name system may be needed as a part of external support facilities.

6.2.4.6 Common security framework for user, control, and management plane

Functionality of the network (i.e. network services) is often divided into three planes. User plane services directly manipulate user packets, while the other two planes provide supporting services, which affect user packets indirectly. Control plane encompasses signalling for dynamical setup of network state according to users' needs. Management plane comprises services for the management of user and control planes.

It is important to note, that basic active network operation mode applies equally to all planes: the active code, that users inject into network, can provide new or enhanced network services in user plane (e.g. an enhanced queuing algorithm), control plane (e.g. an enhanced multicast routing protocol), or management plane (e.g. monitoring and configuration service for a newly installed queuing algorithm). In other words, active networks do not make a distinction and the same mechanisms are used for dynamic provisioning of new or enhanced network services in all planes.

Since users can affect all three planes of an active network infrastructure, it is required that security framework protects all planes. This basically means that the common security framework protects all the resources in an ANN that belong to all planes. In other words, functional resources of control plane, such as control protocol state, and management plane, such as security policies and configuration information, must not be neglected.

Here, there is no specific secure operation scenario. Rather, it is once again emphasized that an ANN must protect the functional resources of its control and management planes, in addition to HW resources and functional resources of user plane.

Neither do we identify here the required mechanisms and functional blocks. We note that this requirement simply means that all other security requirements apply to all planes. Consequently, a cross plane security framework depends on all the functional blocks that are required by the rest of security requirements.

6.2.4.7Explicit and dynamic security policies

Security policies in an ANN have to be explicit and dynamically managed as opposed to static and integrated with policy enforcement implementation. The reason for this is that active networks allow dynamic provisioning of new and enhanced services. Consequently, the number and type of network services and the number of users that are ``present" in any given ANN is constantly changing. Policies have to reflect this fact and have to be changed accordingly, thus they have to be dynamic. In order to be dynamic, policies have to be specified explicitly and independently from policy enforcement implementation.

Put simply, policies in an ANN mandate who can do what in an ANN. Policies reflect operations and management decisions of an AN operator (administration), service level agreements between operators of different domains, SLAs between an operator and its customers, usage mode of various active codes, etc.

Overview of secure operation scenario:

- user presents an ANN a request to add, modify, or remove specific policy(-ies) along with its credentials
- policy based authorization kicks in
  specific policies govern, who is authorized to manage other policies
- if authorization succeeds, user is granted permission to manipulate the specified policies (specific objects in a policy database)
- policy enforcement kicks in to prevent user from abusing granted privileges, e.g. manipulating policies that were not specified in a request
- ANN checks for and resolves policy conflicts, i.e. situations where new or modified policy is in contradiction with existing policies
- when conflicts are resolved, policy is saved in an ANN for future reference

As seen from above, secure operation scenario for explicit, dynamic policies includes scenarios for policy based authorization, policy enforcement, and conflict resolution. Thus, all the functional blocks required for these security services apply here as well. Additionally, these functional blocks are required:

- policy database
- policy manager

6.2.4.8Verification of active code

Active code verification is additional preventive measure, which can lower the burden on policy enforcement. The main goal of verification is to prevent execution of an untrusted newly arrived active code. How an ANN decides whether to trust the code, depends on the specific approach to verification. Note that in general, trusting a code does not mean the same as trusting the user, which is represented by the code.

Overview of secure operation scenario:

- active code is loaded to an ANN
- the ANN checks if the code can be trusted
- if check fails, the request (code) is discarded, otherwise it is forwarded to authorization

As seen from above, the following functional block is required for code verification:

- code verification engine.

We note, that, depending on the technical approach, code verification can be null. That is, there is no explicit and distinct action taken to verify active code.

6.2.4.9System integrity

This requirement is the last line of defence when other ANN security measures have been defeated or circumvented and the system itself has been compromised, rather than only its services having been degraded. A compromised ANN can simplify many kinds of serious security breaches not only at the compromised system but network-wide.

This security requirement depends on the basic security assumption---what is the Trusted Computing Base (TCB) of the respective system architecture. The TCB is a core of an ANN, which is always trusted, i.e. it is assumed that it cannot be compromised.

Also note that when TCB is defined such that it contains all crucial system components, then this requirement does not apply.

There are two approaches to system integrity checks:

- static: integrity checks of crucial system components are performed once, at system boot [28]
- dynamic: integrity checks are performed repeatedly in a running system

Second approach is preferred for its dynamic response to system compromise incidents, however its viability needs further investigation.

Overview of secure operation scenario for dynamic system integrity checks:

- the TCB keeps integrity tokens for other crucial system components; integrity token is generally in the form of a hash, MAC or digital signature
- periodically, TCB verifies the integrity of crucial system components by calculating their integrity tokens and comparing them with the values in its token database
- if a compromised (faulty) component is discovered, TCB can trigger an alarm, or it can start a recovery process (from locally stored copy or remotely from a trusted server)
- crucial system components can be modified by a system administrator or security manager; whenever this happens, integrity tokens in TCB database have to be updated in secure fashion.

It follows from the secure operation scenario that the following functional blocks must exist and co-operate in every ANN in order to provide ANN system integrity:

- (integrity) token database, i.e. TCB database
- a system integrity checker within the TCB
- remote recovery server (optional)

6.2.4.10 Network wide protection

It is possible that specific protocol, i.e. the respective active code, is consistent with a local security policy of an ANN. However, due to global, network wide behaviour of the protocol, it can degrade the performance of the network or even completely disable it. A promising approach to address this issue is the use of market (economy) based schemes for protection of network resources, such as Marketnet [43] and Xenoservers [44]. It remains to be investigated how these schemes can be applied to FAIN framework.

6.2.4.11 Explicit management of interdomain trust

In a closed environment, users of a system are known in advance. On the contrary, in an open mutli-domain environment, users residing in one domain are generally not known to systems in another domain. Thus, some sort of trust between domains has to be established in order for authorization to be meaningful, i.e. allow differentiation of users from another domain. This issue needs further investigation.

6.2.4.12 High-level Security Framework

Based on the discussions in section 6.2.4, a high level security framework can be posited, the scope of which is depicted in Figure 6-28.

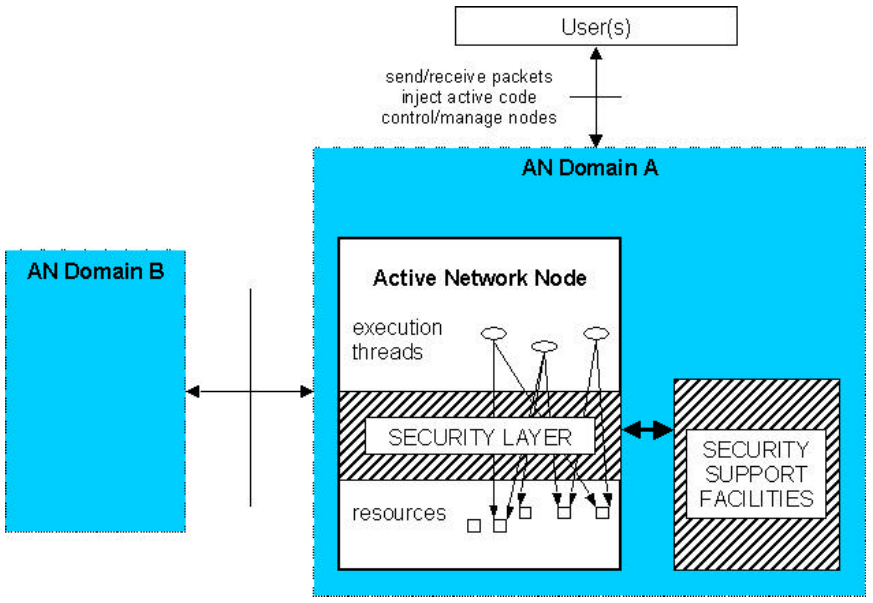


Figure 6-28: Scope of active network security framework

Security technology is an important middleware layer technology, which in ANNs prevents unauthorized use of node's functional, and hardware resources, which is achieved by mediating access to these resources. Aside from the security layer, which is a part of ANN architecture, security framework comprises support facilities, which are external to an ANN. This is due to the fact that actual mechanisms residing in the middleware security layer depend on various facilities, such as PKI infrastructure.

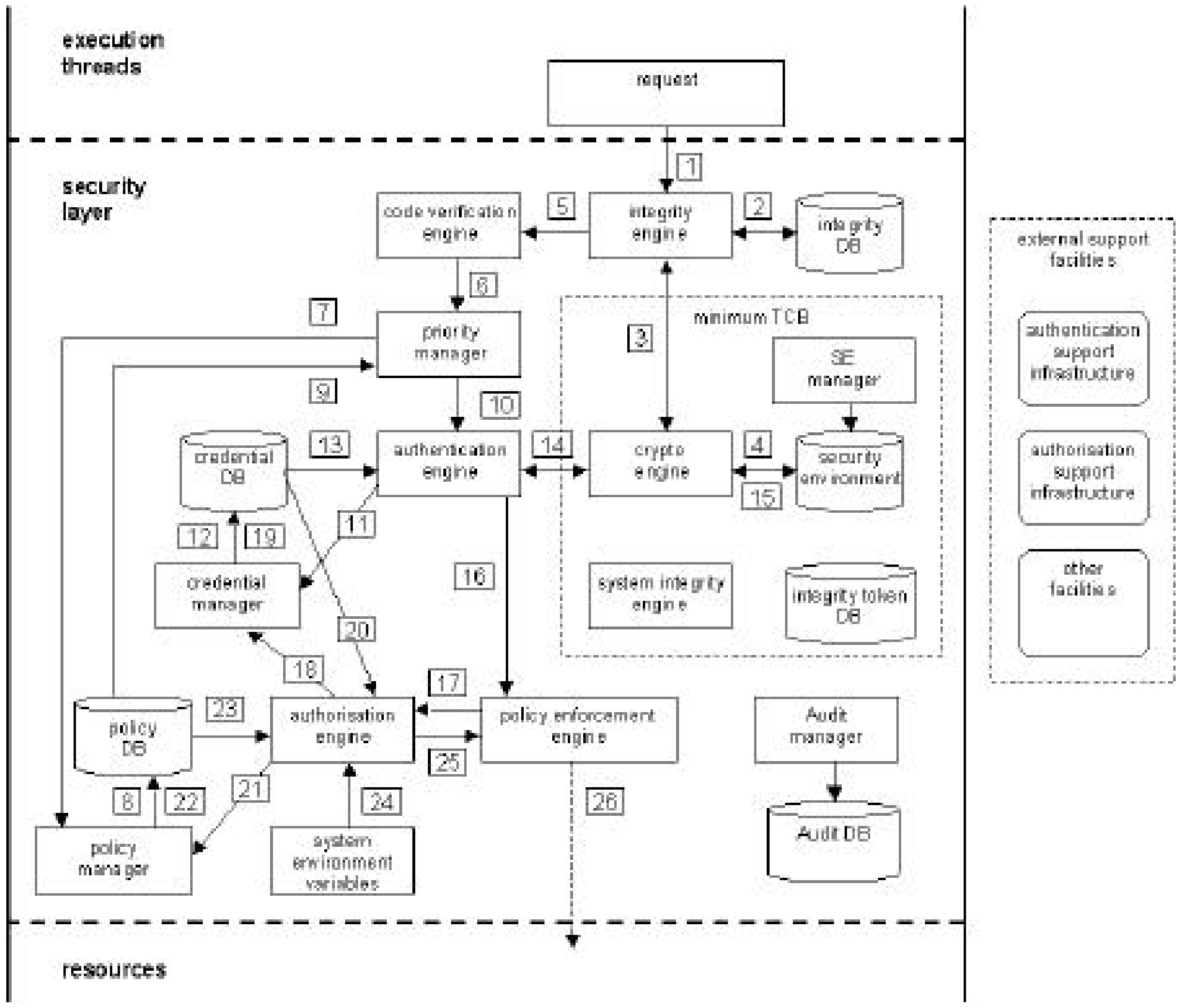


Figure 6-29: High-level security framework

Figure 6-29 shows how functional blocks which we identified in section 6.2.4 combine to form the AN security framework. It also depicts at the conceptual level, how a request (e.g. for execution of specific active code or to access particular resource) is processed by middleware security layer. This in general includes parsing the request and:

- checking packet/code integrity (1)
- verifying active code (5)
- assigning priority (6)
- authentication and checking ownership of credentials (10)
- making authorization decision (17)
- blocking or forwarding request, based on authorization decision (26)

In addition, all security relevant events in the ANN system are logged into audit DB by the audit manager. This information is used for intrusion detection and for security incident analysis, either by a human operator or automatically

The following security support facilities are also shown in Figure 6-29:

- authentication support infrastructure, such as PKI infrastructure and certificate repositories

- authorization support infrastructure, such as credential repositories
- other facilities, such as recovery servers, trusted time-stamp servers, global time-reference, etc.

6.2.5 Initial FAIN security architecture

6.2.5.1Security priorities for initial phase

In the initial phase (year 1) of the FAIN project only a subset of security requirements will be addressed in detail. Thus, we need to decide here which security requirements deserve higher priority. A list of high-priority security requirements comprises:

- authentication
- authorization
- policy enforcement
- active code/packet integrity
- code verification
- audit

We have compiled this list in light of the main objective of the FAIN security architecture, which is “to provide secure and robust operation of FAIN AN infrastructure in spite of unintentional and malicious misbehaving of AN users, i.e. their respective codes". From this perspective, our criteria in assigning priorities can be summarized as follows:

- How subtle is particular security requirement, i.e. the respective threat “behind" the requirement?
- More subtle yields lower priority.

Authentication, authorization and policy enforcement

FAIN ANN is essentially a multi-user system. As in any such system, enforcement of access control is a requirement of high significance within every FAIN ANN. On the other hand, FAIN aims at developing a flexible system. In order to achieve the desired level of granularity we decompose access control in authentication, authorization and policy enforcement. These three security requirements have the highest-priority within FAIN security architecture.

Active code/packet integrity

Active code is executed within an ANN and performs actions on behalf of a user. Therefore, active code is the “carrier of activity” and as such, it is a powerful tool when misused by malicious users, which tamper with active code while in transit over the network. For instance, the whole access control system could be circumvented, if the original active code can be modified or swapped with any other code. Similarly, there are ways to obviate access control system by tampering with active packets, such as cut & paste attacks and replay attacks. This is why protecting integrity of active code and packets deserves a high-priority.

Code verification

Protecting the active code integrity is a first step to ensure non-modification of the transient code. However this is considered pretty basic and we need to go beyond that in order to achieve a high level of security. The active code has to be somehow marked and tightly coupled with one or more entities, based on which further security decisions can be made. The code carries credentials from these entities, which have to be verified in order to set the security context within which this active code can execute. As code verification is critical into taking further security decisions, this is considered a high-priority requirement for the FAIN security architecture.

Audit

The Audit Manager component is an integrated part of the security architecture. Via this component

a) all events occurring from the usage of the security subsystem are implicitly logged for further future usage.

b) It also provides an interface to explicitly log any other events coming from other parts of the FAIN architecture in a clear and homogeneous way.

The info gathered by the audit manager are stored into the audit database and via a policy controlled way are available for further exploit.

By decomposing auditing activity in this way allows the active node base code to be simpler as it doesn't have to implement complex handling of audit messages. Audit logs should be securely stored not only locally on the node but also in a distributed scheme (e.g. [34]) as this offers better survivability to attacks against the node. Apart from the node audit, the active code may perform its own auditing and possibly report it via an interface to the node's audit facilities.

Modern computer systems do not emphasize enough on the significance of the audit facilities. However audit tools help in realizing possible security leaks (or even preventing some) and make sure that mistakes are not repeated. We feel that within the AN community special care has to be taken with audit activities and therefore it is also considered a high-priority security requirement.

6.2.5.2Technical approach

In this section we present proposed ways to tackle high-priority security issues. In many cases, there are several technical ways of realizing functionality represented by functional blocks which we identified in section 6.2.4. When deciding which option is best fitted to FAIN, we consider other project goals, such as flexibility and performance.

6.2.5.2.1 Active packet/code integrity

In general, protecting integrity of active packet/code while in transit over network involves cryptographic operations. The most common approach is as follows:

- at the sending end---generate integrity protection token (data):
  - calculate a hash of the packet/code
  - encrypt the hash to protect it from modifications
  - send the encrypted hash together with the active packet/code
- at the receiving end---verify the integrity of the packet/code:
  - decrypt the hash that accompanies the received active packet/code
  - calculate a hash of the active packet/code
  - compare the two hashes; if they differ active packet/code has been modified and should not be allowed execution

The hash value, which is carried along with active packet/code and is used for integrity, can be protected in either of two ways:

- with asymmetric encryption
- with symmetric encryption

If asymmetric encryption is used, integrity protection is provided by digital signatures and there is no need for ANNs to maintain a private/public key pair.[8] ANNs only need to be able to obtain the certificate chain, which verifies the validity of the public key of the party signing the active packet/code. Thus, the advantage of asymmetric encryption is that it eases management of encryption and decryption keys. However, the downside is that asymmetric encryption is on the order of two magnitudes slower than symmetric encryption.

[8] Note that other security requirements may/will impose this.

In case symmetric encryption is used, the encrypted hash is known as a MAC value. However, this requires each ANN to maintain a non-compromised private/public key pair and a public key certificate. ANN uses asymmetric encryption to establish a shared secret key with the sending end. Thus, asymmetric encryption in this case is still used, but this time only to set-up a secret key for symmetric encryption. Additional downside of symmetric encryption is that integrity protection requires a negotiation phase before active packet/code can be injected into the AN.

For initial FAIN security architecture we propose the combination of asymmetric and symmetric encryption for active packet/code integrity, in order to leverage the advantages of both. The proposed approach is as follows:

- each ANN has a public/private key pair and a public key certificate
- each ANN maintains a shared secret key with every of its direct neighbouring ANNs; neighbouring ANNs employ asymmetric cryptography for establishing and updating shared keys
- sending end signs active packet/code (using asymmetric encryption) and injects it into the AN
- the ingress ANN fetches the public key of the signer and verifies it against its certificate
- the ingress ANN then uses this key to check integrity of the received active code
- if active code is intact, ingress ANN calculates a MAC value, using a secret key it shares with the next hop ANN
- ingress ANN sends MAC value along with active packet/code and its signature
- every subsequent ANN
  - uses the secret key it shares with previous-hop ANN and checks integrity
  - calculates new MAC values using the secret key it shares with the (physical) next hop ANN
  - sends the new MAC value along with the active packet/code

The proposed approach represents a trade-off between FAIN goals of security and performance. On one hand, the described approach is based on the assumption that trust exists between ANNs, which obviously reduces the level of security. However, this is a valid assumption at least in a single domain, which is under the control of single authority. The trust within domain is applied by per-hop symmetric encryption.

On the other hand, this approach is advantageous for ANN performance, since it leverages high speed of symmetric encryption algorithms. Furthermore, because (pre-established) per-hop shared keys are used, it effectively eliminates the symmetric key negotiation phase. Note that per ANN public/private keys and per-hop cryptographic calculations are used, but this is required by active packet integrity, anyway.

6.2.5.2.2 Policy enforcement

In the initial phase our discussion is limited to enforcement mechanisms up to and including FAIN Node facilities level, i.e. we currently omit the discussion of policy enforcement within EEs/VEs.

Policy enforcement as defined in section 6.2.3.3 is the active component that enforces authorization decisions and thus enforces the use of ANN resources, which is consistent with local security policies.

We distinguish two types of resources, hardware and functional resources. Hardware resources include basic low-level ANN resources such as memory, storage capacity, CPU cycles and link bandwidth. Functional resources are high-level resources in the sense that they consume some portion of hardware resources. However, with these resources it is not important how much memory or storage space they consume but rather what purpose they serve within an ANN, i.e. what function they provide. Examples of functional resources include:

- special purpose files, such as configuration files,
- policy entries in the policy database,
- ANN state,
- ANN API functions themselves, etc.

We note that all resources in an ANN, hardware and functional, are accessible at certain node interface. In order to prevent unauthorized use of ANN resources, policy enforcement has to be scattered across different ANN subsystems that provide specific subsets of ANN API functions.

Thus, basic technical approach to policy enforcement is to add an “adaptation" software layer on top of every subsystem API, which mediates access to node API functions. Whenever an ANN function is called by an “external" entity (such as VE, EE, active code), this software layer:

- intercepts the request (call to node function) and suspends it
- provides call parameters to authorization engine effectively asking for authorization decision; parameters include requestor ID, called function name, object(s) name, amount of requested hardware resources, etc.
- when authorization decision is returned
    - if request is authorized, enforcement layer resumes the execution of the request
    - if request is not authorized, enforcement discards the request and thus prevents unauthorized actions from taking place

In addition to this “high-level" operation, policy enforcement also has to operate at low-level in order to enforce proper usage of low-level hardware resources. At the “lower level" enforcement is embodied in a more complex policing algorithm(s), which can control the scheduler(s) for specific resource and thus impose limits on resource usage by an entity. For example, ATM policing mechanisms fall into this category.

6.2.5.2.3 Authentication

Authentication is a process of verifying an identity claimed by or for a system entity [29]. Symmetric or asymmetric cryptography can be used for authentication. Symmetric cryptography is suitable only for closed systems due to its scalability problems. Thus, we propose asymmetric cryptography for FAIN. This requires every AN user to have a public/private key pair and a valid public key certificate.

Nevertheless, common remote authentication protocols employ a handshake, i.e. a two way communication in order to perform authentication. In active networks this would require an end-host to perform an authenticating handshake protocol with every ANN en route, which is clearly unacceptable. Thus, we propose the use of “unidirectional” procedure, where authentication is based on digital signatures and one-way communication from end-host to an ANN. Overview of this authentication scenario:

- User employs its private key to digitally sign the active packet it sends and adds a signature to the packet
- ANN uses the public key certificate to verify the validity of the user’s public key.
- If valid, ANN employs user public key to verify the digital signature of the packet

PKI infrastructure is needed to support authentication based on digital signatures.

6.2.5.2.4 Authorization

As we have seen in section 6.2.5.2.2, there will be several enforcement engines in FAIN ANN, each of them residing in a different FAIN ANN subsystem and responsible for mediating access to functions and resources the respective subsystem. On the other hand, authorization component can be either integrated with policy enforcement or logically separated from it. In the former case, there would also have to be one authorization engine per ANN subsystem. In the latter case, only one, general-purpose authorization engine can be implemented and used by all policy enforcement engines.

We recommend the latter approach for FAIN due to the following reasons:

- no duplication of work; this is especially important if we consider that design and implementation of any security component is a difficult and subtle task
- inherent flexibility as a consequence of separation of authorization from enforcement

- possibility of reuse of existing tools (see section 6.2.7.2).

6.2.5.2.5 Code Verification

Verification can enable us to trust to some extent that the active code will behave safely and properly and that we can have some guarantees on its resource usage on the node and in the network. But we shall say in general that verification provides only enhanced trust in proper and safe code execution, which normally is not related to the trust in the user on which behalf the code is executed.

At the first level verification will only enable us to define which code can actually be run on FAIN node. If code is not trusted it will be dropped or it will run on an EE with minimal facilities available. In the latter case the EE is the same one that will be used to run anonymous active code.

We can distinguish between two wide groups of verification:

1. Digitally signed code, so we trust in the user, organization or repository that has signed the code. Digital signature can be checked at the NodeOS level, immediately after it is available.
2. Various other mechanisms that can enhance the trust in proper and safe execution. These mechanisms mainly operate within EEs, and include techniques like proof carrying code [31], JAVA bytecode verification [32], inherited by language design like SNAP [33] and also other methods which deal more with authentication, authorization and enforcement.

If there is resource consumption estimate available, simple resource check is also possible. Since the scope of initial FAIN security architecture is limited to the NodeOS level, we propose the use of first approach, which employs digital signatures for code certification.

6.2.5.3Security Architecture

In this section we depict the proposed security architecture for initial phase of FAIN. In Figure 6-30, all parts "belonging" to security architecture are shaded. As depicted in Figure 6-30, FAIN security architecture roughly comprises three parts: security subsystem, other ANN security components, and external security support facilities. Note that the scope of initial FAIN security architecture does not include EE layer of FAIN ANN architecture.

6.2.5.3.1 Security Subsystem

Most of security critical decisions are made by security subsystem, which is one of several subsystems within an ANN. Security subsystem is also responsible for management of security critical data, such as encryption keys, credentials, and policies.

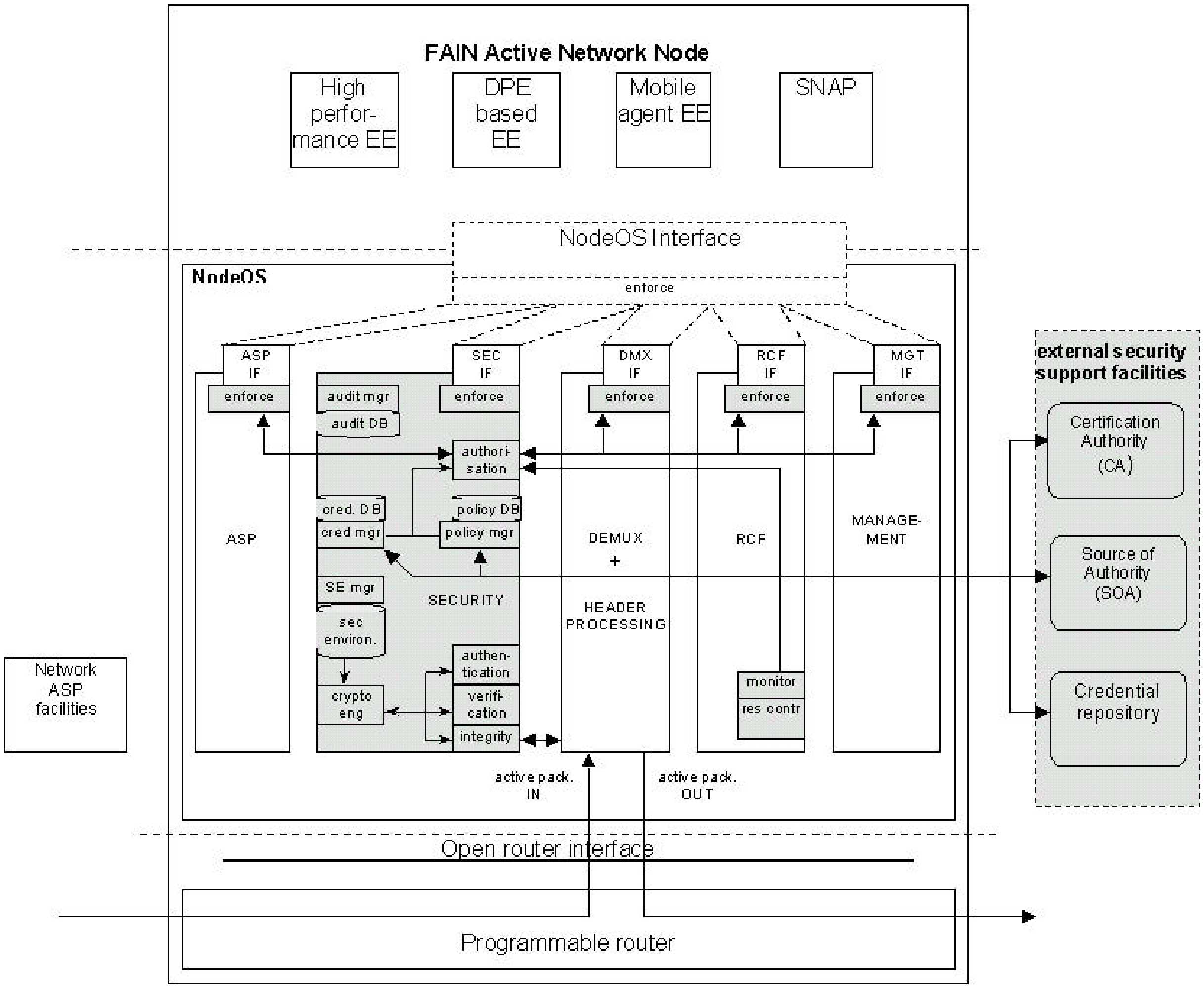


Figure 6-30: Initial FAIN security architecture

This subsystem is the core of FAIN security architecture and includes the following components (see Figure 6-30):

1. **Crypto Engine:** performs the actual cryptographic operations, such as symmetric encryption/decryption, asymmetric encryption/decryption, and hashing. It implements various cryptographic algorithms, which are used by other components in the security subsystem.
2. **Security Environment (SE):** in a secure fashion stores various encryption keys, which are required by crypto engine. For example, SE stores ANN’s public key pair (private and public key) and all secret keys that an ANN shares with its neighbours (one per neighbour).
3. **SE Manager:** is used for managing the keys in SE. SE manager can provide facilities for manual configuration of encryption keys and can also automatically manage keys, e.g. by triggering a key exchange protocol with neighbouring ANN.
4. **Integrity Engine:** checks the integrity of active packets and active code. It depends on integrity protection data contained within an active packet and on crypto engine to do the necessary cryptographic operations.
5. **Verification Engine:** performs code verification (at NodeOS level), if any. It may depend on special data contained within an active packet and on crypto engine to do the necessary cryptographic operations.

6. **Authentication Engine:** 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.
7. **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 ANN is authorized or not. Authorization engine provides this "service" to all policy enforcement engines in an ANN.
8. **Policy database:** stores security policies, which govern who can do what in an ANN.
9. **Policy Manager:** when asked by the authorization engine, 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.
10. **Credential database:** stores users' credentials, such as public key certificates and attribute certificates.
11. **Credential Manager:** when asked by authorization engine, 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. search and download credentials from an external credential repository.
12. **Audit database:** stores an audit log of security critical events.
13. **Audit Manager:** will be the place where all security architecture's components audit their function in order to be used later in resolution of problems or even to make decisions. E.g. an Intrusion Detection System would use a view of the audit DB in order to recognize attacks against the system. The audit could be also distributed for survivability reasons.

6.2.5.3.2 Other ANN security components

The second part of security architecture includes components that are part of ANN but are external to security subsystem. This includes policy enforcement engines and various components providing environment variables, e.g. resource usage monitor.

Various subsystems within an ANN offer their services and objects for use by users via their interfaces. Access to these objects and services is governed by security policies. Thus, enforcement of node security policies has to be performed at the point where they can be violated, i.e. at interfaces. At every ANN subsystem, a policy enforcement engine acts as an adaptation layer, which is responsible for mediating access to subsystem services and objects based on the authorization decision.

While authorization is only a decision making, enforcement is an active process that prevents access to services and objects by unauthorized users. Enforcement engine operates such that it suspend the request at interface, asks authorization engine whether this request is allowed and acts upon authorization decision, i.e. either allows or denies execution of the request.

In addition to these "high-level" enforcement engines, there are also "low-level" enforcement engines, which are tightly coupled with specific hardware resources available within an ANN and therefore they are considered as part of Resource Control Framework (RCF).

Finally, there are some components in an ANN, which provide authorization engine with necessary data to make authorization decision. For example, resource usage monitor provides data on current hardware resource consumption by particular user, and a clock provides current time and date.

6.2.5.3.3 External Security Support Facilities

In the initial security architecture, we envisage these security support facilities:

- Certification Authority (CA)
- Source of Authority (SoA)
- Credential repository

Authentication based on digital signatures requires a user to have a public key pair and a valid public key certificate. Public key certificate binds a public key and an identity of its owner and are issued by a trusted third party called Certification Authority (CA). When a user enters an ANN, he must present his public key certificate to authentication engine. Alternatively, he can provide a pointer to his public key certificate in the form of a reference to certificate repository.

In the initial phase of FAIN, a single CA is sufficient for demonstration and testing purposes. This can be later extended with more CAs forming a fully-fledged Public Key Infrastructure (PKI).

Similarly, a scalable approach to authorization requires a user to have one or more attribute certificates. Attribute certificates bind public keys directly to privileges, which can be exercised by the owner of the key. Attribute Certificates are issued by a trusted party called Source of Authority, which may not necessarily be the same as the Certification Authority. When a user enters an ANN, he must present one or more attribute certificates either directly or by reference to a repository. Later, when a user tries to execute an action, attribute certificates are used by the authorization engine to decide whether he has the necessary privileges.

Credential repository can store both, public key certificates and attribute certificates. Repository can be implemented in many ways, such as a directory service or a web repository.

6.2.5.4Operation of security architecture

Basically, there are two checkpoints where security functionality from Figure 6-30 is employed to protect an ANN: when a user enters an ANN and when a user tries to execute some action within an ANN. The former is represented by an arrival of an active packet in ANN and we call it *entry-level protection*. The latter occurs when a request for certain operation arrives at NodeOS interface and we call it *execution-level protection*. In the following subsections we will use textual description and sequence charts to demonstrate how security protections work in these two cases.

In addition to these two types of security protections, one can distinguish two operations, which do not directly provide any security protections. Rather, these two are a sort of “backplane” operations, which support entry- and execution-level security protections. These support operations are: setup of a shared secret key between neighbouring ANNs and obtaining the missing credentials from a node external repository.

A secret key, which is shared by a pair of neighbour ANNs, is used for hop-by-hop symmetric encryption of portions of active packet, which is leveraged e.g. for integrity protection. To setup a shared secret key between two ANNs, any key exchange protocol can be used. Key exchange has to be performed when a new ANN is added to the AN and whenever the key lifetime expires.

On some occasions, a situation may arise, when the credentials needed to make authorization decision are not present in an ANN. In this case, the missing credentials have to be searched for and obtained from somewhere in the network, usually from a repository service.

Finally, there is an audit facility within FAIN ANN, which is responsible for keeping a log of all security critical events within an ANN. This information is required for activities such as intrusion detection and analysis and assessment of security breaches.

6.2.5.4.1 Entry-level security protections

Figure 6-31 depicts a sequence of security operations that are performed for every packet that arrives at ANN. These security checks are aimed at detecting anything suspicious about this particular packet and, if so, discarding it. A packet is only de-multiplexed to appropriate to EE if it passes all checks.

Upon entering an ANN, an active is first processed in order to extract information (SecurityData in Figure 6-31) needed for security checks. This information includes:

- Digital signatures, which are used for authentication, integrity, and verification
- MAC values, which are used for integrity protection
- Public key certificate(s), which are used for checking digital signatures
- Attribute certificates, which are used for authorization

After this information has been provided to security subsystem, entry-level security checks are triggered. Security subsystem verifies credentials, checks integrity of active packet and active code, performs code verification (if any), and performs authentication and returns the result of these operations to the de-multiplexing subsystem.

Only if all these checks are successful, the packet is allowed to “enter” an ANN, i.e. it is first processed at NodeOS level (e.g. IP processing) and then forwarded to appropriate EE for further processing. If any of security checks fails, this is reported to de-multiplexing system, which discards the packet.

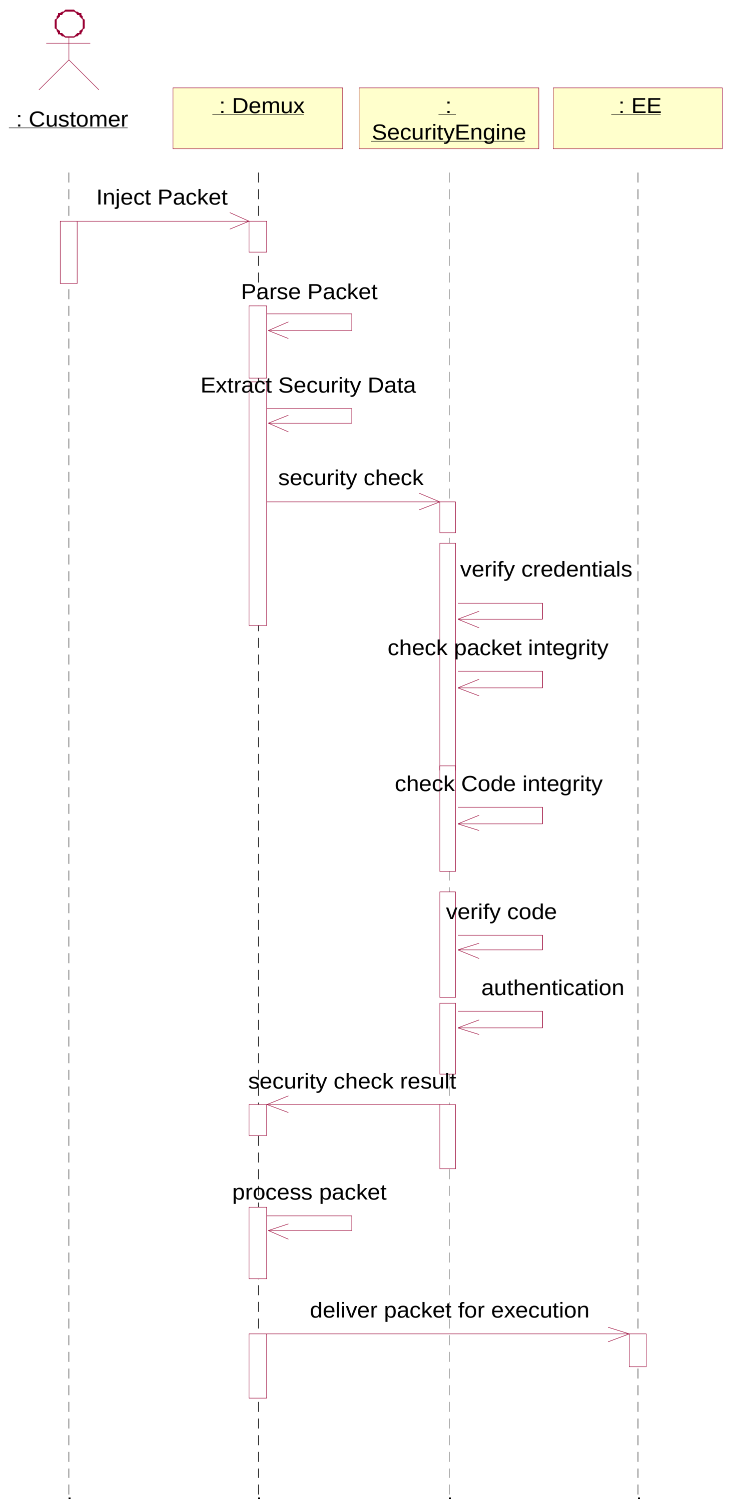


Figure 6-31: Entry-level security protections

6.2.5.4.1.1 Active packet integrity

Figure 6-32 depicts a sequence of events, which are required to check integrity of the newly received active packet.

Integrity protection is based on cryptography as explained in section 3.1.1. Every packet carries along at least one special token, which is used for integrity protection (IntegrToken in Figure 6-32). This token has a form of a digital signature and/or a MAC (Message Authentication Code).

It is envisaged that both approaches to packet integrity will be used in FAIN. This is due to the fact that digital signatures are required for authentication, so it is sensible to leverage digital signatures for integrity as well. However, this applies only to static parts of an active packet, which do not change en route and can be signed by the source of the packet. For the dynamic parts of an active packet, which can change within an ANN, per hop integrity protections based on MAC must be used.

Figure 6-32 depicts the case, when active packet integrity is provided with MAC, i.e. by leveraging hop-by-hop symmetric encryption. In this case, there is no need for integrity engine to process public key certificates. Instead, crypto engine can find the required decryption key in ANN's security environment.

Integrity engine checks integrity in three steps. First, it asks crypto engine, which performs all cryptographic calculations, to decrypt the integrity token, in this case a MAC value; crypto engine needs to get appropriate decryption key from ANN's security environment. This decryption process returns a hash of the packet as it was seen by its sender.

Second, it asks crypto engine to calculate hash of the packet. The last step is to compare this hash against the decrypted token. If they are equal, then integrity of the packet can be assumed. If these two values differ, however, then integrity check has failed.

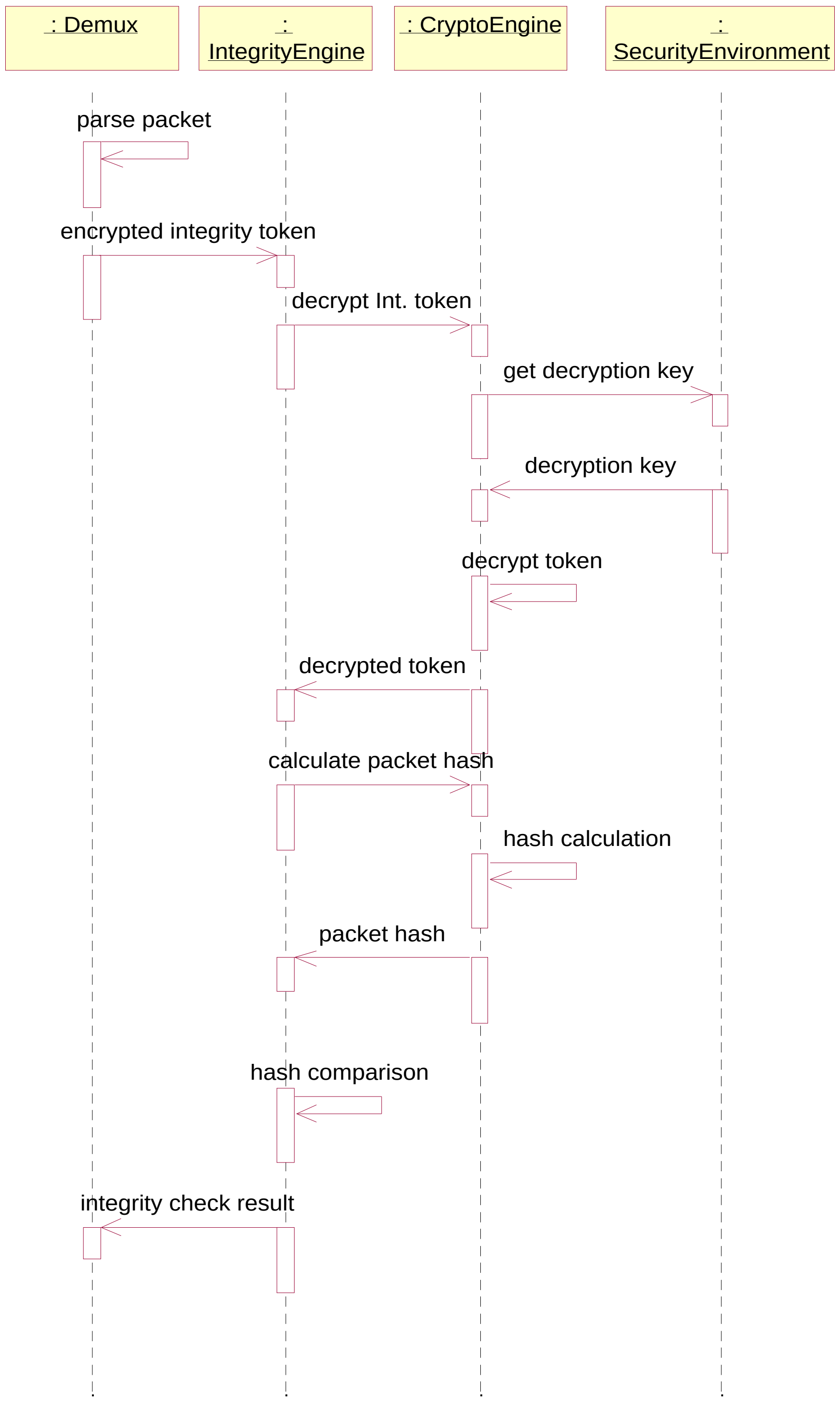


Figure 6-32: Packet integrity sequence chart

6.2.5.4.1.2 Active code integrity and code verification

Figure 6-33 depicts a sequence of events, which are required to check integrity of the newly received active code. Note that integrity checks for active packet and active code need to be separate because of the fact that these protections are in most cases provided by different encryption keys, i.e. different actors. The reason for this is that active code can be tampered with even before it is included in any active packet. Thus, digital signature generated by packet source at packet creation does not suffice for active code.

Every active code is accompanied by at least one special token, which is used for integrity protection (IntegrToken in Figure 6-33). This token has a form of a digital signature and/or a MAC (Message Authentication Code).

It is envisaged that both approaches to integrity will be used in FAIN. Digital signatures by code provider/manufacturer can provide integrity protection until code is injected into the active network. From there per-hop MAC protection can be used, which is expected to yield performance gains. Additionally, the advantage of per-hop MAC protection is that it covers both packet and code at the same time. Note that we omit the discussion of multi-domain issues in the initial phase.

Figure 6-33 depicts the case, when active code integrity is provided with digital signature generated by code provider. In this case integrity engine must first process code provider’s public key certificate in order to extract and validate providers public key.

After extracting a valid public key integrity engine checks integrity in three steps, similar to active packets. First, it asks crypto engine to decrypt the integrity token, in this case a digital signature. This decryption process returns a hash of the code as it was seen by the code provider.

Second, it asks crypto engine to calculate hash of the code. The last step is to compare this hash against the decrypted token. If they are equal, then integrity of the code can be assumed. If these two values differ, however, then integrity check has failed.

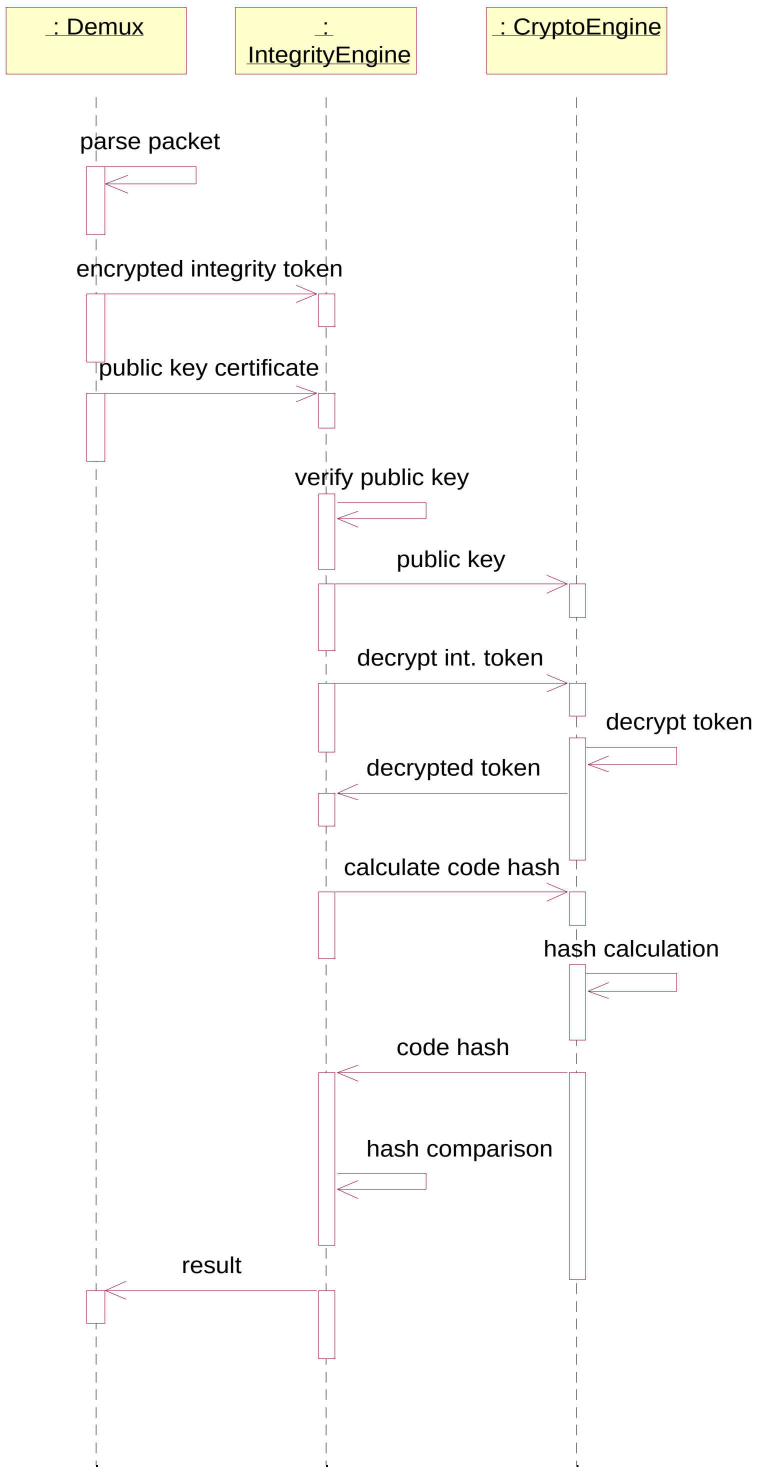


Figure 6-33: Active code integrity sequence chart

The majority of active code verification techniques are specific to particular EE. Since we have limited our current scope to NodeOS only, we do not address these mechanisms. The only general verification mechanism, which can be placed in the NodeOS is based on digital signatures by trusted parties.

For the initial FAIN security architecture we suggest that these trusted parties be active code providers. This effectively eliminates the need for distinct code verification process within the NodeOS, since code provider's digital signature is checked as part of code integrity check.

6.2.5.4.2 Execution-level security protections

Once an active packet has successfully passed entry-level checks, active code(s) can execute and perform operations within an ANN on behalf of some user. Obviously, some users will have more privileges than others, i.e. security policies define who can do what in an ANN. In order to protect an ANN it is necessary to prevent users from abusing their privileges and violating security policies.

According to the previous paragraph, execution-level protection includes two steps:

- Evaluating every execution request against node security policies, which is performed by authentication engine (see Figure 6-30) and
- Allowing or denying execution based on positive or negative authorization decision, respectively; policy enforcement engines are responsible for this (Figure 6-30)

6.2.5.4.2.1 Policy enforcement

Figure 6-34 depicts a sequence of events that take place whenever a request to perform certain action is received from an EE at the ANN subsystem interface (we chose RCF interface for illustration). Crucial to policy enforcement is the subsystem specific enforcement engine, which is implemented as an adaptation layer mediating requests at subsystem interface (see Figure 6-30).

Every request at RCF interface is intercepted and suspended by this adaptation layer (RCF enforcement engine in Figure 6-34). Before execution a request has to be evaluated against local security policies. Enforcement engine does not itself evaluate, whether this request is compliant with local security policies. Instead it invokes the authorization procedure within the security subsystem and feeds it with request information, such as: requested action, name of target object and requesting subject ID.

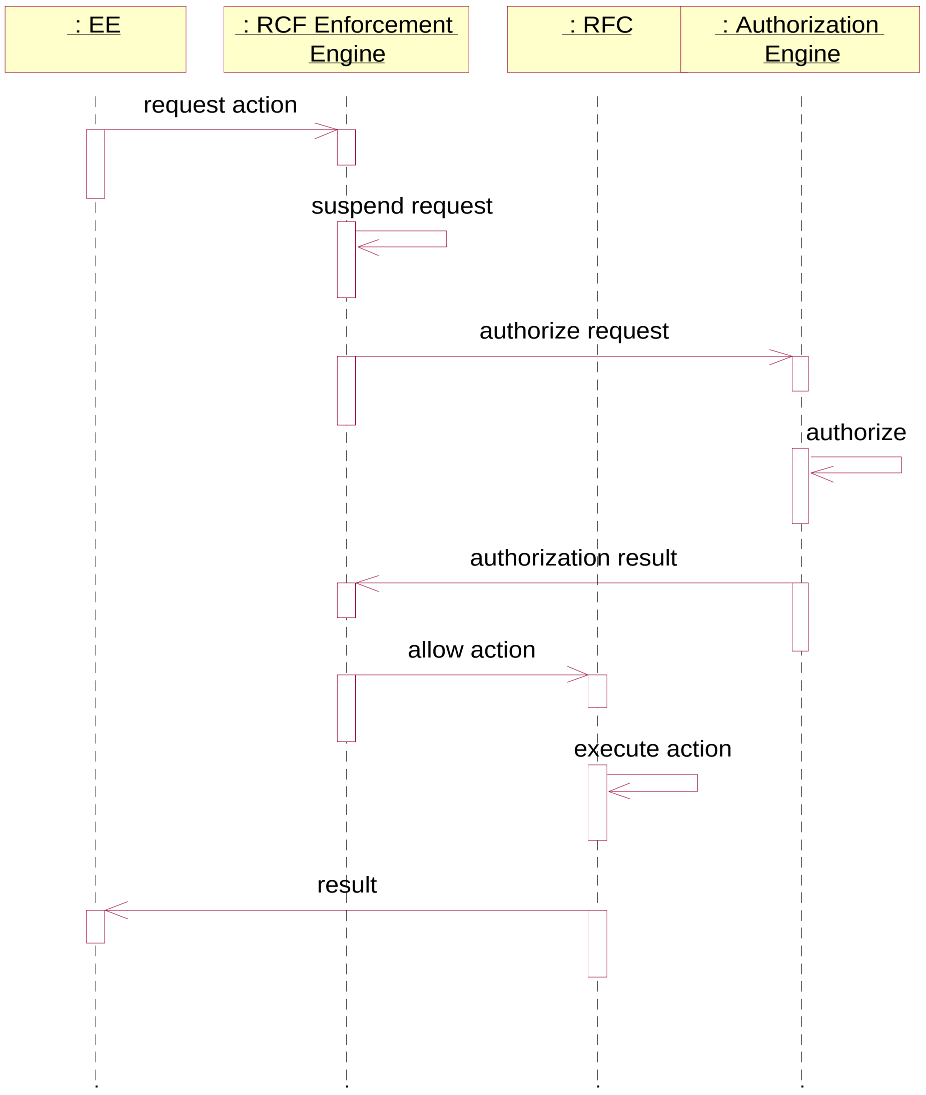


Figure 6-34: Policy enforcement sequence chart

Only after authorization returns, “request authorized” does an enforcement engine allow execution of the request by the underlying subsystem (RCF in Figure 6-34). Obviously, if authorization returns a negative answer, i.e. “request not authorized”, then enforcement engine simply discards the suspended request. This way, it prevents execution of unauthorized requests and essentially enforces users to adhere to local security policies.

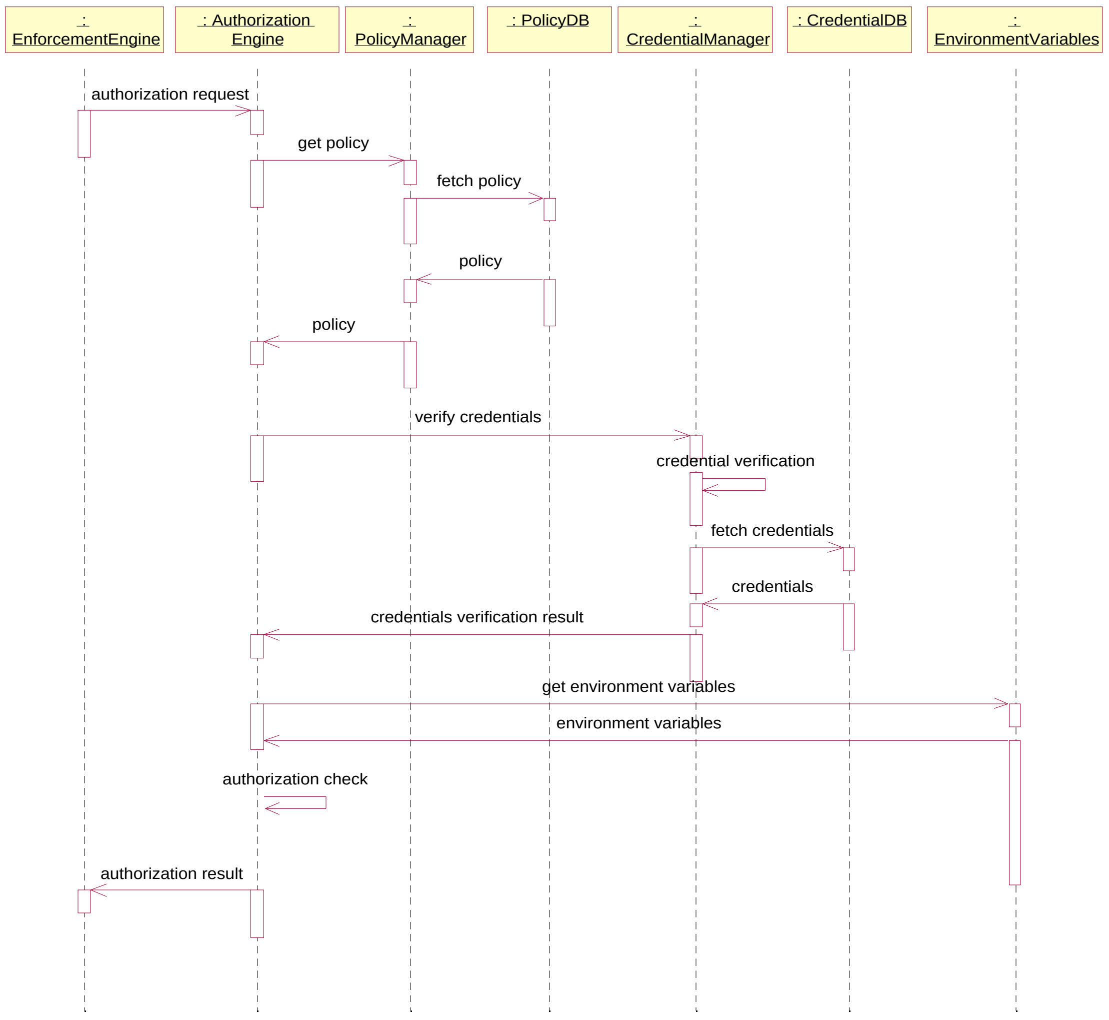


Figure 6-35: Authorization sequence chart

6.2.5.4.2.2 Authorization

Figure 6-35 depicts a sequence of events that are required to authorize the request, i.e. to decide whether to grant it or not. In flexible access control systems, authorization is not integrated with enforcement. Instead it is separated logically and in implementation. In this way, a single authorization engine can be used by multiple policy enforcement engines.

Authorization decision is based on the following set of data:

- request information (action, object name, subject ID)
- local security policies, which govern the way in which particular object can be used
- credentials associated with particular subject ID
- current values of environment variables, such as time of day and amount of resources used by subject

Enforcement engine provides the request information when it asks for authorization decision. This information is used as an “index” by policy and credential managers for fetching appropriate policies and credentials, respectively. Environment variables are provided to authorization engine upon request by facilities, such as system clock and resource monitoring module within RCF.

Finally, after gathering all the required information, authorization engine processes this data according to its internal rules, which return a simple result, either saying, “request authorized” or “ request not authorized.” This is returned to the calling policy enforcement engine, which then acts accordingly.

6.2.6 Relation to existing work

FAIN aims to develop a heterogeneous ANN, allowing coexistence of various technologies that enable installation and execution of active code within an ANN. Consequently, FAIN security architecture is aimed at providing a more general solution which provides necessary protections for such an heterogeneous system.

This is reflected by the fact that security architecture we have presented does not incorporate details of specific EEs that exist in the FAIN ANN. Its goal is to be as EE independent as possible and provide a common set of basic security services required by all AN enabling technologies.

Some research projects on active networks have already tried to tackle the issue of security [28][37][38][39]. Contrary to FAIN, all these approaches are tied to specifics of particular model of programmability. When designing a more general AN security architecture, which is the case in FAIN, these specifics can not be assumed. Java Security Architecture [42] proved to be useful for AN security, but again it is technology specific and it also has some drawbacks [38]. There has also been some more general work on AN security [40][41], however this work is basically a draft of main ideas on AN security and it is unclear whether this work has progressed since.

6.2.7 Implementation issues

It is beneficial to leverage existing technology as much as possible. In this section we present some of the existing software security toolkits that can be used in the implementation phase of initial FAIN security architecture.

6.2.7.1OpenSSL

SSL and TLS protocols can provide secure communication between two applications. OpenSSL [35] is a public domain implementation of these protocols whose core consists of a general purpose cryptographic library and an SSL library implementing protocol itself. In FAIN these can be reused “as is” to a large extent for providing the desired functionality of many components of the FAIN security architecture. For instance, most of the components responsible for entry-level security checks can be implemented by OpenSSL.

However, implementing a specific security protocol (SSL/TLS), OpenSSL has some drawbacks with regards to FAIN security. These will require adapting OpenSSL or amending it with the development of additional components. A non-exhaustive list of issues that need to be addressed regarding integration of (parts of) OpenSSL in the implementation phase of FAIN security architecture includes:

- SSL/TLS operates on top of TCP, while FAIN security may require that security critical data, such as digital signatures and credentials, be available at an earlier stage in packet processing path,
- SSL/TLS is an end-to-end protocol and consequently can use two way communication for authentication (i.e. a handshake protocol); in FAIN authentication must be based on unidirectional communication,
- SSL/TLS is a session-oriented protocol, which means that separate encryption key is set-up for each session. This does not fit nicely with the notion of per-hop integrity protections, where the same key is used for all traffic between neighbouring nodes

6.2.7.2KeyNote

KeyNote [36] was designed and developed to “help applications answer questions of the form ‘does this potentially dangerous action conform to my security policy?’ It provides applications with a standard interface for getting answers to such questions.”

This is exactly what authorization engine in FAIN security architecture has to do, so KeyNote fits the role of authorization engine. The KeyNote Toolkit is a publicly available implementation of a KeyNote trust management system and can be used in the FAIN implementation phase.

However, the most notable limitation of KeyNote is that it precludes the use of Certificate Revocation Lists (CRLs). While we do not consider this to be a serious drawback in the initial phase, we do plan to evaluate just how serious this limitation is in the FAIN context.

6.2.7.3Policy enforcement

In FAIN security architecture policy enforcement is performed within each ANN subsystem by a software “layer” sitting on top of the subsystem interface and mediating access to subsystem functionality and resources by VEs, EEs and active code. Thus, each ANN subsystem has its own mediating layer called enforcement engine. Consequently, while we consider these enforcement engines as part of the security architecture, they are delegated for implementation to respective ANN subsystem.

6.3 Demultiplexing Design

The description is provided in a separate internal report, [45], due to its confidential nature.

7 ACTIVE NODE IMPLEMENTATION

The previous section has described engineering components, which will make it possible for virtual environments to be isolated from one another. This part of the active node design develops a set of programmatic interfaces that enforces an information model on service providers, which will allow a node operator to enforce policies for each service the node will provide.

Before describing the information models, more technical issues have to be discussed.

7.1.1 Technical Issues

7.1.1.1Operating Systems Designs

Execution environment will appear as:

- Network interfaces which will run in a privileged mode
- Processes that intercept packets and do not run in a privileged mode.

A privileged mode is kernel space in Unix, Linux and BSD operating systems, while the converse is user space. In user space, execution environments will be user processes and they will only be able to receive and send packets using sockets and the `select()` method. It will be possible to coarsely constrain the resources an execution environment as part of a VE uses using the host operating systems methods of access control and resource allocation.

This is not the case for execution environments in kernel space. The only resource control is by means of the limits set in the build of the kernel. Some real-time operating systems are able to constrain the resources used by network interfaces (and other devices) by attaching tasks to each device. These tasks are monitored in the same way user space processes are in conventional operating systems.

7.1.1.2Implementations

Execution environments may be:

- Hardware devices interacting with the host active node using interrupt signals and DMA requests.
- Software devices registering themselves with their host as interrupt handlers
- Software processes registering themselves using sockets and programming software devices to send data to them.

Software devices will be implemented as kernel modules. Software processes will be implemented in two ways:

- Standalone daemon process
- ORB-connected daemon process[9]

The implementation of which will be in one of three modes:

- Runtime executable
- Interpreted object code in a virtual machine[10]
- Interpreted instruction code in an interpreter[11]

Some user process will be responsible for the set of execution environments it loads and activates. One of the goals of system design has been to provide a means of loading and activating execution environments that can be used by all of the different types of implementations.

7.1.2 System Components

There are three system components, which a service provider must contain, in his virtual environment within the active node:

- Resource control
- Packet demultiplexing control
- Execution environment loading

It will be seen that activation of the execution environment can be made dependent upon the activation of the packet demultiplexing scheme. The system design will be presented as a suite of interface definition language, IDL [54], files. These make use of pre-defined services and datatypes from the OMG as part of CORBA. These Common Object Services, COS, from CORBA are described first.

7.1.3 Common Object Services

An important point to note is that it is not necessary to engineer an active node system with an ORB to make use of the definitions of the common object services that are defined with most ORBs. IDL compilers provide standard datatypes and language bindings and these stubs can be used to implement standard ORB services if no ORB is available or it is not a complete implementation.

The implementation need not provide a complete service: just enough to complete the architecture. Using IDL and the standard CORBA services makes it easier to upgrade the implementation at a later date. Two services are used:

- `CosCollection`

[9] An Object Request Broker, [OMG98]

[10] Typically a Java Virtual Machine in user space

[11] There are execution environments which process simple packets in kernel space, SANE [Unk01]

- `CosLicensingManager`

Data types are used from two other services:

- `ResourceAccessDecision`
- `TimeBase`

`CosCollection` is used to provide a container type for the packet demultiplexing rules and for resource control names. `CosLicensingManager` is a simple interface for authorising objects. `TimeBase` simply provides a uniform datatype for seconds.

The use of `ResourceAccessDecision` does need some explanation. The design of this service requires that all resources be known by a name, that there is a sequence of operation names for the resource and that credentials be presented with each resource access request. The `ResourceAccessDecision` service has its own management infrastructure, which is used to create an association between a client, a resource, its resource access manager and the client's credentials. This object interaction is not given in this system architecture. It is only necessary to refer to resources by name and to hold credentials to use them. Resources are known by `ResourceName` structures, which contain a set of `ResourceNameComponent` entities and a naming authority for the resource.

7.1.4 Resource Control Frameworks

There are two sets of resources: the packet demultiplexing devices and the processing rights. These are dealt with in modules `Limit` and `Dmux`.

7.1.4.1Operating System Resource Limits Limit in `limit.idl`

This module starts with some definitions of null interfaces that are used to type languages for user resource limits. It then defines a set of names for resources that would be used in controlling the resources used by a user process under the Unix operating system.

A structure `LimitResource` is introduced which contains one resource name and one structure for current and maximum limits. `LimitResources` is a `CosCollection` that is designed to contain a set of `LimitResource`. `LimitResources` is given two methods to control the resources: check if active and make active.

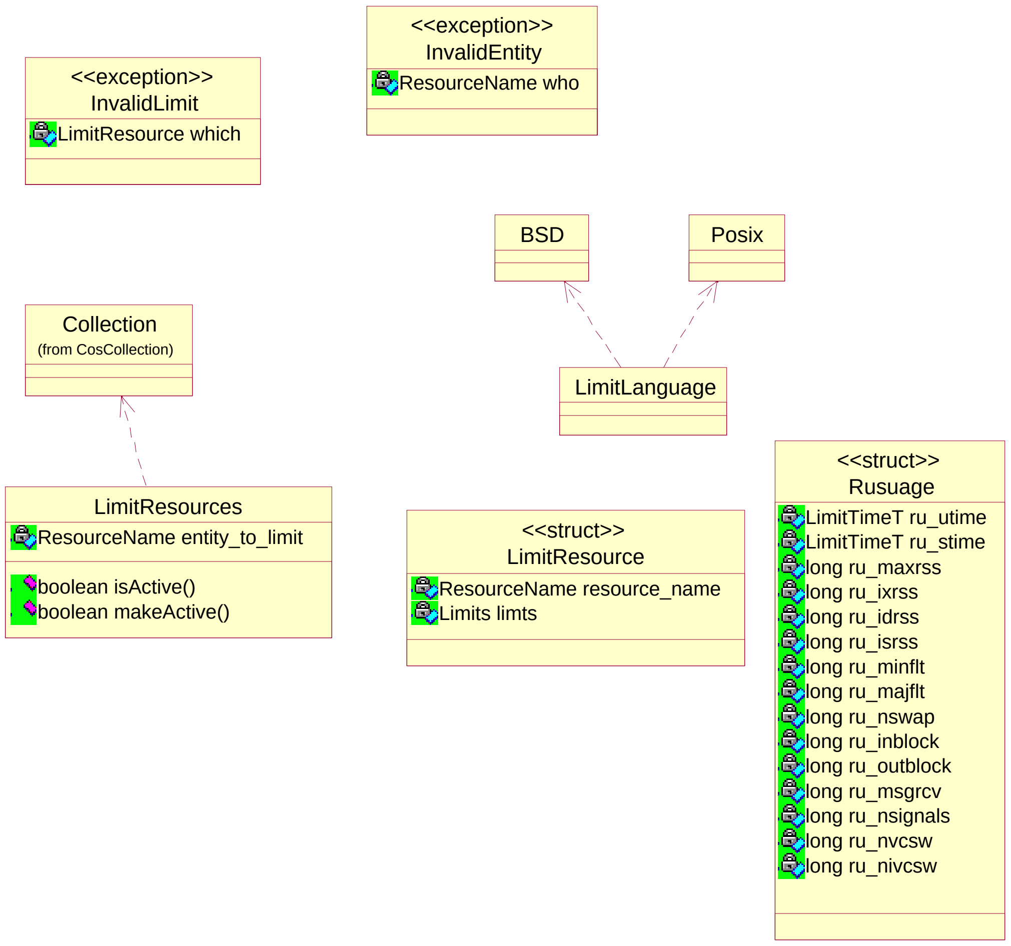


Figure 7-1 LimitResources Information Model

`LimitResources` is an extension of `CosCollection`. One operation is added `makeActive()`. This would ideally be a two-phase commit or abort operation using semaphores. This is a basic type; it is aggregated into another component, which is discussed later.

A detailed specification of the `limit.idl` is given in Appendix D.1

7.1.4.2Packet Demultiplexing Dmux in `dmux.idl`

This module is very similar in design to Limit, but is slightly more complicated because of the underlying data structure. Berkeley packet filtering rules are trees, whilst resource limits are tables. The data structure is recursively defined:

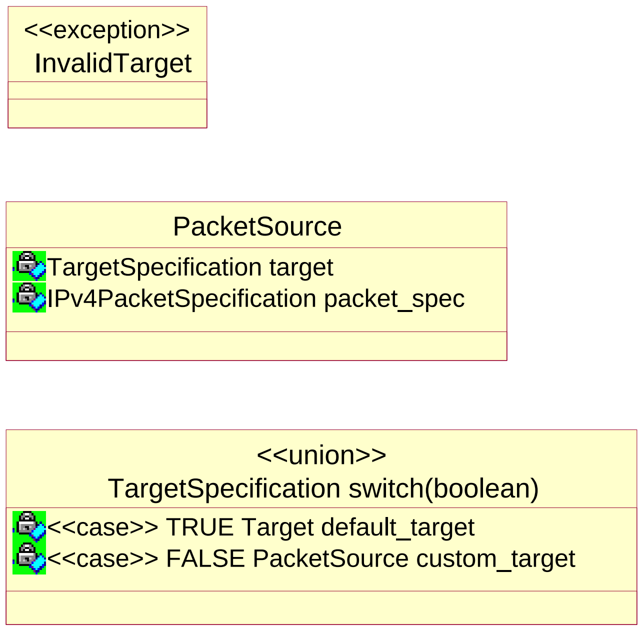

<<exception>>
InvalidTarget
PacketSource
TargetSpecification target
IPv4PacketSpecification packet_spec
<<union>>
TargetSpecification switch(boolean)
<<case>> TRUE Target default_target
<<case>> FALSE PacketSource custom_target

Figure 7-2 PacketSource Information Model

```
interface PacketSource;
   union TargetSpecification switch(boolean) {
     case TRUE: Target default_target;
     case FALSE: PacketSource custom_target;
};
// This should really be a struct and it should be forward
interface PacketSource {
   attribute TargetSpecification target;
   attribute IPv4PacketSpecification packet_spec;
};
```

IPv4PacketSpecification is the definition of a packet-trapping rule. The `TargetSpecification` is a controlled resource that will be one of the three packet buffer queues - input, forward or output. Contained within the definition of IPv4PacketSpecification are the two resources that can be acquired by an execution environment: a port redirect, `redirpt`, and an interface `vianame`. For each `PacketSource` specified, a virtual environment loading the execution environment would have to acquire the packet buffer and a port or an interface.

Detailed specification of the `dmux.idl` may be found in Appendix D.2

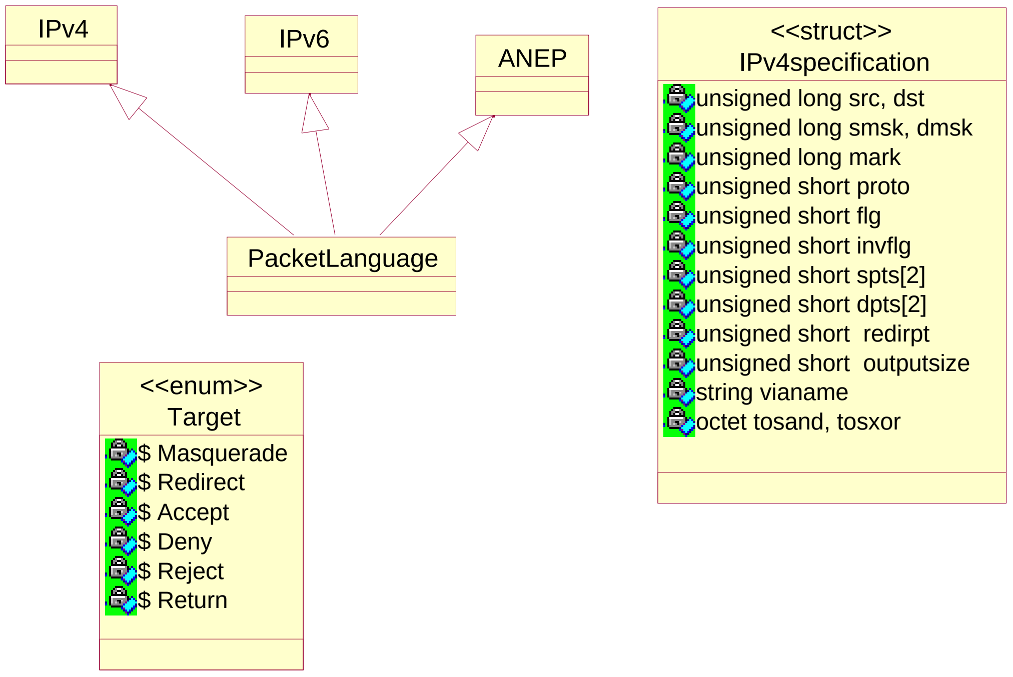


Figure 7-3 PacketLanguage Information Model

The packet language information is aggregated into another structure.

7.1.4.3Resource Management RCF in `rcf.idl`

The Rcf module contains two management interfaces:

- `ResourceManager`
- `ResourceProfileFactory`

Each virtual environment and then each execution environment must obtain resources from its ResourceManager. The resources are contained in a `ResourceProfile`, which is created by a ResourceProfileFactory.

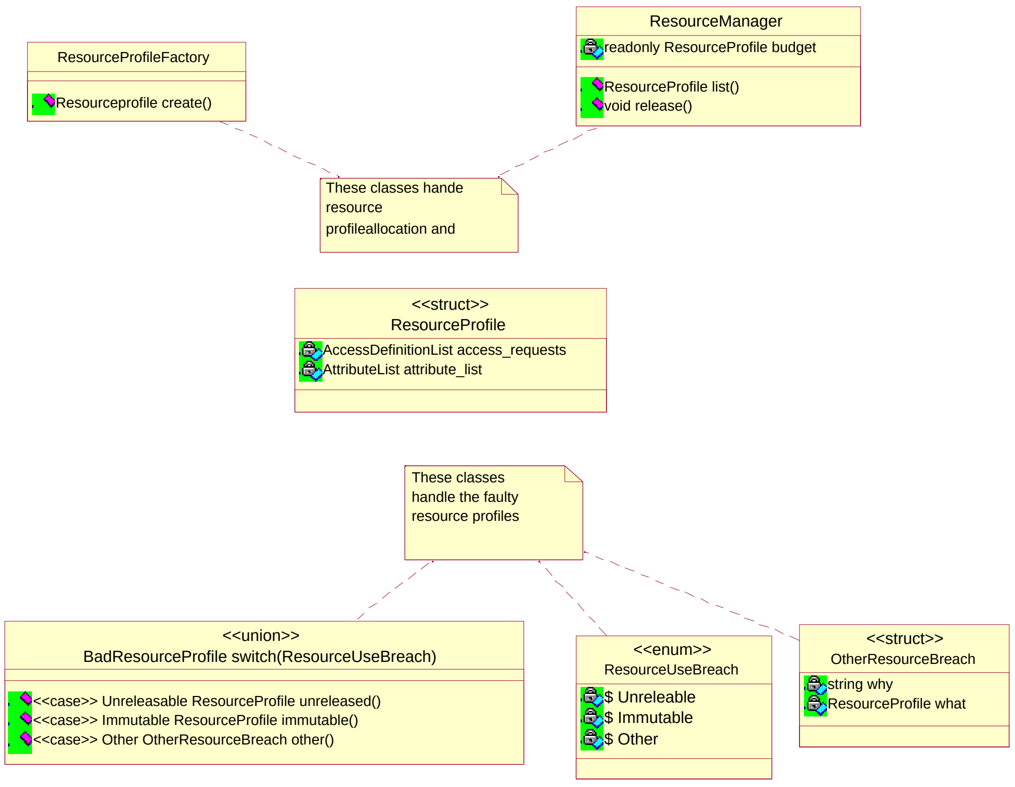

ResourceProfileFactory
Resourceprofile create()
ResourceManager
readonly ResourceProfile budget
ResourceProfile list()
void release()
These classes hande resource profileallocation and
<<struct>> ResourceProfile
AccessDefinitionList access_requests
AttributeList attribute_list
These classes handle the faulty resource profiles
<<union>> BadResourceProfile switch(ResourceUseBreach)
<<case>> Unreleasable ResourceProfile unreleased()
<<case>> Immutable ResourceProfile immutable()
<<case>> Other OtherResourceBreach other()
<<enum>> ResourceUseBreach
$ Unreleable
$ Immutable
$ Other
<<struct>> OtherResourceBreach
string why
ResourceProfile what

Figure 7-4 ResourceProfile Information Model

Additional interfaces in RCF are for obtaining `ResourceNamingAuthority` entities, Figure 7-1 LimitResources Information Model. In a fully-connected ORB system these would be obtained on start-up from the `CosNaming` service. But, because these resources are so closely tied to the operating system, and that a full ORB might not be available, the `ResourceNamer` interfaces and their methods provide a simple means of obtaining the right name for each resource.

Detailed specification of the `rcf.idl` is provided in Appendix D.3

7.1.5 Licensing Safe in `safe.idl`

The module that provides an interface by which virtual environments and execution environments can gain authority to acquire resources is the `Licenser` interface in the Safe module. This is simply the `CosLicensingManager` interface.

The module Safe is the least developed of all of these modules. It should be used to contain other internal interfaces for authorising code. It should be a derivation of the Security services architecture developed within CORBA [53].

The information model for the `Licenser` component is so simple, it does not merit a class diagram.

A detailed specification of the `safe.idl` is provided in Appendix D.4

7.1.6 Environment management in ActiveNode in `fain.idl`

7.1.6.1Licensing and Resource Management

These are put into a sub-type of the `SuperStructure` container. Should there be a need for other types of manager to be given to virtual environments and execution environments the `SuperStructure` type can be further sub-typed.

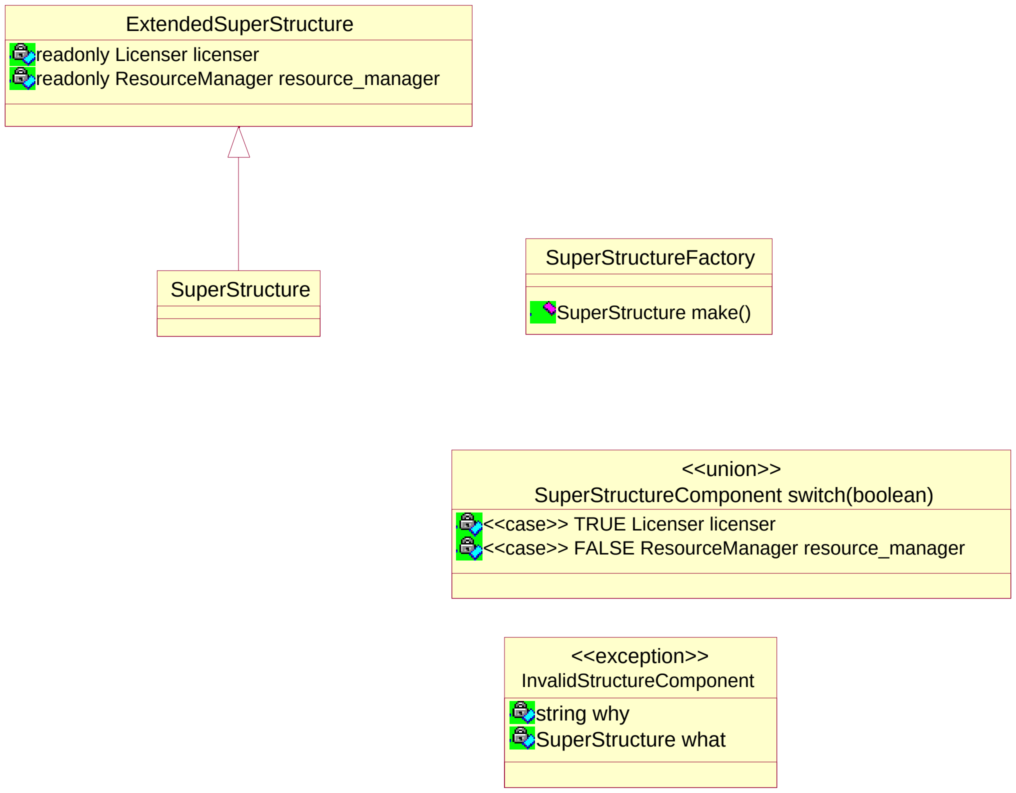

ExtendedSuperStructure
readonly Licenser licenser
readonly ResourceManager resource_manager
SuperStructure
SuperStructureFactory
SuperStructure make()
<<union>>
SuperStructureComponent switch(boolean)
<<case>> TRUE Licenser licenser
<<case>> FALSE ResourceManager resource_manager
<<exception>>
InvalidStructureComponent
string why
SuperStructure what

Figure 7-5 SuperStructure Information Model

7.1.6.2Execution environments

The base class is EE. This is sub-typed twice:

- To `UnmanageableEE`, which is intended to contain kernel modules, which cannot have their resources managed. This should be a pseudo-interface, because it will not support any of the standard ORB object operations.
- To `ManageableEE`, which would contain user processes that can be ontrolled by a resource manager and would support basic ORB object operations.

`ManageableEE` is then sub-typed to VE, which provides a method to create EE, which, because of the sub-typing, can be any of `ManageableEE` or `UnmanageableEE` or VE. Provision is made for a supervisor virtual environment with the attribute privileged. Virtual environments may create execution environments using the make method. This requires both a `CodeBase` and a `SuperStructure` entity. Execution environments are associated with a `PacketSources` entity. When activated, an execution environment will provide the redirect or via name entities specified in the `PacketSources` entity.

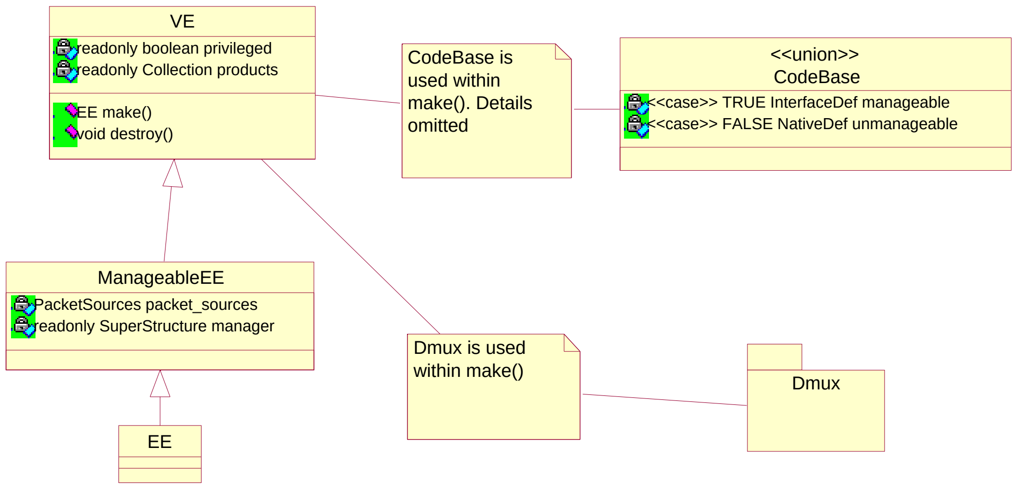

VE
readonly boolean privileged
readonly Collection products
EE make()
void destroy()
CodeBase is used within make(). Details omitted
<<union>>
CodeBase
<<case>> TRUE InterfaceDef manageable
<<case>> FALSE NativeDef unmanageable
ManageableEE
PacketSources packet_sources
readonly SuperStructure manager
EE
Dmux is used within make()
Dmux

Figure 7-6 Virtual and Execution Environment construction

Detailed specification of the `fain.idl` is provided in Appendix D.5

7.1.7 Open Distributed Processing Reference Model

The prescriptive model for open distributed processing, ODP, systems [52] was the basis for CORBA. The ODP Node entity is responsible for creating Capsule entities. Each Capsule has a factory interface for generating objects, which would have an interface accessible outside of the Capsule for taking part in distributed computations and a number of local engineering interfaces, which are used within the Capsule and by the host Node to manage the cluster of engineering objects that the computational object uses.

It is expected that the engineering of a Node is part of a computational system and the engineering objects have administrative interfaces in it. The computational objects in this FAIN system present the `LicensingManager`, `ResourceManager` and `ManageableEE` interfaces.

7.1.8 Engineering Issues

In a practical Unix-based system, an ODP Node is the host computer and the Capsule entities are the user processes. Virtual environments are competing for resources within the kernel: there can be only one port on the host having number 3325, there can be only one network interface called `ipsec0`: because of this it may be easier to engineer the system so that there is a one-to-one mapping between Node and Capsule. Resource management need only take place within one Capsule. This would reduce the need for an ORB, since all object services can be locally linked and located.

7.1.9 Outstanding Architectural Issues

This system design is probably as complete as it needs to be. The outstanding issues are extending the modules to add different kinds of resource sets.

7.1.10 Node Resources

7.1.10.1 BSD `sysctl` interface

The BSD `sysctl` interface does not operate in the way the comments in Appendices B and C would lead one to believe. This is not a limitation. The FAIN project is expecting to implement its own resource allocation facilities and the mapping between resource names in a naming authority and the operating system can be reconciled. It would be desirable if the BSD system resources could all be named and made accessible through the `sysctl` interface. It would then only be necessary to implement a naming service that just has the resource allocation facility as its domain.

7.1.10.2 More Extensive Node Resource Management

The only set of resources put under a management scheme in x 3.2.1 has been the operating system resources available to processes (and process groups and users). Java allows objects running on threads within virtual machines to be assigned a sub-class of `SecurityManager`, which will operate according to some set of policies. These policies can limit access to local file space and to network resources. It would be possible, for example, to restrict access to the host's ports for each instance of a class. Real-time operating systems allow drivers for physical devices to be put under resource control. It would then be possible to restrict access to buffers on different network devices. There is an immediate need to extend the resource control management scheme for Java objects.

7.1.10.3 Packet Demultiplexing

The packet demultiplexing scheme given in x B is based on the Berkeley packet-filtering scheme. Recently, Linux operating systems have begun to deploy a variant of the Berkeley scheme that allows some state to be stored with each packet classification. IPv6 promises to make active networking more useful still. The ability to embed arbitrary headers within IP packets will make it easier to carry code and credentials. It can be expected that packet demultiplexing for IPv6 will allow processing based on the presence and correct decoding of headers. If interoperation with `anetd` in Abone or ANON is required, then the ANEP packet demultiplexing scheme would need to be implemented as well.

7.1.10.4 Safe Computing

Currently only a `LicensingManager` has been prescribed in this module. Execution environments will need to check each code packet they receive. For kernel-based packet processors, this will not be able to make use of any ORB-based services, but user space processors could make use of other services in the `Security.idl`.

7.1.10.5 Code Bases

The code for an execution environment is specified using the `Fain::CodeBase`. This uses the definitions `InterfaceDef` and `NativeDef` to specify the code to be used. Code is also a resource. A `NativeDef` would be kernel module that could be loaded to provide a transcoder, for compressed and streamed MPEG, or encrypting device, part of an IPSec implementation. It will probably be necessary to provide a `ResourceNamingAuthority` for `Fain::CodeBase` entities. This will probably need nothing than a directory map with access control list. A module name and a credential to use the special operation load would install a kernel module with the privileges of the superuser.

8 CONCLUSIONS AND FUTURE WORK

D2 describes the output achieved in WP3 as part of the first stage (1st year) of FAIN project. In this deliverable we have laid out the foundations of FAIN system captured by the reference architecture. VEs are the abstractions based on which resource control is provided by the node. They also act as containers of services that are executed in well defined EEs. Multiple EEs may also be possible in a VE.

We have identified and produced the first detailed designs of a number of components that deal with resource control, security and demultiplexing issues, which are necessary for the realisation of the reference architecture. For each one of them we worked out operation scenarios in the form of use cases to demonstrate their operation. Based on these designs we have produced a number of programmatic interfaces as a means to point towards the AN node implementation.

Due to the demanding nature of building an AN testbed and consequently an AN node that supports a modular architecture, we have identified a number of possible system architectures with the most probable being the one where a VE is composed of a control plane EE and a high performance EE that collaborate in order to implement a network service. Further investigation is under way for establishing the exact design of such engineering option.

Insofar, we have worked in a vertical manner by giving emphasis on the design of the individual components and how these may function internally. We pay little or no attention to the interactions between different components that eventually define and dictate the operation of the AN node as a whole. The latter is also necessary to guarantee the integration of this functionality and eliminate any inconsistencies that may appear across different components.

This is the subject of the next stage in FAIN regarding WP3. We expect to achieve this by creating a number of scenarios in the form of use cases that depict operation across different components. In addition, we will enhance and extend the initial set of interface specifications (IDL specifications) found in this document that provide more clarity and the basis for an integrated implementation. Finally, we will continue to refine the FAIN node architecture using as input results from the other workpackages.

9 REFERENCES

[1] ITU-T Recommendation Q.1201 “Principles of intelligent network architecture”-1992; Recommendation O.1224 “Distributed functional plane for intelligent networks-CS-2” –1997; Recommendation Q.1225 “Physical plane for intelligent network CS-2”-1997; Recommendation Q.1211 “introduction to intelligent network CS-I”- 1993; Recommendation Q.1229 “Intelligent network user’s guide for capability set 2”- 1997

[2] Bell Communications Research Inc., “AIN Release 1 Service Logic Program Framework Generic Requirements”, FA-NWT-001132

[3] DARPA Active Network Program, http://www.darpa.mil/ito/research/anets/projects.html, 1996

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

[5] FAIN R1 Report- “FAIN Enterprise Model”- WP2-DT-006-I99.doc

[6] RFC 2475, “An Architecture for Differentiated Services”, 1998

[7] A.A. Lazar, K-S. Lim, and F. Marconcini, “Binding Model: Motivation and Description”, 1995

[8] Biswas, J., et al., “The IEEE P1520 Standards Initiative for Programmable Network Interfaces”, IEEE Communications, Special Issue on Programmable Networks, Vol. 36, No 10, October, 1998. http://www.ieee-pin.org/

[9] Biswas et al., “Proposal for IP L-interface Architecture”, IEEE P1520.3, P1520/TS/IP013, 2000

[10] Wetherall, D. J., J. V. Guttag, and D. L. Tennenhouse, “ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols”, IEEE Openarch, April 1998.

[11] Decasper, D., G. Parulkar, S. Choi, J. DeHart, T. Wolf, B. Plattner, "A Scalable, High Performance Active Network Node", IEEE Network, January/February 1999.

[12] Schwartz, B., A. W. Jackson, W. T. Strayer, W. Zhou, D. Rockwell, and C. Partridge, “Smart Packets for Active Networks”, OPENARCH'99, March 1999

[13] Alexander, D. S., W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith, “The Switchware Active Network Architecture”, *IEEE Network Special Issue on Active and Controllable Networks*, vol. 12 no. 3, May/June 1998.

[14] Calvert, K. L., ed. “Architectural Framework for Active Networks”, Version 1.0, Active Network Working Group, July 1999

[15] AN Node OS Working Group, “NodeOS Interface Specification”, January 2000

[16] Calvert K. L., S. Bhattacharjee, E. Zegura, and J. Sterbenz, “Directions in Active Networks”, IEEE Communications Magazine, October 1998.

[17] Hicks, M., et al., “Experiences with Capsule-based Active Networking”, Tech. Report, University of Pennsylvania, 2000

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

[19] Smith, J. M., et al., “Activating Networks: A Progress Report”, IEEE Computer, 1999.

[20] Berson, S., et al. “Introduction to the Abone”, 2000

[21] Van der Merwe, J.E., S. Rooney, I.M. Leslie, and S.A. Crosby, “The Tempest: A Practical Framework for Network Programmability”, IEEE Network, Vol. 12, No. 3, pp. 20-28, May/June 1998

[22] Campbell, A. T., H. De Meer, M. E. Kounavis, K. Miki, J. Vicente, and D. Villela, "The Genesis Kernel: A Virtual Network Operating System for Spawning Network Architectures", 2nd IEEE International Conference on Open Architectures and Network Programming (OPENARCH'99), New York March 1999

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

[24] Braden, B., A. Cerpa, T. Faber, B. Lindell, G. Phillips, and J. Kann, "ASP EE: An Active Execution Environment for Network Control Protocols", ISI Technical Report, December 1999.

[25] Denazis S. G., et al., "Designing Interfaces for Open Programmable Routers", IWAN'99, 1999.

[26] FAIN Project, D1, Deliverable 1, "Enterprise Model and AN requirements", May 2001

[27] FAIN WWW Server – www.ist-fain.org

[28] D. S. Alexander, W.A. Arbough, A. D. Keromytis, and J. M. Smith. A secure active network environment architecture: Realisation in SwitchWare. IEEE Network, Special Issue: Active and Programmable Networks:37-45, May/June 1998.

[29] R. Shirey. RFC 2828: Internet Security Glossary. May 2000. Internet Engineering Task Force

[30] B. Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C. John Wiley and Sons, Inc., Second edition, 1996.

[31] G. Necula. Compiling with proofs. School of computer science, Carnegie Mellon University, Sep 1998, http://www.cs.cmu.edu/~necula/papers.html.

[32] G. McGraw and E.W. Felten. Securing JAVA: Getting Down to Business with Mobile Code. NY:John Wiley & Sons, Inc., 1999.

[33] J.T. Moore, M .Hicks, and S. Nettles. Practical Programmable Packets, INFOCOM 2001 proceedings. Apr 2001, http://www.cis.upenn.edu/~switchware/SNAP/.

[34] B. Schneier and J. Kelsey, Cryptographic Support for Secure Logs on Untrusted Machines, The 7th USENIX Security Symposium proceedings, USENIX Press, Jan 1998, pp. 53-62

[35] OpenSSL Home Page, http://www.openssl.org

[36] KeyNote, http://www.cis.upenn.edu/~angelos/keynote.html

[37] B. Braden, B. Lindell, and S. Bernson. A Proposed ABone Network Security Architecture. ABone draft, Nov 1999.

[38] Active Networks Working Group. SANTS Security Overview, May 2000.

[39] S. Schwab, R. Yee, and R. Dandekar. AMP Security Overview. Technical Report, NAI Labs, May 2000.

[40] Active Networks Security Working Group. Security Architecture for Active Networks, draft, Jul 1998.

[41] Seraphim Group. Security Architecture for Active Networks Modified by Seraphim Group, draft, May 2000.

[42] Li Gong. Java Security Architecture (JDK1.2). Technical Report, Sun Microsystems, Oct 1998.

[43] Y. Yemini et al. MarketNet: Protecting Access to Information Systems Through Financial Market Controls. To Appear in Decision Support Systems Journal.

[44] D. Reed et al. Xenoservers: Accounted Execution of Untrusted Code. Mar 1999, Seventh IEEE Workshop on Hot Topics in Operating Systems.

[45] WP3-HEL-026-D2-DeMUX-Int.doc, FAIN Internal Report R36, May 2001.

[46] *Gísli Hjálmt_sson and Samrat Bhattacharjee*, " Control on Demand - An Efficient Approach to Router Programmability," in IEEE JSAC, Vol. 17, No. 9, September 1999, pp. 1549-1562.

[47] Paul Menage. RCANE: a resource controlled framework for active network services. In Proceedings of the First International Working Conference on Active Networks -- IWAN '99, June 1999

[48] Active Networks NodeOS Working Group. NodeOS Interface Specification. January 10, 2001

[49] D. J. Wetherall, J. Guttag, and D. L. Tennenhouse. *ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols*. In Proceedings of IEEE Openarch'98, April 1998

[50] Steven Hand, "Self-Paging in the Nemesis Operating System**",** Usenix Third Symposium on Operating Systems Design and Implementation, February 1999

APPENDIX A: R23.1 – REVIEW & EVALUATION OF ACTIVE NETWORKS RESEARCH PROJECTS

A.1 Introduction

This appendix summarises the analysis of the main active networks research projects based on literature survey and review as follows: GENESYS, BANG, ANON, M0, JanOS, NETSCRIPT, Openet, Click Router, ANTS, CANES, TEMPEST, JanOS, SwitchWare, ABLE, ARP, DARWIN, ANN, EXOKERNEL, ABONE, ALAN and PRONTO projects.

The detailed analysis could be found at:

ANN review is described in WP3-ETH-005-R23-Int

EXOKERNEL review is described in WP3-ETH-003-R23-Int

ANON review is described in WP3-GMD-008-R23-Int

BANG review is described in WP3-GMD-004-R23-Int

M0 review is described in WP3-GMD-007-R23-Int

ANTS review is described in WP3-HEL-006-R23-Int.doc and WP3-HEL-010-R23-Int.doc

CANES review is described in WP3-HEL-006-R23-Int.doc and WP3-HEL-010-R23-Int.doc

OPENET review is described in WP3-HEL-009-R23-Int.doc

TEMPEST review is described in WP3-HEL-005-R23-Int.doc

JanOS review is described in WP3-GMD-003-R23-Int.doc and WP3-GMD-006-R23-Int.doc

CLICK ROUTER review is described in WP3-NTU-001-R23-Int.doc

NETSCRIPT review is described in WP3-JSIS-001-R23-Int.doc and WP3-JSIS-001-R23-Int.doc

PRONTO review is described in WP3-KPN-004-R23-Int.doc

ALAN review is described in WP3-NTUA-002-R23-Int.doc

GENESIS review is described in WP3-SAG-002-R23-Int.doc and WP3-SAG-008-R23-Int.doc

ABLE review is described in WP3-UCL-004-R23-Int.doc

ARP review is described in WP3-UCL-003-R23-Int

DARWIN review is described in WP3-UCL-003-R23-Int

ABONE review is described in WP3-UPC-003-R23-Int.doc

A.2 Scope of the Analysis

This document describes the results of a review of 19 research projects, which have originated in the Programmable and Active networks schools of thought. The Programmable/Active Networks differences are not profound as they lie with the degree of programmability they engineered into their prototypes.

1.These projects have being analysed on the following criteria:

2.Available Project References

3.Main strong point of the reviewed architecture

4.Main weak points that the reviewed architecture possesses

5.Are there any innovations?

6.Can any specific parts/ideas be adopted?

7.How can it be related to FAIN?

8.How easily can it be implemented

9.Is it worth being evaluated in R23.2?

The results of this analysis were used in the in depth review of a short list of AN platforms as follows: ANTS, ANN, BANG, JANOS, OPENET, ABLE and SWITCHWARE, which are part of the R23.2 report “ FAIN Analysis of Active Networks Platforms”.
In addition the review results were used as the basis of the development of the FAIN Active Network Reference Architecture, which is described in R23 Report.

A.3 Project ANN (ETH)

- This review is based on: WP3-ETH-005-R23-Int
- Main strong point of the reviewed architecture:
  - ANN creates a high-performance execution environment for transport-plane packet processing.
  - Basis for the SPCs (Smart Port Cards) of the WUGS (Washington University of St. Louis Gigabit Switch).
  - Runs on NetBSD.
  - Kernel-module plugin environment.
  - Distributed Code Caching.
- Are there any innovations: The IP stack was modified so that active packets can trigger the installation of a plugin into the execution environment. It further allows a cut-through path in case no-special packet processing is required.
- Can any specific parts/ideas be adopted: The plugin installation mechanism together with the Code Server and the distributed code caching.
- How easily can it be implemented: In case the ANN architecture was selected to be the platform for FAIN, the NetBSD OS would run on any legacy personal computer. Otherwise, the implementation of the ANN execution environment within another Unix OS requires a huge effort: the kernel as well as the IP stack required modifications.
- How can it be related to FAIN: ANN provides on of the most promising approaches for high-performance AN with the plugin approach. Therefore, it could serve as starting point for the implementation of the FAIN execution environment architecture (see next).
- Report on the main weak points that the reviewed architecture possesses:
  - Security is based only on signed code (kernel space plugin).
  - No separation of EEs is provided: only one exists per kernel.
- Is it worth being evaluated in R23.2: YES: the ANN should be evaluated as one of the most promising approaches within the high-performance AN research-field. Further, it was developed in a joint project between a Washington University of St. Louis and ETHZ.

A.4 Project ExoKERNEL (ETH)

- This review is based on: WP3-ETH-003-R23-Int
- Main strong point of the reviewed architecture:
    - The ExoKernel project focuses on the creation of a high-performance library operating systems (libOS).
    - It does so by providing a microkernel like approach that provides nothing more than the very basic functionality of an operating system.
    - The concept persued consists of the separation from management and protection; i.e. the decisions are taken within a libOS while the microkernel provides the lower-level configuration and set-up functionality.
    - Dynamic installation of libOS.
    - Tailoring of the libOS according to the needs of the targeted application.
- Are there any innovations: Conceptually, no: the microkernel approach is fairly well known. The implementation aspect clearly allows the dynamic installation of new libOS.
- Can any specific parts/ideas be adopted: Interfacing of the libOS to the microkernel (secure bindings).
- How easily can it be implemented: This can only be implemented if a new OS is to be developed.
- How can it be related to FAIN: In case we decide, a new OS is required, we clearly can profit from the work done in ExoKernel.
- Report on the main weak points that the reviewed architecture possesses: The research performed in ExoKernel focuses on other aspects than the AN research-area.
- Is it worth being evaluated in R23.2: NO. See above.

A.5 Project ANON (GMD)

- This review is based on: WP3-GMD-008-R23-Int
- Main strong points of the reviewed architecture:
  - Only other Active Network testbed system
  - Federated management style allows for decentralised and scalable overlay network
  - Allows dynamic membership, unlike ABONE
- Are there any innovations:
  - Decentralised management system
  - Dynamic membership
- Can any specific parts/ideas be adopted:
  - See above
- How easily can it be implemented:
  - Very easily, uses standard UNIX tools: Lynx, PGP, one hand-coded network daemon and the standard UNIX tools
- How can it be related to FAIN:
  - Promising architecture for our own testbed
- Report on the main weak points that the reviewed architecture possesses:
  - Further development needs to be done before it is suitable for wide scale deployment
- Is it worth being evaluated in R23.2: Maybe not in R23.2, but when design of the FAIN testbed is being discussed we should definitely look at ANON as a possible architecture

A.6 Project BANG (GMD)

- This review is based on:
  - WP3-GMD-004-R23-Int
- Main strong point of the reviewed architecture:
  - Flexibility by Interfaces and Programmability
    The (three) layered architecture and approach of BANG allows an easy change of components and service deployment. The interfaces among the layers are well-defined and potentially might be mapped directly to the architecture of FAIN.
    BANG supports different levels and models of programmability (capsule vs. code downloading). Frameworks for Code Distribution and Resource Control exist.
  - High-Performance
    BANG is based upon high-performance routers, which are configurable, but with fixed interface (part). The generic router interface is analogy to the L interface of IEEE P1520.
  - Security
    BANG has security support given by protection mechanisms provided by an agent environment in the upper-most active part/layer and by Java virtual machine.
- Are there any innovations:
  - The key innovations are the BANG architecture (frameworks and interfaces) and
  - Combination of flexibility (programmability) and high-performance.
- Can any specific parts/ideas be adopted:
  - BANG architecture could be considered/mapped to FAIN architecture
  - BANG frameworks (Code Distribution/Resource Control) can be used to ease integration of different EEs.
- How easily can it be implemented:
  - A sample implementation is available at GMD and potentially other FAIN partners.
  - Current implementation is based upon Hitachi Gigabit Router GR-2000. Its interface is well abstracted so that most parts of BANG could be (re-) used on nodes without a GR-2000.
- How can it be related to FAIN:
  - Could serve as a starting point for a FAIN active node
  - Architecture and Frameworks can be used partly in FAIN.
- Report on the main weak points that the reviewed architecture possesses:
  - The active node architecture is based upon a fixed part, which is not easily extensible. The well-defined interface to the fixed part allows simple change of it.
- Is it worth being evaluated in R23.2:
  - Yes, because BANG appears to be a promising candidate to cover a number of important and critical criteria for FAIN. It meets in particular well high-performance and flexibility together with control and management due to its architecture and frameworks approach.
- Any other remarks: Scalability of the BANG approach could be studied in FAIN.

A.7 project M0 (GMD)

- This review is based on: WP3-GMD-007-R23-Int
- Main strong points of the reviewed architecture:
    - Extremely minimal and purist Mobile Code system, allows for fast and efficient Mobile Code applications
    - Flexible due to the ability to create and reprogram Messengers on-the-fly (meta-programming)
    - Applications have already been implemented (see section 4)
- Are there any innovations:
    - Pure instruction-based communication model
- Can any specific parts/ideas be adopted:
    - Minimal design of EE, implement dynamic services on top of fast and efficient EE
- How easily can it be implemented:
    - Can easily be implemented, EE is pure C, no Java VM etc. needed.
- How can it be related to FAIN:
    - Lightweight and complete architecture can be utilised for the purposes of a demo
- Report on the main weak points that the reviewed architecture possesses:
    - Purist approach makes it difficult to integrate with other AN approaches
- Is it worth being evaluated in R23.2: Yes, as it represents an approach, which FAIN should investigate. (I.e., not just plugins, but also capsules)

A.8 Project ANTS (HEL)

- This review is based on: **WP3-HEL-006-R23-Int.doc / WP3-HEL-010-R23-Int.doc**
- Main strong point of the reviewed architecture:
    - Capsules are a competitive forwarding mechanism. They add extensibility at the IP packet level, so the architecture can accommodate heterogeneous types of nodes (active and non-active). The hierarchical organisation of code (forwarding routine-code group-protocol) is another strong point. By the use of a message digest (such as the MD5) upon these parameters, ANTS succeed in a) providing a trustworthy naming scheme for their services and b) protecting their system against spoofing, since the integrity of code can be verified locally.
    - The demand-pull scheme is a clever and scalable way to distribute code rapidly. As ANTS is implemented in a JVM, type safety is provided.
- Main weak points that the reviewed architecture possesses:

- The restricted node API suggests that not all (arbitrary) services can be composed.
    - The requirement of ANTS that the forwarding routines are single threaded, and the fact that are implemented in Java, have a bad effect in performance.
    - The idea of letting untrusted parties use the ANTS framework has the effect that misbehaving programs monopolise system's resources. Until better solutions than the TTL field are found, the use of trusted authorities is indispensable.
    - The limit of 16 KB of code per service can cause difficulties in the composition of demanding services.
- Are there any innovations? :
    - They were among the first to introduce the notion of code mobility inside the network.
    - The lightweight demand-pull protocol that is used for code distribution can be regarded as another innovation.
- Can any specific parts/ideas be adopted?:
    - It is an open issue whether we should adopt a similar to the capsules approach. An interesting approach could be to use capsules to deploy objects, in nodes that have open interfaces and thus can compose all desired services.
    - The hierarchical organisation of code, and the demand pull code distribution scheme could be adopted by FAIN. The latter seems to need some enhancement, as it can lead to a considerable amount of capsule losses (a code server can be used in conjunction).
    - We can draw useful conclusions from the way they implement their services.
- How can it be related to FAIN?
  It can successfully play the role of an execution environment during the evaluation of the R23.2 platforms.
- How easily can it be implemented:
  The ANTS implementation Toolkit v. 1.3 is available, which is the ANTS execution environment.
- Is it worth being evaluated in R23.2?:
  Yes because it has become the reference platform to other ANs and it will be useful to compare FAIN with it. It will also be useful for us to gain experience from the implemented services, and from the way that the code-messages distribution mechanism works.

A.9 Project CANEs (HEL)

- This review is based on: **WP3-HEL-004-R23-Int.doc/WP3-HEL-010-R23-Int.doc**
- Main strong point of the reviewed architecture:
  - The idea of combining injected code (capsules) with pre-loaded one (underlying programs) can be powerful enough to lead to the scalable and controllable composition of AN services.
  - The node interface (Bowman) provides an extension mechanism that can be used dynamically to provide support for additional abstractions and user specific protocols.
  - Services such as application specific congestion control, caching and multicasting, were successfully implemented, with promising results.
- Report on the main weak points that the reviewed architecture possesses:
  - There is not much emphasis on security (i.e. protection against spoofing).
  - If underlying programs are not dynamically loaded, the system has programmability restrictions.
- Are there any innovations? :
  - They were among the first to introduce the idea of mobile code that serves as glue to node-resident code.
  - They provide a framework for the composition of AN services.
- Can any specific parts/ideas be adopted by FAIN?:
  - The idea of dynamically loaded injected/underlying programs, combined with the open/extensible interface, makes the architecture very powerful. This can be a serious input in FAIN.
  - The implementation itself of the NodeOS (Bowman) interface, together with the extension mechanism can also be adopted by FAIN.
  - Ideas/conclusions from congestion control tailored to MPEG streams, and from modulo-lookaround caching schemes can be drawn.
- How easily can it be implemented:

The CANEs execution environment (Odyssey) and the implementation of the NodeOS interface (Bowman) are publicly available.

- Is it worth being evaluated in R23.2:
  Yes, CANEs can play the role of an execution environment and Bowman that of the NodeOS interface, while evaluating hardware platforms in R23.2

A.10 Project Openet (HEL)

- This review is based on: **WP3-HEL-009-R23-Int.doc**
- Main strong point of the reviewed architecture: The Accelar routing switch has two separate planes: forwarding and control. The forwarding engine is composed of ASIC and has 5.6-256 Gbps back plane. The control plane occupies the whole CPU independent of packet forwarding. Therefore the Openet architecture might have high performance and programmability.
- Are there any innovations: Programmable Gigabit Routing Switch (Accelar) and ORE EE
- Can any specific parts/ideas be adopted: Accelar, Oplet Runtime Environment (ORE) SDK
- How easily can it be implemented: ORE and its software development kit are free to registered users and there is also a discussion list from which you can get good support.
- How can it be related to FAIN: It seems that we can get high performance and programmability by the Accelar and ORE. In addition, Openet seems to be useful for interoperability testing.
- Report on the main weak points that the reviewed architecture possesses: When the Accelar starts, the ORE downloads the service package using URL and activates the service and its dependent service. Alternatively, a network administrator can load the service package into the ORE and activates it using the telnet style shell or the ORE control API.
- Is it worth being evaluated in R23.2: **Yes**. Because BANG and Openet follow the same approach in building an active/programmable node. They both put an emphasis on service deployment and they both build on JVM. We believe that porting BANG code to ORE should not be a problem, as it will also benefit from the rich features of ORE. Moreover, if we substitute JFWD with an extended version of the BANG L-interface it will provide us with a very good platform to start with.
- Any other remarks: < **None** >

A.11 Project Tempest (HEL)

- This review is based on: **WP3-HEL-005-R23-Int.doc**
- Main strong point of the reviewed architecture: The strong point of the architecture is the component that is called Divider that facilitates the co-existence of different control architectures based on a number of open interfaces. Generalising this concept to active IP network is quite relevant for FAIN.
- Are there any innovations: The strong point happens to be also an innovation, which has been patented by the proponents of Tempest.
- Can any specific parts/ideas be adopted: There are a lot of ideas that can be adopted and adapted to FAIN. The Divider component bears a lot of similarities with the Resource control framework/component. If we generalise its concepts to support simultaneously different network architectures then EEs in FAIN become virtual networks. Conversely, if we stick to control architectures then EEs become service environments. The other idea is about application specific policies whereby applications may submit their own policies on top of the existing policy framework running in the network.
- How easily can it be implemented: No code is available but implementation of the concepts is straightforward.
- How can it be related to FAIN: What is relevant in FAIN in terms of implementation is how to adapt the ideas in FAIN. Implementation of Tempest is based on ATM switches and services relevant to it. In terms of inter-operating seems possible if Tempest supports the same, as FAIN, node interface and Tempest’s control architectures are ported to FAIN platform (Java)
- Report on the main weak points that the reviewed architecture possesses: Conceptually you can hardly find any worth-mentioning weak points. However, Tempest was conceived for ATM networks, which is a weak point per se.
- Is it worth being evaluated in R23.2: No code is available**.**
- Any other remarks: What Tempest provides is a well-defined framework that clearly separates architectural components and concepts. This framework with a little effort may be adapted to FAIN in such a way that meets FAIN objectives.

A.12 Project Janos (HEL & GMD)

- This review is based on:
  - WP3-GMD-003-R23-Int.doc,
  - WP3-GMD-006-R23-Int.doc
- Main strong point of the reviewed architecture:
  The Janos project's objective is to develop a principled local operating system for active network nodes, oriented to executing untrusted Java bytecode. The strong points of this approach are:
  - Janos is a well-organised, layered architecture that can prove beneficial to resource management and control.
  - Janos provides support for NodeOS interfaces and abstractions.
  - As it is based on OSKit, Janos prototype can run on bare hardware.
  - A very distinct component is the JanosVM, which is a Java runtime with process and resource management augmentations. It is a resource-aware, multi-heap garbage collector, Java Virtual Machine.
  - The Java NodeOS provides interfaces and abstractions to Java-based EEs.
  - ANTS is already ported.
  - Both Java and non-Java EEs are supported. The primary execution environment though is the Java-based ANTSR.
- Are there any innovations:
  - The Lego-like combination of five distinct parts.
- How easily can it be implemented:
  - The project as a whole is very complex and work is ongoing. Importing some interesting parts of this implementation would require a fair amount of work.
- How can it be related to FAIN? Can any specific parts/ideas be adopted?:

  Yes. FAIN could benefit from some concepts/ideas of JanOS and even adopt some parts of it:
  - The combination of OSKit and Moab provide a pure-C NodeOS as well as an implementation of the NodeOS interface.
  - The OSKit alone can be used for porting or further developing an already existing OS.
  - When the Janos Virtual Machine coupled with the Java NodeOS interfaces are implemented, a suitable run-time for other Java-based EEs would be provided.
  - The Flask security model is a robust security framework, from which we can draw useful conclusions.
  - It is also interesting to see how the ANTS services will be realized in the Janos framework.
- Report on the main weak points that the reviewed architecture possesses: There is code available for some parts of the architecture. However, not all parts have been implemented and since this is a complex and time-consuming approach, nothing guarantees that the parts interesting for FAIN project will be made available soon enough to be exploited.
- Is it worth being evaluated in R23.2: Yes, provided that the missing code(JanosVM), is available on time.

A.13 Project Click Router (HEL & NTUA)

- This review is based on:
    - WP3-NTU-001-R23-Int
- Main strong point of the reviewed architecture:
    - Programmability and Interface Flexibility
      A Click router configuration is a directed graph whose nodes are called elements and each element, which is a C++ object, has a class that determines its behaviour. Some element classes support additional arguments, by which the user can determine which Click Router's operations are to be done. Every action performed by a Click router's software is encapsulated in an element, from device reading and writing to queuing, routing table lookups, and counting packets.
    - Packet Storage

  Unlike some systems, Click elements do not have implicit queues on their input and output ports, or the associated performance and complexity costs. Instead, Click queues are explicit objects, implemented by a separate element *(Queu*e). This enables valuable configurations that are difficult to arrange otherwise.

    - Good Performance
      Click Router, running Linux, can forward 64-byte packets at 73.000 packets per second, just 10% slower than Linux alone.
    - Available software

  Click Router's application is free as well as all its numerous software components.

- Are there any innovations:
    - Innovation is the Click Router architecture as every element is an entity and includes in itself his software.
- Can any specific parts/ideas be adopted:
    - Click Router can be included as a hole, as Fan's Node OS .
- How easily can it be implemented:
    - It consists an independent application (its only requirement is the hardware platform, running in Linux), so it can be easily implemented.
- How can it be related to FAIN:
    - It is a full programmable router, so it can serve as FAIN's router.
- Report on the main weak points that the reviewed architecture possesses:
    - Perhaps the only weak point of Click Router is its security even if there are several elements that support IP security as well as RFC 2507-compatible IP header compression and decompression, communication with wireless radios and tunnelling.
- Is it worth being evaluated in R23.2:
    - Of course it is worth being evaluated in R23.2, as Click Router is an interesting platform, with open source. It is a completely software based platform and it seems to have very good performance.

Any other remarks:

A.14 Project Netscript (JSIS)

- This review is based on:

WP3-JSIS-001-R23-Int.doc, WP3-JSIS-002-R23-Int.doc

- Main strong point of the reviewed architecture:

Dynamic composability of protocols from small protocol parts (boxes), part of the software source available.

- Are there any innovations:

Application of the UNIX STREAMS dataflow model to dynamic creation of protocols and active networks.

- Can any specific parts/ideas be adopted:

Not clear how to use Netscript ideas in WP3. Maybe dynamic composability is somehow related to work in WP4.

- How easily can it be implemented:

It seems easy enough, Netscript is written in JAVA (on Solaris) and so it can be added to any node implementation. But it has to be rewritten to comply to future Node API (To speak in DARPA terms is more EE than NodeOS)

- How can it be related to FAIN:

Not in WP3 case.

- Report on the main weak points that the reviewed architecture possesses:

Partial software availability, speed (it seems that Netscript can be slow but no real figures exits), has been frozen for two years (at least web view),

- Is it worth being evaluated in R23.2: No.
- Any other remarks: No.

A.15 Project PRONTO (KPN)

This review is based on: WP3-KPN-004-R23-Int.doc

- Main strong point of the reviewed architecture: *Strong points of the PRONTO platform are the modular design and the use of frame peeking and a separate forwarding layer*
- Are there any innovations: *the frame peeking concept is an innovative approach to increase the performance*
- Can any specific parts/ideas be adopted: *the concept of a separate forwarding layer which operates on the data path level and the concept of frame peeking should be considered when designing the FAIN platform.*
- How easily can it be implemented: *This depends on the node implementation to be used: when a new node is built, these concepts can be included from the design phase. When an existing platform is used, the effort needed to include these features depends on the design of the platform.*
- How can it be related to FAIN: *these concepts relate to better performance for active packet processing, and are therefore suitable in FAIN.*
  (this question is not totally clear to me…)
- Report on the main weak points that the reviewed architecture possesses: *the platform was developed for research purposes. There is not much attention drawn to features like security.*
- Is it worth being evaluated in R23.2: *No, because the platform was specifically developed to research functionality of active nodes. The design is therefore focused on ease-of-use and flexibility. The platform is therefore not suitable to use in an operational environment*

Any other remarks: <INSERT>

A.16 Project ALAN (NTUA)

- This review is based on: WP3-NTU-002-R23-Int.doc
- Main strong point of the reviewed architecture:
    - The use of the Java language makes it flexible and easily implemented onto any OS.
    - ALAN is pretty easy to be deployed on top of current IP networks.
    - ALAN's security is based on the use of trusted hosts and servers as well as the use of signature at proxylets for their verification. In addition, its proxylet runs in its own Java machine.
- Are there any innovations:
    - The key innovation that ALAN has introduced is the fact that that it introduces application-layer networking instead of network-layer networking.
- Can any specific parts/ideas be adopted:
    - ALAN's architecture has some fundamentals differences from FAIN's architecture. Therefore we believe that FAIN cannot adopt any ALAN's component.
- How easily can it be implemented:
    - Since ALAN is addressed to the application level and it uses the Java virtual machine there are not any considerable OS or hardware requirements and it can be easily implemented.
- How can it be related to FAIN:
    - ALAN aims at developing an active network as well as FAIN does. However, since they believe that programming at the router level will introduce difficulty in the deployment of active network infrastructure, we think that it cannot be related to FAIN.
- Report on the main weak points that the reviewed architecture possesses:
    - The ALAN's performance is restricted by the fact that it uses Java virtual machine along with the fact that computational is done at the application layer rather that network level
    - Instead of using the network layer, ALAN makes use of the application layer.
- Is it worth being evaluated in R23.2:
    - No, because ALAN may be close to active networks' techniques but it is based on different approach. It is based on application-layer networking while we in FAIN concentrate at network-layer networking.
- Any other remarks:

ALAN and FAIN are moving towards the same goal through different ways. So, we could say that they are in "parallel" ways.

A.17 Project GENESIS (SAG)

This review is based on: WP3-SAG-002-R23-Int, WP3-SAG-008-R23-Int.

Main strong point of the GENESIS architecture: Genesis is a framework/environment that supports the dynamic creation, deployment and management of virtual network architectures based upon an existing network topology. The created networks may use particular protocols and may be tailored to specific services. Central points of the Genesis architecture are:

Specification of virtual networks ("profiling")

Generation of specified virtual networks ("spawning")

Life cycle management of the generated virtual networks

An important point is that the Genesis Project defines separation concepts for programmable networks which, in future, may lead to standards for interfaces, etc.

Are there any innovations: Profiling method (specification of virtual networks with their characteristics on topology, QoS, connectivity, inheritance from parent networks, etc., and their relation to network controllers) and respective tool support Spawning process (performed by Genesis Framework tools) Routlets (i.e., virtual router nodes) with their interfaces and the respective control units (spawning controller, composition controller, allocation controller, and data path controller) Virtual network design principles: *Separation* (child virtual networks operate in isolation with their traffic carried securely and independent from other networks) – *Nesting* (a child network inherits the capability to spawn other virtual) – *Inheritance* (child networks can inherit architectural components, e.g., resource management capabilities and provisioning characteristics, from parent networks) Metabus (lowest level communication infrastructure)

Can any specific parts/ideas be adopted: Concepts developed in the Genesis Project may be rather relevant for FAIN, in particular the separation principles and design choices taken by the project (see above). In particular, the concept Binding Interface Base (BIB), the Routlet interfaces and the resource management may be mentioned.

How easily can it be implemented: The Genesis Kernel Framework is very ambitious and complex (its a four year project anyway). It may be safe to assume that it is far from being easily implemented. However, taking over some of the concepts may facilitate the FAIN implementations.

How can it be related to FAIN: As already said above: it may provide concepts and ideas.

Report on the main weak points that the reviewed architecture possesses: There is no code available for experiments and concrete evaluation. The Genesis Kernel Framework is a tightly intertwined system using especially adapted routlets, transport modules, controlling units and BIB objects. Changing the framework architecture may produce chaos. There is no consideration of security of any kind

Is it worth being evaluated in R23.2: Further evaluation of the Genesis Framework is only valuable if new material does come up. In particular, as long as no code can be made available, there is no value in a further evaluation of the framework.

With respect to the concepts and separation principles in Genesis, we deem it worthwhile to review the evolving FAIN architecture and principles with a particular consideration of the results of the Genesis project.

Any other remarks:

None.

A.18 PROJECT ABLE (UCL)

Introduction: ABLE is an active network architecture that primarily addresses the network management challenges. Its main component is the active engine that is attached to any IP router to form an active node.

Main advantages: The design is based on standards, thus can easily be deployed in today's IP networks. That is, the mobile code is written in Java, and it is encapsulated together with data using the standard ANEP headers over UDP. The engine communicates with the router using SNMP that enables it to monitor and control the router's operation.

Physically, the IP router and the active engine may either reside on different machines or co-reside inside the same box. This structure simply enables any commercially off-the-shelf (COTS) IP router to be upgraded into an active router by adding an adjunct active engine.

The separation between COTS IP router and the active engine protects non-active traffic from the effects of erroneous operations of the active part of the network, and inflicts minimal additional delay on non-active traffic. It also makes gradual deployment of active nodes in current networks easy.

In today's active network initiatives, capsule applications carry their code and terminate after execution while the programmable switch approach is implemented by 'well-known' session IDs that may receive data and act on it. The ABLE approach can handle both approaches.

Ideas to be adopted for FAIN: The Java implementation provides (nearly) platform-independent encoding and dynamic loading. This also gives rise to flexibility and extensibility. The Java package mechanism in conjunction with Java access rules also provides coarse-grain protection for the EE.

- ABLE's focus on network management issues complements the active node architecture blueprint detailed in the FAIN Technical Annex.

The ABLE node architecture adheres to the guidelines of the Node OS Working Group and it is in line with the FAIN Node architecture. The active engine would be a suitable candidate to serve as an EE implementation example for our project.

A.19 PROJECT ARP (UCL)

Introduction: The ARP (Active Reservation Protocol) project is exploring the use of portable and dynamically extensible protocol code for network control protocols, especially for signalling protocols. In terms of the active networks architecture under development by the DARPA research community, the ARP project is building an Execution Environment (EE) appropriate for active signalling, as well as significant Active Applications (AAs) to execute in that environment.

Main advantages:

- This ARP approach to network control protocols may significantly reduce the need for protocol standardisation, since a (single) implementation of a protocol needs to interoperate only by itself.
- Of interest in the ARP project are active applications that implement network signalling protocols, such as RSVP. It is expected that active signalling will provide rapid deployment of new facilities, simplicity, customisability, and generality of function.
- Additionally, the EE isolates each AA from one another, thus preventing it from either boundary violation or resource violation. The EE is able to isolate itself from the AA such that a malicious AA is prevented from circumventing the EE's mediation between applications and resources. The basic ASP EE mechanism for preventing boundary violations is based on Java's strong typing and safe pointer variables.
- An additional security feature inherent in the ASP EE is the prevention of AA spoofing, i.e., an AA is not allowed to send packets with an AA specification that is not its own. The EE also protects source identification field against modification by an AA.
- The ASP EE supports dynamic loading of AA code, security and resource protection, timing service, and network I/O, i.e., the node operating system (NodeOS) and the EE cooperate to dispatch received active packets to the appropriate AAs and to allow these AAs to send packets into the network.

Main disadvantages:

The ARP Project concluded that the code to implement real network control algorithms is typically too large for an individual capsule; hence the ASP EE supports only out-of-band loading.

Ideas to be adopted for FAIN

- ASP EE and its AAs are expressed in Java byte code, hence provides (nearly) platform-independent encoding and dynamic loading. This also gives rise to flexibility and extensibility.
- The ARP Project has also addressed significant attention towards security aspects, namely EE protection, AA isolation, and prevention of AA spoofing. The Java package mechanism in conjunction with Java access rules provides coarse-grain protection for the EE.
- Even though 'ASP' stands for active signalling protocol, the ASP EE is designed for a wide variety of network control applications in addition to signalling.
- The ASP EE could serve as an EE implementation example for our project since it has been tested and deployed on Abone nodes.

A.20 PRoject Darwin (UCL)

Introduction: The main Darwin objective is the implementation of a programmable router architecture. The Darwin system is a resource management system for service-oriented networks. The four components that, combined, constitute a complete prototype of Darwin:

- scheduling algorithms in support of hierarchical link sharing;
- resource management algorithms in space that integrate the allocation of communication, computing and storage resources;
- a runtime environment for application specific runtime resource management in the network; a reservation protocol for application-aware networks; and
- a controlled test bed for testing, and initial experimentation and evaluation.

Main advantages:

- It will accelerate the evolution of the Internet from an infrastructure that provides basic communication services into a more sophisticated infrastructure that supports a wide range of electronic services such as virtual reality games and rich multimedia retrieval services. In short, specific services are to be installed and instantiated on demand.
- In view of the architecture, the network has mechanisms to verify that application input (i.e., parameters, specification, and programs) can be acted upon safely since incorrect input can endanger the operation of the network. This consists of a combination of language, compilation and runtime techniques.
- The network guarantees that only authorised entities can modify network state. The virtual application mesh also provides means for security. As part of the creation of the mesh, relationships of trust can be established between the nodes in the mesh. This speeds up security checking during execution.

Ideas to be adopted for FAIN:

- Darwin is not specifically an active network initiative. In fact it is categorised under the DARPA Quorum program, which is developing the technologies that will allow mission-critical defence applications to achieve survivable, predictable, and controllable quality of service on a globally managed pool of distributed resources.
- The issue of interest is Darwin's approach towards designing programmable router architecture in supporting control plane extensibility. It is an architecture to support customisation by extending router functionality, pretty much similar to our goals in the FAIN project since customisation allows application providers and service providers to tailor resource management, and thus service quality.
- Potential applications include customisation of traffic control and management, and value-added services. The project envisions deploying a virtual private network (VPN) service, in which VPNs can use different traffic control policies or control protocols. Some tests carried out included video conferencing with video transcoding and mixing support, and application-specific multicast.

A.21 Project ABone (UPC)

- This review is based on: WP3-UPC-003-R23-Int.doc
- Main strong point of the reviewed architecture. It is scalable. In general, the interoperability and portability of code is good.
- Report on the main strong points that the reviewed architecture possesses (e.g. is it suitable for rapid service deployment, is it scalable, is there and emphasis on security etc.)
- Are there any innovations: No
- Can any specific parts/ideas be adopted: No, in our opinion there is nothing really valuable to be adopted by FAIN.
- How easily can it be implemented: Very easy, after a subscription process you only need a UNIX machine and download some code (i.e. anetd,...)
- How can it be related to FAIN: In my opinion, only in the testing phase to prove the interoperability of FAIN.
- Report on the main weak points that the reviewed architecture possesses: Poor performance and security, packets need to be directly addressed to an ABONE node so as to they can be processed in an EE running of that node.
- Is it worth being evaluated in R23.2: Yes/No : No, in my opinion it does not add much to the work that should be developed in FAIN.
- Any other remarks: Its large size makes Abone interesting for the testing phase.

A.22 Project Switchware (UPEN)

- This review is based on: WP3-UPEN-002-R23-Int
- Main strong point of the reviewed architecture: 2-level architecture supporting both active nodes and active packets. Extensive security mechanisms.
- Are there any innovations?
    - 1st active application (active bridging)
    - 1st SIGCOMM paper on active networks (1997)
    - 1st secure node environment (SANE)
    - 1st secure bootstrap of active network node (AEGIS)
    - 1st active internetwork (PLANet)
    - 1st formal specification of AN EE (PLAN)
    - 1st hardware AN element (P4)
- Can any specific parts/ideas be adopted: 2-level architecture, security mechanisms
- How easily can it be implemented: code developed in Ocaml, probably requires non-trivial reimplementation
- How can it be related to FAIN: FAIN should adopt a 2-level architecture to support both active packets and active node extensions. The security mechanisms developed in the context of Switchware should be borne in mind while designing FAIN’s security infrastructure.
- Report on the main weak points that the reviewed architecture possesses: development in Ocaml not particularly portable. Better performance than Java, but probably still too slow for production systems.
- Is it worth being evaluated in R23.2: Yes. Switchware embodies one of the most developed active networking projects under the DARPA umbrella, both in terms of experience as well as in achievement.

Any other remarks: N/A

APPENDIX B: R23.2 – REVIEW & EVALUATION OF ACTIVE NETWORK PLATFORMS

B.1 ANTS Evaluation

B.1.1 ANTS Platform Evaluation – Introduction

This evaluation is dedicated to the ANTS platform. Its development started around 1996 as doctoral thesis of David Wetherall [1], which formed the basis of the ANTS platforms. The first prototypes were further developed and resulted in the ANTS version 1.2, available 98/99 (http://www.sds.lcs.mit.edu/activeware/ants/) and an update to version 1.3.1, which was ready around mid 2000 (http://www.cs.washington.edu/research/networking/ants). The Version 1.3.1 was subject of this evaluation.

B.1.2 Overview Criteria list

This initial list of criteria from the Zurich kick-off meeting is intended to provide a rough overview.

Category	**Examples**	**Time to Avail**	**Cost**	**Porting Time**	**User Population**	**Market Support**	**IPR**	**Exten-sibility**	**Usa-bility**
HW/Plat-form	SUN, PC	now	-	-	large	Supported Product	-		-
Node OS	Solaris 2.6 + Linux	now now	- -	- -	Large Large	Supported Product Open source product	-		-
EE Progr. Languages	Java	now	-	-	large	supported	-		-
Package	ANTS	now	none	hours	???	No / courtesy of developers	MIT – WashU?	?	Not much
Mgmt. Infra-structure	none	-	-	-	-	-	-	-	-
Node/EE Interfaces	Proprietary / Unix	-	none	-	???	No	???		

B.1.3 Runtime Environment

See sections B.1.4.4 and B.1.4.5.

B.1.4 Concepts and Architecture

For further information about ANTS (Active Network Transport System) see also the references (section B.1.9) and the FAIN Documents [9][11] (ANTS platform description and summary) and [10] (ANTS services) .

B.1.4.1 Concept & Overview

The ANTS implementation uses a variant of the in-band approach towards building an active network architecture. Instead of always transporting code with every packet, ANTS nodes cache the most recently used code in order to avoid reloading the code for a related group of packets. Packets, called capsules in the ANTS implementation, carry parameter values for a related piece of code. If the node that a packet passes through contains the related code, the node initialises the code with a packet's parameter values and then executes the code. If the code is not present on the node, the node requests the code from its nearest upstream neighbour. Using this type of code transport mechanism, active nodes become primed “on the fly” by the packets that pass through them. (This paragraph is a slightly adapted paraphrase after [6].)

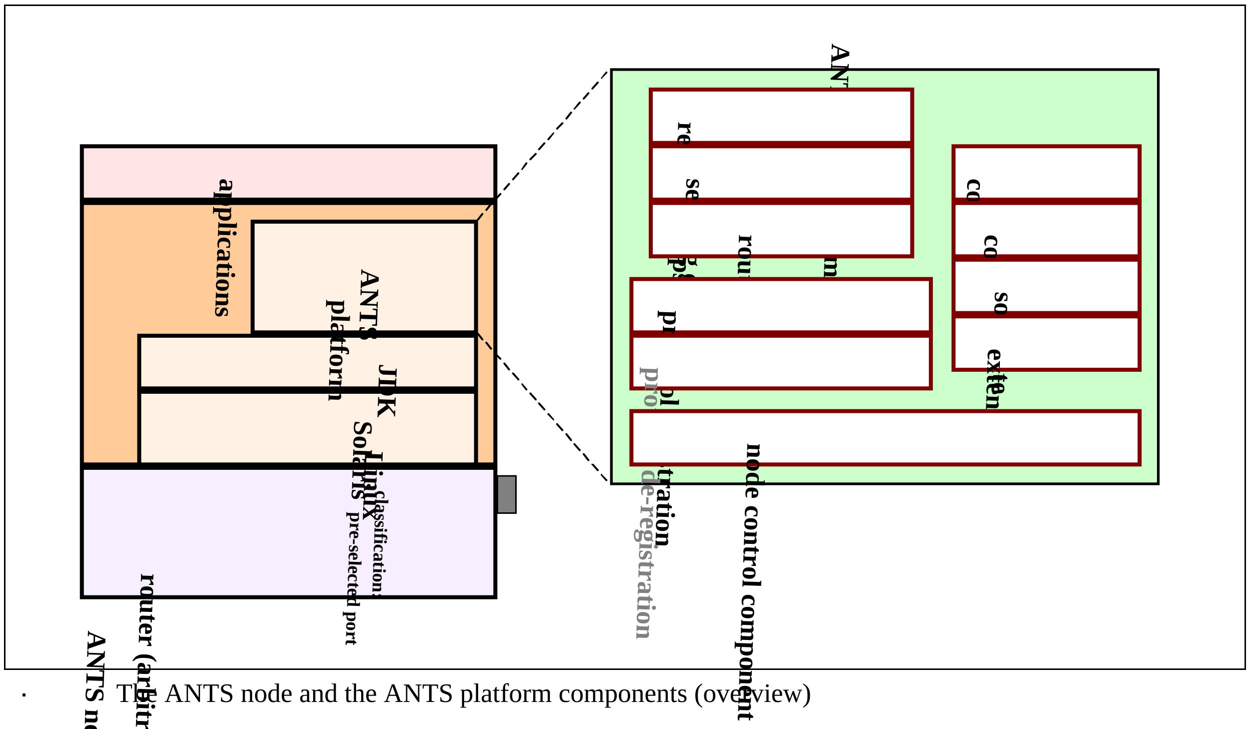
B.1.4.2 Product family

Ants is a university prototype, not a commercial product. Hence it lacks all user support, maintenance, and documentation save a few research papers and the possibility to contact the developers which may react by courtesy. However, there is no licence and no fee. It may be downloaded form the above mentioned sites (section B.1.1).

B.1.4.3 Functional Architecture

B.1.4.3.1 The ANTS Node

The ANTS node (see •) consists of an arbitrary router, the NodeOS (which is JDK over Solaris or Linux), the ANTS platform and, on top, the Java end user application(s).



· The ANTS node and the ANTS platform components (overview)

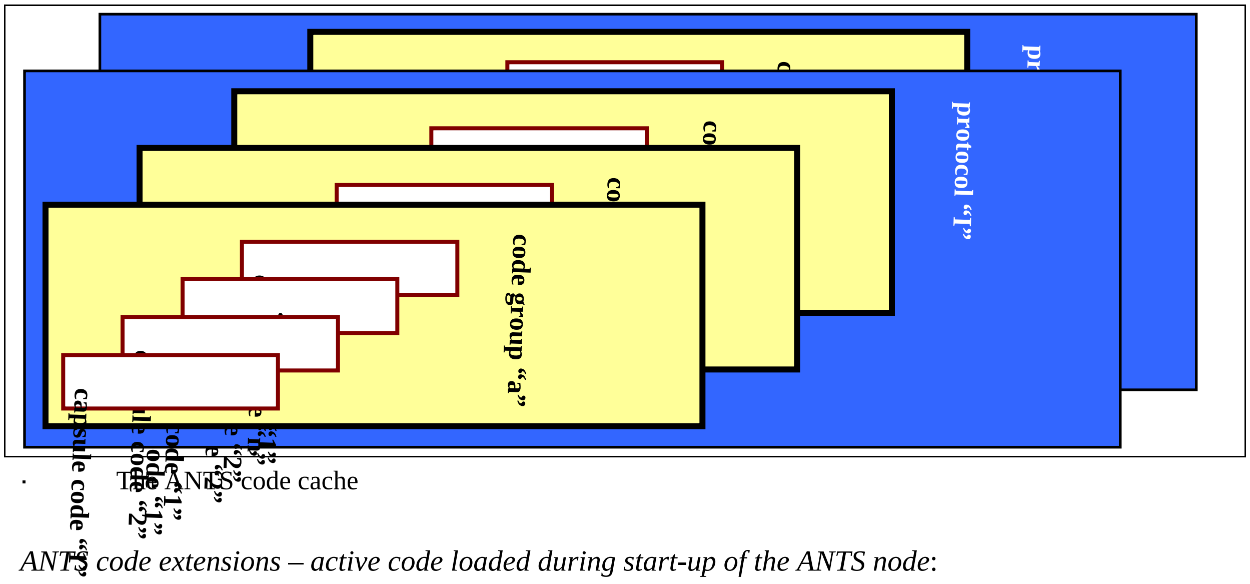
The ANTS platform comprises the node control component, a component (group) for registration of the ANTS protocols (i.e., the active services running on the ANTS platform), a component group responsible for the handling of the active code and the state information associated with the active code and a component group that handles the data path for ANTS packets (the ANTS routing).

ANTS handles its own routing (see section B.1.4.3.3).

The ANTS code cache – protocols, code groups and capsule code:

At an ANTS node, the active code is hierarchically structured according to the following scheme:

- Protocols. One, or possibly more than one protocol form an active service. Protocols have to be explicitly registered at the ANTS node (in the current ANTS version, they cannot be de-registered, yet de-registration seems to be planned). A protocol comprises several code groups.
- A code group is the unit of code distribution (see the code distribution scheme in section B.1.4.3.2). The maximal size of a code group is limited to 64 K. It contains several pieces of capsule code.
- Capsule code is the active code that is referenced by a ANTS packet (i.e., a capsule). It may be arbitrary Java code with somewhat restricted access to system functions (see section B.1.4.6). Capsule code may freely use the “code extensions” available on the ANTS node (for code extensions, see below). The ANTS network administrator has to make sure that all code extensions used by the capsule code is available. Capsule code is derived from the base class “ants.Capsule”.



· The ANTS code cache

ANTS code extensions – active code loaded during start-up of the ANTS node:

In addition to capsule code, a second kind of active code is supported by ANTS, the so-called "code extensions". A code extension is a piece of Java code, which is loaded onto an ANTS node during the node start-up. When a capsule arrives at a node it has access to all extensions loaded on this node. Code extensions are never loaded dynamically and must be distributed by some mechanisms outside of ANTS. They cannot be removed from the node during its lifetime.

There are no restrictions posed upon the code of code extensions. Code extensions are derived from the base class "ants.Extension".

ANTS soft state – the inter-protocol communication mechanism of ANTS:

For the communication between active protocols, ANTS provides a concept called soft state. The soft state is a shared pool of memory with fixed size (configuration of the ANTS node). Each capsule can place entries into this pool. These entries are accessible to other capsules via a key that is determined by the capsule that is writing the entry. Capsules belonging to the same protocol need to know only the key to gain access to the entry. Capsules from other protocols must specify the protocol-id of the creating capsule in addition to the correct key.

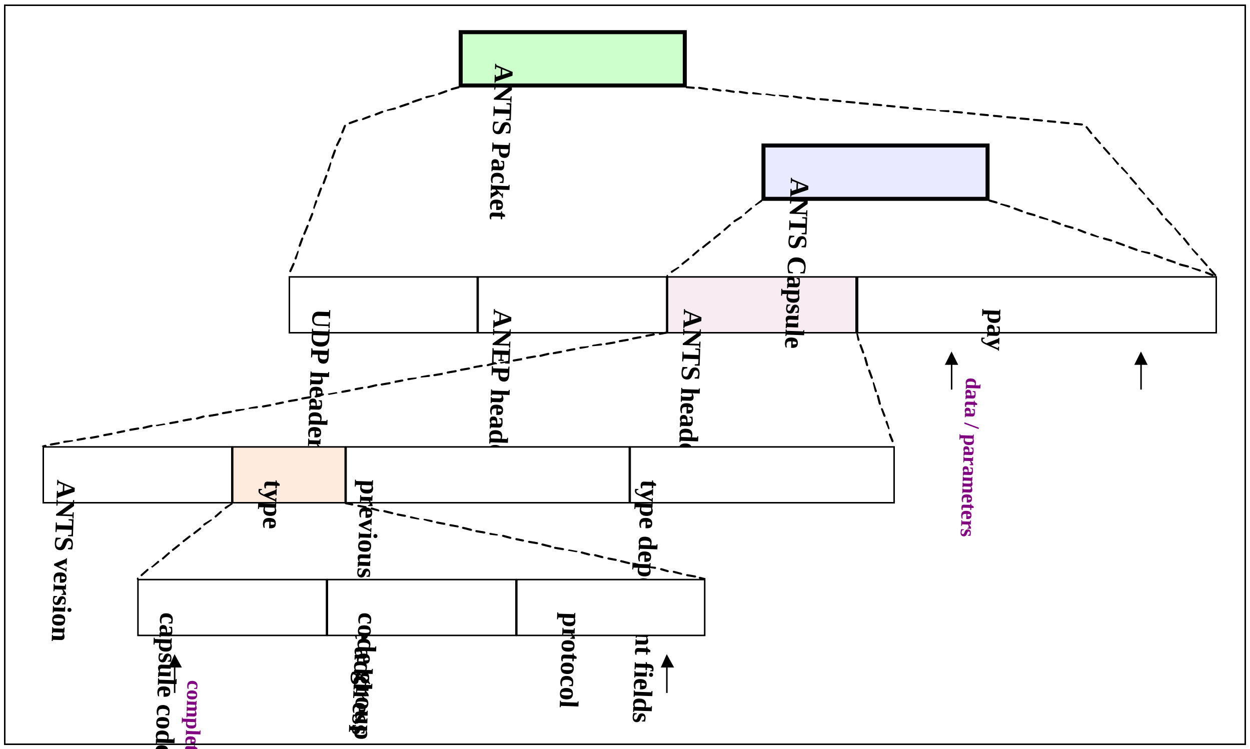
Capsules, belonging to the same protocol as the capsule, which wrote the soft state entry, have unrestricted access to the information contained in the soft state. Capsules belonging to other (foreign) protocols may be granted access explicitly. There is, however, not yet a fine-grained distinction between the different foreign protocols that may access the soft state entries: either all-foreign protocols available on the ANTS node get admission or none.

The number of entries placed into the soft state by one capsule is not restricted. Entries may remain in the soft state unrestrictedly, only if there is no longer free space in the soft state, then the oldest entries are replaced by the new ones.

B.1.4.3.2 The ANTS Capsules and the Code Distribution Mechanism

The ANTS capsule – how the active code gets invoked:

In ANTS, the active packets are called capsules. They have a particular structure: A UDP packet contains an ANEP (version 1) packet, which comprises an ANTS capsule. The capsule consists of the "ANTS header" and the "payload". The capsule structure is depicted in • .

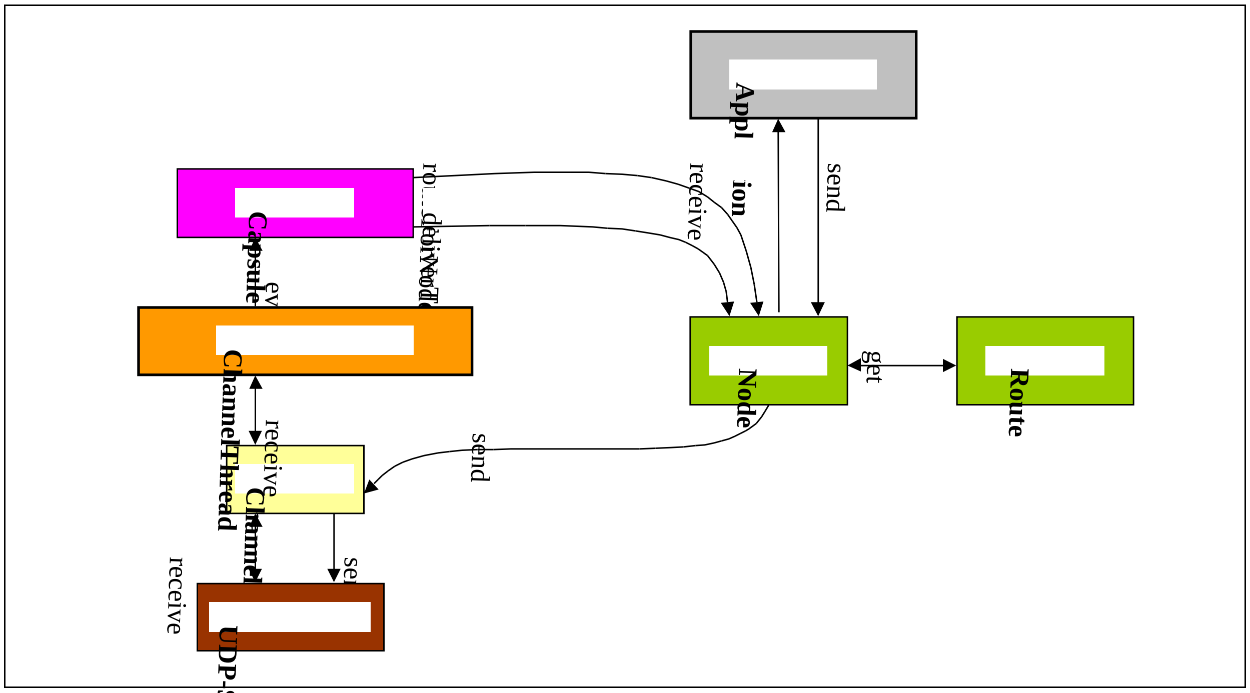


· The ANTS Capsule

In the “type” field of its header, each ANTS capsule carries a reference to the capsule code but not the code itself (which is stored in the local code cache, see above). This reference is a digital signature, which is computed directly from the capsule code (fingerprint); it also indicates the code group and the protocol the capsule code is supposed to belong to. When a capsule arrives at a node, the node checks the local cache for an entry with the requested signature. When the code is found it is executed immediately. If not, then the code distribution mechanism (see below) tries to retrieve the code.

The capsule code signature plays an important role. It is used for different purposes (identification and integrity check):

- Identification of the code used during the look-up of the code reference in the local code cache. When the local code cache contains the code no additional checks are made.
- Identification of the code during the dynamical loading of the capsule code from other ANTS nodes (see below).
- Authentication of the dynamically loaded code by comparing the locally computed signature of the loaded code with the signature contained in the requesting capsule. Both signatures must be equal or the loaded code is neither placed in the local cache nor executed.



· Capsule processing

The "resources" assigned to a capsule are restricted: each capsule carries a figure, indicating the maximal number of "ANTS-hops" (TTL) which is decremented by one if the packet is forwarded by an ANTS node. In theory (e.g., [1]), this resource figure should also restrict the number of newly generated capsules: the sum of the resource figures assigned to all capsules generated at the node on behalf of a certain incoming capsule, including the new resource figure of the capsule itself when forwarded, should be less than the resource figure of the incoming capsule. Unfortunately, this mechanism is not implemented (decrementing by one works; however, to all generated capsules the same resource figure is assigned as to the forwarded one). Careless usage of this feature (bug) can easily ruin all active applications as our test implementations show (loss of nearly all packets, etc.).

The ANTS code distribution mechanism:

ANTS has a lightweight standard protocol for dynamic code loading. If code requested by a newly arrived capsule is not found in the local code cache, it is dynamically retrieved from the node which sent the capsule (unit of code distribution is, however, the code group, thus the request for one capsule code brings the entire, usually tightly related capsule code in the code group to the requesting node). Upon its arrival, the capsule code is (a) cached in the local code cache, (b) the digital signature of the newly arrived code is computed and (c) stored, then (d) the requested capsule code is authenticated (i.e., the digital signature of the new code is compared with the digital signature of the requested code as contained in the capsule, see above), and – if ok – eventually (e) executed.

The flexibility of the code distribution scheme can be employed by the designer of an active application in order to adjust to the needs of the service. David Wetherall, the author of [1], writes: "We believe that our code distribution scheme has qualities that will prove it efficient, adaptive, and robust, though this must be borne out by experimentation. In order for it to best accommodate the largest number of scenarios, we also include a number of special cases. First, for very small protocols, the code may be carried along with every capsule if desired. Second, capsules may be constructed to "prime" a path with protocol code to reduce the start-up period. Finally, popular protocols may simply be preloaded to avoid dynamic code distribution." Thus, the code distribution scheme makes it easy to

- Distribute the code along with the capsules which need the code (dynamic code retrieval at arrival of the referencing capsule if code is not yet available – this includes that the execution of the active service has to wait until the code arrives),

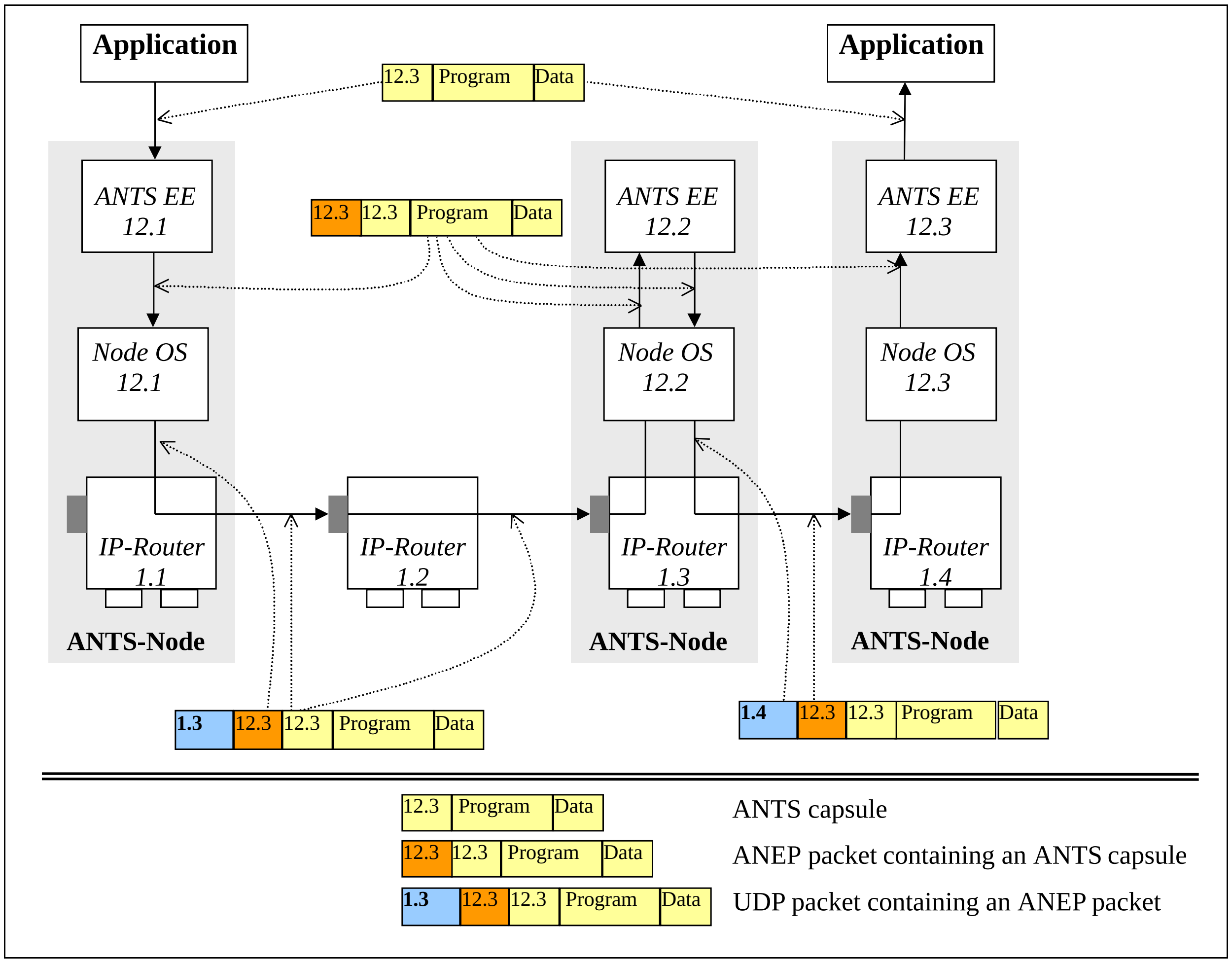
- Force the code distribution by sending a pilot capsule (the code retrieval scheme will be executed ahead of the arrival of the first capsules of the active service),
- Downloading the capsule code during the installation of the ANTS node (no dynamic code distribution but management overhead during the installation of the ANTS node).
- The last mechanism is similar to the mechanism that is responsible for putting the code extensions into place.

B.1.4.3.3 The Embedding of an ANTS Network into an IP Network

Running an ANTS platform requires a configuration of the respective node, including the determination of the ANTS specific routing tables. Packets can be forwarded from one ANTS node to the next only. Along the path between ANTS nodes, not-ANTS routers my occur which do not appear in the ANTS (layer) network (if correctly configured).

ANTS includes standard protocols for routing table updates. If necessary, ANTS nodes can be configured to allow a dynamic update of the routing tables. This mechanism is itself an active service within ANTS.

One IP-host may contain more than one ANTS node.



- The embedding of ANTS into a network

B.1.4.3.4 ANTS Standard Protocols

ANTS has standard protocols for dynamic code loading and routing table updates. The code for these protocols is contained in each ANTS node. Other protocols may be included as well but this requires changes to ANTS and a recompilation. There is no way to add such kind of standard protocols dynamically.

B.1.4.4 Hardware environments to support the platform

ANTS can be used on any Standard-PC or SUN. As ANTS is based on JAVA it should be running on any platform supporting JDK1.1 or JDK1.2.

B.1.4.5 Software environments to support the platform

ANTS is available for Linux 1.2.2 or higher and Solaris 2.6 or higher (Solaris 2.7 was used to run some tests and is known to work).

ANTS is based on JDK1.1 or JDK1.2 (for out tests we used JDK1.2).

B.1.4.6 Interface and instruction set

The implementation of ANTS is based on Java. ANTS capsules and ANTS Code extensions use Java as their implementation language, too.

The ANTS capsule code can use only a small number of Java and ANTS system classes:

- java.lang.Throwable
- java.lang.Exception
- java.lang.SecurityException
- java.lang.ClassNotFoundException
- java.lang.Integer
- java.lang.Long
- java.lang.Object
- java.lang.String
- java.lang.StringBuffer

and

- ants.Node
- ants.Capsule
- ants.DataCapsule
- ants.Xdr
- ants.NodeAddress
- ants.ByteArray
- ants.wrapper.AutoNodeCache
- ants.NodeCache
- ants.Extension

The Java system classes provide means for the exception handling and data classes. The ANTS system classes offers mechanisms to a more specific handling of the capsules and some base classes for the system:

The class “ants.Node” is the environment in which the capsule code is executed. There are a few standard functions as, e.g., “routeForNode” (forwarding capsules), “deliverToApp” (deliver capsules to the local application, i.e., their final destination), “getExtension” (identifying an code extension object), “getCache” (identifying the soft state object), “getAddresse” (identifying the ANTS node address), “register” (registering a new protocol), “unregister” (removing a once registered protocol – this function is not yet implemented), “attachApplication” (associating an application with the ANTS node), and some others (“time”, “sleep”, etc.). The class “ants.NodeAddress” converts IP addresses into strings and vice versa. The class “ants.ByteArray” is, as expected, the data type byte array.

The class "ants.Capsule" is the base class of the capsules. The class "ants.DataCapsule" is derived from "ants.Capsule" and used for the processing of data capsules (data transfer with default routing). The class "ants.Xdr" is a helper class for unwrapping/wrapping capsule data from the arriving packet.

The class "ants.NodeCache" implements the ANTS soft state. It is closely related to the class "ants.wrapper.AutoNodeCache" which also covers the soft state and provides some extensions.

The class "ants.Extension" is the base class for the ANTS code extensions.

B.1.4.7 Development environment

There is no special development environment for ANTS. To develop active protocols for ANTS, any Java development environment can be used. There are no particular requirements.

B.1.4.8 Sample Application

Within ANTS version 1.3.1, two applications are provided: PING (the obvious ping of a node using ANTS capsules) and TRACE (an extended ping that records the address of any node the capsule visits on it's way to the destination node and reports the data back to the sender). See section B.1.5.2 for performance measurements based upon the ANTS PING application.

The implementation of ANTS itself uses capsules for (a) dynamic loading of capsule code, (b) updating the routing tables of an ANTS-node, and (c) negotiations of dynamic addresses on behalf of an ABONE edge node.

In the literature, the following ANTS services are mentioned: active caches for stock quotes [3], active reliably multicast [4], integration of IPv6 and AN [5].

See also the FAIN Document [10] where three application of ANTS (mobile hosts, multicasting and auction services) are described.

B.1.4.9 Missing Functionality and other Remarks

There is no way to assign priorities to the execution of packets. There are examples where this causes an extremely high loss of packets as our experiments demonstrated.

Since only one channel (see •) is used, there is no clear separation of the different data flows in the node (different directions, different nodes, different protocols). As long as a capsule program is active, channels are blocked for all other activities.

In the literature to ANTS, one may find statements that related extensions are planned (de-activating of packets and a later re-activation, suspending the execution of capsule code, rendezvous of packets at a given node, etc.), however, the current implementation does not show any signs of it.

B.1.5 Performance Criteria

B.1.5.1 Criteria List

PC1.) *CPU cycles for packet forwarding (active & data): some Active Network platforms (e.g. MBASH) use their own packet format for data packets (SAPF) which results in faster forwarding speeds than traditional IP forwarding. How many cycles does it take to forward an active packet versus a data packet in the platform?*

See section B.1.5.2 for some measurements; section B.1.4.9 describes a few problems we realised during the experimentation with ANTS.

PC2.) *Packet size (active & data): If the platform uses special packet formats (see below) instead of running over UDP/IP for example, then what are the comparative sizes of these formats?*

ANTS is running over UDP/IP.

For the ping example (see section B.1.4.8), the following packet sizes is reported by ANTS: (a) code request capsule: 92 bytes, (b) code response capsule: 1117 bytes (part 1) & 695 bytes (part 2), (c) ping data capsule: 84 bytes.

PC3.) *Packet format (ANEP, SAPF etc.): Active Network Encapsulation Protocol (ANEP) and Simple Active Packet Format (SAPF) are examples of active packet formats used by various projects to identify packets containing active code and/or to handle data packets without the overhead caused by TCP/UDP and IP.*

ANEP Version 1.

PC4.) *Implementation language of runtime/execution environment: If the runtime/EE is written in Java or another interpreted language, then it will incur a performance penalty, whilst allowing greater portability.*

Java

PC5.) *Implementation language of Node OS: In the cases where a dedicated Node OS is used, then the same notes as the above point apply.*

There is no dedicated NodeOS. Linux and Solaris are implemented in C.

PC6.) *Overhead of inter-active code communication: In some AN platforms, the active code can communicate with each other using something like the Tuple-space in the Linda programming language.*

The overhead for the access to a soft state entry (the ANTS interactive code communication mechanism) is expected to be on the order of a hash table lookup and a couple of function calls. (see section **Error! Reference source not found.**).

PC7.) *Overhead of spawning a new thread: Often AN platforms see themselves as spawning new threads of control on remote systems. What is the cost of such an operation in terms of CPU cycles, memory usage, network usage etc?*

There is no particular concept for spawning new threads on remote systems.

PC8.) *Overhead of mobile code isolation (runtime checking): Some platforms implement isolation of active code, by performing runtime checks on the memory addressing performed by the active code. What is the overhead of putting and getting information from and to the shared memory area?*

There are no additional checks beyond the checks done by the Java virtual machine.

PC9.) *Code shipping vs. code caching: Or capsules vs. plug-ins, which approach is used? Can the platform support both approaches? What is the overhead (in terms of network usage, CPU usage, memory usage etc.) of one approach compared to the other - in the cases where the platform supports both modes? Can the platform automatically switch from one mode to the other. E.g., load a class on-demand and then cache it for future use.*

ANTS capsules do not carry the code themselves, they carry a reference (signature / fingerprint) to the code cached in the node (see section B.1.4.3.2). If the referenced code is not available, a lightweight protocol is used to retrieve the code from the next node upstream the forwarding path of the capsule. For authentication see SC2.

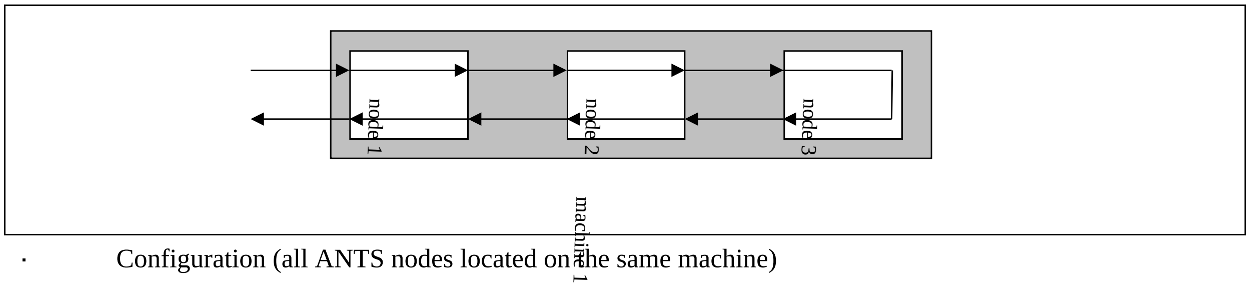
B.1.5.2 Performance Measurements

The following paragraphs contain some performance measurements for ANTS. All reported times are in msec.

This first test reports the RTT (round trip time) for ping-capsules depending on the number of ANTS-Nodes the capsule has to travel before returning to the sending node.

All ANTS nodes located on the same machine:

The first test runs with all ANTS-nodes located on the same machine (in our tests we used a Ultrasparc II with 2 CPUs)



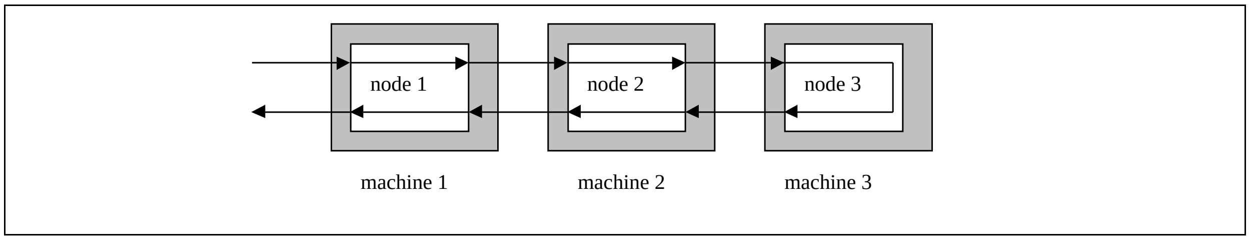
· Configuration (all ANTS nodes located on the same machine)

Result:

	1 Node			2 Nodes			3 Nodes		
	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Run 1	0.00	0.034	6.00	1.00	2.39	49.00	2.00	3.87	38.00
Run 2	0.00	0.029	6.00	1.00	2.43	37.00	2.00	3.95	39.00
Run 3	0.00	0.030	6.00	1.00	2.58	50.00	2.00	3.91	45.00
Avg. 1 - 3	0.00	0.031	6.00	1.00	2.47	45.33	2.00	3.91	40.67

Each ANTS node located on a different machine:

The second test runs with each ANTS-node located on a different machine (in our tests we had to use a heterogeneous network of suns with one Ultrasparc II with two CPUs, one Ultrasparc II with one CPU and one SPARCstation 5)



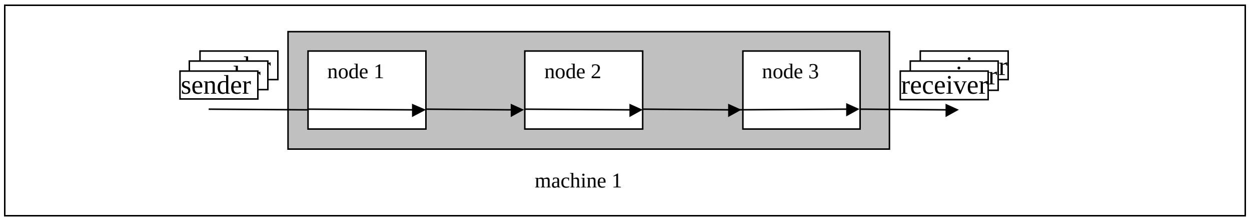
· Configuration (each ANTS node located on a different machine)

Results:

	1 Node			2 Nodes			3 Nodes		
	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Run 1	0.00	0. 026	6.00	3.00	6.13	162.00	7.00	11.34	484.00
Run 2	0.00	0. 029	6.00	3.00	5.76	100.00	8.00	10.40	348.00
Run 3	0.00	0. 037	6.00	3.00	7.85	527.00	7.00	10.03	259.00
Avg. 1 - 3	0.00	0. 031	6.00	3.00	6.58	263.00	7.33	10.59	363.67

Different number of Sender/Receiver pairs in parallel:

The test sends 5000 messages containing only a timestamp and a sequence number from a sender to a receiver. After sending a message, the sender waits 1 msec before sending the next message (without this break many messages are lost – there is no apparent reason why the sender has to sleep for 1 msec in order to avoid this excessive packet loss). The test runs with different numbers of sender/receiver pairs in parallel. The test is run on one machine only.



· Configuration (different number of sender/receiver pairs in parallel)

Results:

1x Send/Receive	Receive # 1		
	Min.	Avg.	Max.
Run 1	1.0	2.79	43.0
Run 2	1.0	2.89	51.0
Run 3	1.0	2.82	49.0
Avg. 1 - 3	1.0	2.83	47.7

2x Send/Receive	Receive # 1			Receive # 2		
	Min.	Avg.	Max.	Min.	Avg.	Max.
Run 1	1.0	4.29	63.0	1.0	4.29	65.0
Run 2	1.0	4.58	52.0	1.0	4.82	56.0
Run 3	1.0	4.53	60.0	1.0	4.52	60.0
Avg. 1 - 3	1.0	4.47	58.3	1.0	4.54	60.3

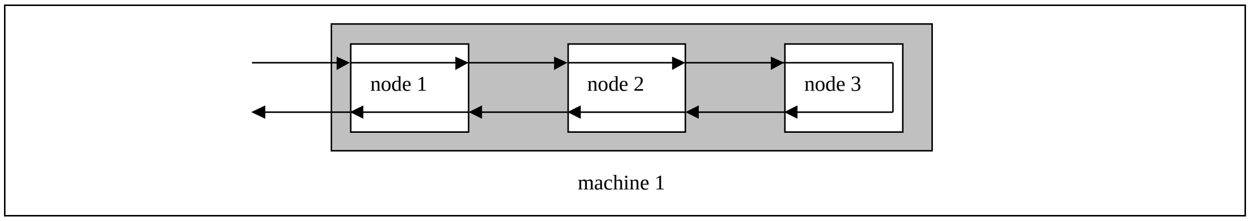
3x Send/ Receive	Receive # 1			Receive # 2			Receive # 3		
	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Run 1	1.0	6.14	69.0	1.0	6.23	86.0	1.0	6.05	77.0
Run 2	1.0	5.70	61.0	1.0	5.86	69.0	1.0	5.81	68.0
Run 3	1.0	5.84	87.0	1.0	5.78	81.0	1.0	5.95	83.0
Avg. 1 - 3	1.0	5.89	72.3	1.0	5.96	78.7	1.0	5.94	76.0

4x Send/Re ceive	Receive # 1			Receive # 2			Receive # 3			Receive # 4		
	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Run 1	1.0	8.33	93.0	1.0	8.45	83.0	1.0	8.34	89.0	1.0	8.25	92.0
Run 2	1.0	7.99	78.0	1.0	8.03	85.0	1.0	7.89	75.0	1.0	7.83	85.0
Run 3	1.0	7.70	104.0	1.0	8.20	104.0	1.0	8.04	97.0	1.0	7.78	99.0
Avg. 1-3	1.0	8.01	91.7	1.0	8.23	90.7	1.0	8.09	87.0	1.0	7.95	92.0

5x Send/Re ceive	Receive # 1			Receive # 2			Receive # 3			Receive # 4			Receive # 5		
	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Run 1	1.0	10.85	80.0	1.0	10.97	93.0	1.0	10.88	85.0	1.0	10.73	95.0	1.0	11.01	93.0
Run 2	1.0	10.85	94.0	1.0	10.89	107.0	1.0	10.92	106.0	1.0	10.80	106.0	1.0	11.04	108.0
Run 3	1.0	10.59	90.0	1.0	10.81	77.0	1.0	10.76	79.0	1.0	10.59	86.0	1.0	10.76	79.0
Avg. 1-3	1.0	10.76	88.0	1.0	10.89	92.3	1.0	10.85	90.0	1.0	10.71	95.7	1.0	10.94	93.3

Different number of pings in parallel:

The test sends 5000 ping-messages containing only a sequence number. After sending a message, the sender waits 1 msec before sending the next message. Without this break many messages are lost. The test is run with different numbers of ping-applications in parallel. The test runs on one machine only.



· Configuration (different number of pings in parallel)

Results:

1x Ping	Ping # 1		
	Min.	Avg.	Max.
Run 1	2.0	3.70	58.0
Run 2	2.0	3.66	50.0
Run 3	2.0	3.59	67.0
Avg. 1 - 3	2.0	3.65	58.3

2x Ping	Ping # 1			Ping # 2		
	Min.	Avg.	Max.	Min.	Avg.	Max.
Run 1	2.0	4.49	54.0	2.0	4.33	52.0
Run 2	2.0	4.79	60.0	2.0	4.70	72.0
Run 3	2.0	4.46	57.0	2.0	4.43	53.0
Avg. 1 - 3	2.0	4.58	57.0	2.0	4.49	59.0

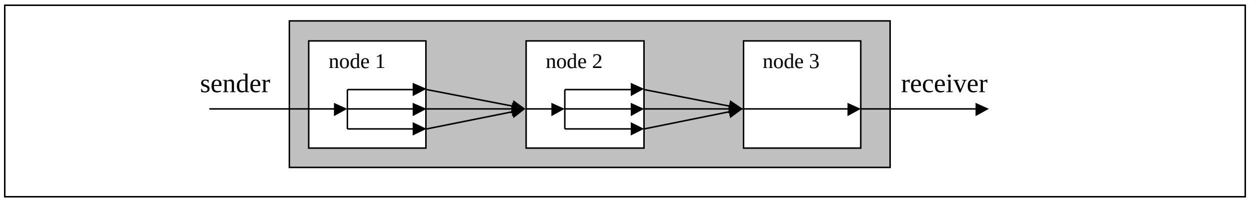
3x Ping	Ping # 1			Ping # 2			Ping # 3		
	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Run 1	2.0	5.79	63.0	2.0	5.71	59.0	2.0	5.60	61.0
Run 2	2.0	5.87	64.0	2.0	5.87	60.0	2.0	5.81	67.0
Run 3	2.0	5.91	71.0	2.0	5.90	74.0	2.0	6.05	73.0
Avg. 1 - 3	2.0	5.86	66.0	2.0	5.83	64.3	2.0	5.82	67.0

4x Ping	Ping # 1			Ping # 2			Ping # 3			Ping # 4		
	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Run 1	2.0	7.16	80.0	2.0	7.15	82.0	2.0	7.33	82.0	2.0	7.03	74.0
Run 2	2.0	7.48	105.0	2.0	7.41	117.0	2.0	7.42	103.0	2.0	7.54	111.0
Run 3	2.0	7.15	74.0	2.0	7.28	77.0	2.0	7.30	68.0	2.0	7.30	63.0
Avg. 1-3	2.0	7.26	86.3	2.0	7.28	92.0	2.0	7.35	84.3	2.0	7.29	82.7

5x Ping	Ping # 1			Ping # 2			Ping # 3			Ping # 4			Ping # 5		
	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Run 1	2.0	8.71	113.0	2.0	8.74	112.0	2.0	8.80	108.0	2.0	8.75	107.0	2.0	8.95	106.0
Run 2	2.0	8.58	70.0	2.0	8.58	79.0	2.0	8.70	73.0	2.0	8.54	77.0	2.0	8.67	77.0
Run 3	2.0	8.80	107.0	2.0	8.80	102.0	2.0	8.60	112.0	2.0	8.74	104.0	2.0	8.68	110.0
Avg. 1-3	2.0	8.70	96.7	2.0	8.71	97.7	2.0	8.70	97.7	2.0	8.68	96.0	2.0	8.77	97.7

Capsule copies:

The test sends 5000 messages from a sender to a receiver. After sending a message, the sender waits a given number of msec before sending the next message. Each node (except the node at the destination) copies the received capsule a number of times and forwards the copies together with the received capsule to the next node. The nodes don't wait after sending a message. The test runs on one machine only.



· Configuration (capsule copies)

Results:

The sender waits 1 msec after sending a message:

Number of copies	1				2				3				4			
Interval = 1	Min.	Avg.	Max.	Loss	Min.	Avg.	Max.	Loss	Min.	Avg.	Max.	Loss	Min.	Avg.	Max.	Loss
Run 1	1.00	6.38	191.00	0%	2.00	98.11	183.00	8.13%	5.00	117.46	224.00	50.44%	3.00	104.00	557.00	62.83%
Run 2	1.00	4.76	59.00	0%	1.00	99.95	279.00	8.72%	2.00	115.80	224.00	49.53%	2.00	104.61	217.00	63.76%
Run 3	1.00	5.12	74.00	0%	1.00	94.16	192.00	7.62%	5.00	117.25	237.00	49.97%	7.00	100.14	228.00	62.33%
Avg. 1 - 3	1.00	5.42	108.00	0%	1.33	97.41	218.00	8.16%	4.00	116.84	228.33	49.98%	4.00	102.92	334.00	62.97%

The sender waits 10 msec after sending a message:

Number of copies	1				2				3				4			
Interval = 10	Min.	Avg.	Max.	Loss	Min.	Avg.	Max.	Loss	Min.	Avg.	Max.	Loss	Min.	Avg.	Max.	Loss
Run 1	1.00	4.13	52.00	0%	1.00	7.17	72.00	0%	2.00	100.00	188.00	22.61%	2.00	86.65	279.00	43.42%
Run 2	1.00	4.12	78.00	0%	1.00	7.18	58.00	0%	2.00	99.75	171.00	22.41%	3.00	87.06	160.00	43.70%
Run 3	1.00	4.21	63.00	0%	1.00	7.33	51.00	0%	5.00	98.99	184.00	22.09%	2.00	86.37	173.00	43.32%
Avg. 1 - 3	1.00	4.15	64.33	0%	1.33	7.23	60.33	0%	3.00	99.58	181.00	22.37%	2.33	86.69	204.00	43.48%

The sender waits 20 msec after sending a message:

Number of copies	1				2				3				4			
Interval = 20	Min.	Avg.	Max.	Loss	Min.	Avg.	Max.	Loss	Min.	Avg.	Max.	Loss	Min.	Avg.	Max.	Loss
Run 1	1.00	3.21	68.00	0%	1.00	4.32	54.00	0%	1.00	10.70	375.00	0.38%	4.00	75.47	196.00	26.13%
Run 2	1.00	3.17	49.00	0%	1.00	4.48	338.00	0%	1.00	9.95	110.00	0.01%	2.00	74.85	169.00	25.96%
Run 3	1.00	3.25	61.00	0%	1.00	4.40	49.00	0%	1.00	10.76	182.00	0.14%	2.00	74.34	234.00	25.45%
Avg. 1 - 3	1.00	3.21	59.33	0%	1.00	4.40	147.00	0%	1.00	10.47	222.33	0.18%	2.66	74.89	199.67	25.85%

There are further reports about performance measures to be found in [1].

B.1.6 Security criteria

SC1.) *Cryptographic subsystem: Does the platform have such a subsystem? Can it provide cryptographic routines to the active code in the system?*

There is no cryptographic subsystem. Cryptographic routines can be provided by an ANTS code extension (see section B.1.4.1).

SC2.) *Authentication, Authorisation, Auditing: How does the AN platform perform authentication? Does it use digital signatures and certificates to authenticate any incoming code?*

Each ANTS capsule carries a digital signature (see section B.1.4.3.2).

How is authorisation performed?

There is no authorisation within ANTS. There is also neither the possibility to associated users with particular capsules and to check those associations at an ANTS node. There are, consequently, no mechanisms to grant specific rights or to induce restrictions to particular users.

Does the system authorise the code as if it was a user on the system?

No.

How fine-grained is the privilege model in the platform?

There is no privilege model at all.

Does it use capabilities?

No.

Does the system provide logging facilities for auditing purposes?

No. There is the possibility to configure system messages in 8 categories, which can be written to the system file STDERR. These messages currently comprise information about routing table modifications, information about capsule loading, and some error messages. We judge these mechanism as a debugging tool rather than an audit log. As debugging tool, it may be helpful and even sufficient; as audit log, it is far from fulfilling the requirements of a practical usage.

SC3.) *Language protection (SafetyNet, PLAN): Does the language used to write the active code that the platform processes provide any form of protection? In Java there is protection against accessing data beyond the bounds of an array, SafetyNet and PLAN are dedicated languages for programming ANs and they provide many language-based security features.*

ANTS uses Java to write the active code and relies mainly on the security mechanisms of Java. There is only a limited number of additional security mechanisms:

- Capsule code can use only a small number of Java and ANTS system classes (see section B.1.4.6).
- The execution time of a capsule is limited (watchdog-timer)
- Capsule resources limitation (which does not work properly in version 1.3.1, see section B.1.4.3.2).

There is no limitation for the memory allocation of an activated capsule:

SC4.) *Sandboxing of EE: Does the EE run in a sandbox to protect the hosting node from being corrupted by malicious or buggy code?*

Unix respective Java mechanisms.

Does the EE run as 'root' on a UNIX machine?

No.

SC5.) *Protection of mobile code: How does the platform protect the active code executing on it?*

Java.

Does it provide isolation properties?

Nothing beyond the isolation properties provided by the Java virtual machine. In [7] the author states:

> *“Additionally, there is no safety mechanism to prevent capsules from corrupting the objects stored in the node cache by other capsules. There is also no safeguard against capsules monopolising node resources. It is clear that additional security mechanisms are necessary to ensure secrecy, integrity, and availability for all the users in the network. These mechanisms have not yet been implemented and are currently being investigated.“*

Does it provide code mangling (cf. Fritz Hohl - University of Stuttgart) or encrypted execution of functions (Sander & Tschudin - Berkeley University)?

No.

SC6.) *Protection of hosts: How does the platform protect itself from damage from malicious or buggy code?*

The NodeOS is protected from damage from the ANTS platform or from capsule code by the fact that both run as user level programs under the Unix operating system.

The ANTS-platform is protected from damage from malicious or buggy capsule code only by the mechanisms provided by Java. The platform and the capsule code are executed by the same virtual machine.

Furthermore, the availability of system functions is restricted (see SC3), in particular, there are no other IO functions than the ones proffered by ANTS.

SC7.) *Security in ANs can be seen in 3 different levels (personal opinion: (a) EE-security, (b) Node Security, (c) Active Code-security. The question that arises is not only what is offered but also in which level and how far this security support goes in matter of configurability and openness (e.g. can I plug in my encryption algorithms or do I have to use only the provided ones?).*

a) ANTS does not provide any special security features (no encryption algorithms, no audit logs, etc.). There is no mechanism to add security features dynamically.
b) Unix mechanisms (see also SC21).
c) same as a).

The ANTS code extensions mechanism provides for a rather unrestricted extensibility. However, extensions have to be called by capsules and hence are not fit to modify system mechanisms and implement any security feature.

SC8.) *Secure communication/Confidentiality. Is it supported between AN nodes ? (e.g. with the usage of SSL/TLS or other proprietary protocol).*

No.

Does the EE provide such services to the active code?

No.

Does the NodeOS provide it also to EEs (for secure inter-EE communication)?

Java provides secure sockets. The current implementation of ANTS doesn’t use secure sockets, however.

SC9.) *Integrity. Is it guaranteed? Will modifications e.g. in the active code be detected ?*

ANTS uses a fingerprint mechanisms to derive a signature from the code.

SC10.) *Accountability and Non-repudiation. Is the EE able to provide proof to third parties of the activities that took place?*

No. There is only a rudimentary log available (see SC2). Since capsules are not associated with a user, there is no way to do any accounting (see also SC2).

SC11.) *Availability. Is there a way to ensure the availability of the basic services to the users even in case of denial of service attacks? E.g. QoS enabled connections between AN nodes, Economy-based resource consumption etc.*

No. It is rather easy to jeopardise the system.

SC12.) *Authentication. What means are there to authenticate the active code? Digital Signature validation ?*

An ANTS node checks the signature of dynamically loaded code against the signature contained in the requesting capsule. There are no mechanisms to assign a digital signature obtained from some independent, trusted authority to a piece of code and to verify this signature before the code is executed (see also SC2).

Login/pass approach ?

No.

SC13.) *Access Control & Authorisation. How is the access to node's resources mediated and controlled ?*

The current implementation of ANTS contains no mechanisms to control access to a node’s resources.

What policy system is used ? Is the policy syntax a standard one (e.g. Java policy syntax) or a EE-specific one ?

There is no policy system and therefore no policy syntax. It seems to be possible to use the Java policy system and it’s syntax but that requires future extensions to ANTS and is not covered in the current implementation.

Runtime Authorisation or static ?

There is no authorisation at all, neither static nor runtime.

SC14.) *EE/Node authentication. Is it possible to authenticate the EE or the AN node before visiting and executing in it (remotely from another node).*

No.

SC15.) *Is there a secure way to install/de-install/find/get active code and/or services to the AN node ?*

No particular secure way to install, etc., the capsule code (only the finger print mechanism, see SC9). De-installation is not possible (see SC16). Find is possibly using the capsule signature. The code distribution mechanism provides the get functionality.

For the code extensions, there are no secure installation procedures at all. It is installed manually in the ANTS nodes. If installed, there are no means to eliminate the extensions (save rebooting the node completely). No find and get functionality.

SC16.) *Is there a way to revoke active code that travels within the AN infrastructure ?*

No (there is no working de-install function; the function as such is available in the ANTS code, but it is an implementation dummy).

SC17.) *Does the NodeOS provide sandboxing between EEs ?*

The NodeOS is a Unix of some flavour. ANTS runs as a normal user level process and is therefore separated from other user level processes and from the NodeOS itself.

SC18.) *Is there a network wide management of the security ? (e.g. network-wide policy change)*

No. The current implementation of ANTS doesn’t contain a management component at all.

SC19.) *Is there a secure schema to ensure the survivability of Audit logs?*

There are no audit logs, see SC2 & SC7.

SC20.) *Does the EE take any measures to provide safe code execution ? Are there any AN specific language-based techniques used (SafetyNet, PLAN) or general ones e.g. Java?*

ANTS is based on Java. See also SC3, SC7 and section B.1.4.6.

SC21.) *What security services does the NodeOS provide to the EE it hosts ?*

Its Unix; see SC7.

What security services do the EEs provide to the active code that execute ? (e.g. Cryptographic support)

None.

SC22.) *Does active code take any countermeasures to preserve the privacy of its data, detect malicious nodes and unwanted modifications to its code ? (e.g. partial result encapsulation, execution tracing, computing with encrypted data etc ?)*

Nothing beyond Java.

SC23.) *Is there dedicated hardware used ? (e.g. smart cards)*

No.

SC24.) *What are the requirements of the AN components from the underlying infrastructure ? (e.g. administrator access is required for full resource handling.)*

No.

B.1.7 Flexibility criteria

FC1.) *Capsules vs. Plug-ins: how 'late' can we bind functionality? Capsules offer a larger degree of flexibility, how do plug-in platforms compensate? Can we mix both models of Active Networking?*

ANTS uses a code distribution scheme (see section B.1.4.1) which ensures that ANTS capsules (eventually) correspond to capsule code resident in the ANTS node. Not available code is retrieved from the node’s neighbours. ANTS code extensions are plug-ins which are installed manually in the node. Also capsule code may be pre-installed manually during the node set-up. There is sufficient mixing of models.

FC2.) *At what layer can we program down to? In software radios we can reprogram all the way down to the link-layer*

The current implementation of ANTS is based on the usage of standard protocols as a transport mechanism for capsules (UDP).

FC3.) *How restricted is the behaviour of the mobile code? (related to security)*

See SC3 and others.

FC4.) *Sandboxing: Does the platform support sandboxing, if so how restrictive is it?*

See SC3 and others.

FC5.) *Language-based control (e.g. static typing): How does this restrict our expressiveness with the language?*

No restrictions (see SC3).

FC6.) *Backwards compatibility: Can different versions of the platform inter-operate with other versions?*

Each capsule contains a version number containing the version of the creating ANTS-node. This version number is checked against the version of the receiving node. If they are not equal the packet is discarded. This implies that all nodes in an ANTS-network must use the same version of ANTS.

FC7.) *What technology is used for installation/de-installation of services (e.g. mobile agents, RPC-paradigm etc)*

See FC1: code distribution scheme (see section B.1.4.1) and manual distribution of code extensions.

FC8.) *Is the system scalable and in what levels.*

In principle, there are no obvious impediments for scalability, at least for the transport part, otherwise, Java may be the bottleneck. We don’t have any practical experiences though.

B.1.8 Miscellaneous

MC1.) *Is the interoperability issue touched ? What foreign EEs can the platform host ?*

ANTS nodes does not interoperate with other active network platforms (see [2]).

MC2.) *Can the platform inter-operate with test bed platforms e.g., ABONE and ANON?*

Yes, e.g., ABONE.

MC3.) Robustness & fault tolerance ?

... were obviously no issue during the development of ANTS. Its a prototype intended to serve as a proof of concepts.

MC4.) *Performance of the system. Does it require special hardware support for the "best-performance" configuration?*

No – there is no particular "best-performance" configuration (see MC3).

MC5.) *Which Software License is the code published under? (E.g., GPL, MPL, BSD etc.)*

No licence, no fee.

MC6.) *Code availability (E.g., can the code be downloaded now?)*

See section B.1.4.2.

MC7.) *Code maturity (Is the code release quality? Beta? Alpha?)*

University prototype of good quality.

MC8.) *Is the code actively maintained?*

No. ANTS is no product, thus maintenance depends on the intentions of the authors. See section B.1.4.2.

MC9.) *What are the dependencies of the code? Does it require Java? A specialised NodeOS?*

Java and Unix.

MC10.) *What features does the NodeOS provide?*

Whatever features Unix does provide.

MC11.) *Which NodeOS API does the platform match, if at all?*

Nothing particulars.

B.1.9 Conclusion

ANTS is a conceptually very interesting project. Its implementation provides a suitable test bed and serves very well as proof of concept. However, the rather complete absence of security, safety and management features suggests not to use ANTS as the sole basis of a FAIN development.

B.1.10 References

[1] David Wetherall: *Service Introduction in an Active Network*. PhD Thesis, Massachusetts Institute of Technology, Feb. 1999

[2] D. Wetherall, U. Legedza, J. Guttag: *Introducing New Internet Services: Why and How*. IEEE NETWORK Magazine, July 1998

[3] K. L. Calvert (ed.): *Architectural Framework for Active Networks Version 1.0*. Active Network Working Group, University of Kentucky, July 1999

[4] U. Legedza, D. Wetherall, J.Guttag: *Improving the Performance for Distributed Applications Using Active Networks*. IEEE INFOCOM, San Francisco, April 1998

[5] Li-wei H. Lehmann, S. J. Garland, D. L. Tennenhouse: *Active Reliable Multicast*. MIT Laboratory for Computer Science, IEEE INFOCOM, San Francisco, April 1998

[6] D. M. Murpy: *Building an active Node on the Internet*. Thesis for the degree of Master of Engineering in Electrical Engineering and Computer Science, MIT May 1997

[7] Van C. Van: *A Defence Against Address Spoofing Using Active networks*. Thesis for the degrees of Bachelor of science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science, MIT May 1997

[8] N.N.: *Node OS Specification*. AN Node OS Working Group, 24. Jan. 2000

[9] Fain Document WP3-HEL-006-R23-INT (ANTS evaluation)

[10] Fain Document WP3-HEL-010-R23-INT (ANTS services)

[11] Fain Document WP3-HEL-012-R23-INT (ANTS evaluation summary)

B.2 Switchware evaluation

B.2.1 Concepts and Architecture

The SwitchWare system is based upon three main components:

- Aegis: This is a secure bootstrap mechanism
- ALIEN: This is the actual active node itself – with this we can download new code or extend the node on-the-fly
- PLAN (Packet Language for Active Networks): This is the programming language used for writing active packets. Its features are that it uses type theory to create a restricted language, which is guaranteed to have certain safety and security properties. PLAN code is used to access node-resident services.

B.2.2 Performance Criteria

B.2.3 Security Criteria

SC1.) *Cryptographic subsystem: Does the platform have such a subsystem? Can it provide cryptographic routines to the active code in the system?*

ALIEN supports cryptographic operations to validate incoming active extensions. These routines can be accessed by active extensions.

SC2.) *Authentication, Authorisation, Auditing: How does the AN platform perform authentication? Does it use digital signatures and certificates to authenticate any incoming code?*

PLAN packets are not authenticated, but ALIEN extensions are authenticated and verified.

How is authorisation performed?

SwitchWare differentiates between PLAN packets and ALIEN extensions. ALIEN extensions are allowed to access node internals, but PLAN packets are not. With the Secure PLAN extension, the active packets present cryptographic credentials on the users' behalf, and the nodes' policies are then consulted to see if the packet can perform the tasks that it wants to perform.

Does the system authorise the code as if it was a user on the system?

No.

How fine-grained is the privilege model in the platform?

N\A

Does it use capabilities?

No.

Does the system provide logging facilities for auditing purposes?

Yes

SC3.) *Language protection (SafetyNet, PLAN): Does the language used to write the active code that the platform processes provide any form of protection? In Java there is protection against accessing data beyond the bounds of an array, SafetyNet and PLAN are dedicated languages for programming ANs and they provide many language-based security features.*

PLAN programs are guaranteed to terminate and to be typesafe.

SC4.) *Sandboxing of EE: Does the EE run in a sandbox to protect the hosting node from being corrupted by malicious or buggy code?*

PLAN programs are guaranteed to be safe, ALIEN extensions are not.

Does the EE run as 'root' on a UNIX machine?

No.

SC5.) *Protection of mobile code: How does the platform protect the active code executing on it?*

“Module Thinning” is used to restrict the interface that the active code can export to the outside world.

Does it provide isolation properties?

N\A

Does it provide code mangling (cf. Fritz Hohl - University of Stuttgart) or encrypted execution of functions (Sander & Tschudin - Berkeley University)?

No.

SC6.) *Protection of hosts: How does the platform protect itself from damage from malicious or buggy code?*

N\A

SC7.) *Security in ANs can be seen in 3 different levels (personal opinion: (a) EE-security, (b) Node Security, (c) Active Code-security. The question that arises is not only what is offered but also in which level and how far this security support goes in matter of configurability and openness (e.g. can I plug in my encryption algorithms or do I have to use only the provided ones?).*

A) EE security is provided by cryptographic primitives for ALIEN extensions & Secure-PLAN packets.

B) Node Security can either be provided by the Linux Node OS or by the Nemesis Node OS. The Secure Active Node Environment (SANE) supports node-to-node authentication using Diffie-Hellman cryptographic key-exchange.

C) PLAN programs have certain safety properties that are guaranteed at compile time.

SC8.) *Secure communication/Confidentiality. Is it supported between AN nodes ? (e.g. with the usage of SSL/TLS or other proprietary protocol).*

No.

Does the EE provide such services to the active code?

No.

Does the NodeOS provide it also to EEs (for secure inter-EE communication)?

No.

SC9.) *Integrity. Is it guaranteed? Will modifications e.g. in the active code be detected ?*

ALIEN active extensions are signed, PLAN programs are not.

SC10.) *Accountability and Non-repudiation. Is the EE able to provide proof to third parties of the activities that took place?*

No. Only logging is available (see SC2).

SC11.) *Availability. Is there a way to ensure the availability of the basic services to the users even in case of denial of service attacks? E.g. QoS enabled connections between AN nodes, Economy-based resource consumption etc.*

PLAN programs have resource limitations imposed on them, whereas ALIEN active extensions do not.

SC12.) *Authentication. What means are there to authenticate the active code? Digital Signature validation ?*

Secure PLAN and ALIEN active extensions use cryptographic credentials to authenticate themselves and to provide integrity checks as well.

Login/pass approach ?

No.

SC13.) *Access Control & Authorisation. How is the access to node's resources mediated and controlled ?*

The SwitchWare port to the Nemesis operating system provides fine grained resource control to the Active Network platform.

What policy system is used ? Is the policy syntax a standard one (e.g. Java policy syntax) or a EE-specific one ?

No.

Runtime Authorisation or static ?

There is no authorisation at all, neither static nor runtime.

SC14.) *EE/Node authentication. Is it possible to authenticate the EE or the AN node before visiting and executing in it (remotely from another node).*

Nodes can authenticate each other, but active packets cannot authenticate nodes in advance.

SC15.) *Is there a secure way to install/de-install/find/get active code and/or services to the AN node ?*

No.

SC16.) *Is there a way to revoke active code that travels within the AN infrastructure ?*

No.

SC17.) *Does the NodeOS provide sandboxing between EEs ?*

No.

SC18.) *Is there a network wide management of the security ? (e.g. network-wide policy change)*

No.

SC19.) *Is there a secure schema to ensure the survivability of Audit logs?*

There is no mechanism to ensure the security of the logs.

SC20.) *Does the EE take any measures to provide safe code execution ? Are there any AN specific language-based techniques used (SafetyNet, PLAN) or general ones e.g. Java?*

PLAN is a specific EE, whereas ALIEN is based on OCaML.

SC21.) *What security services does the NodeOS provide to the EE it hosts ?*

Standard OS security functions.

What security services do the EEs provide to the active code that execute ? (e.g. Cryptographic support)

None.

SC22.) *Does active code take any countermeasures to preserve the privacy of its data, detect malicious nodes and unwanted modifications to its code ? (e.g. partial result encapsulation, execution tracing, computing with encrypted data etc ?)*

No.

SC23.) *Is there dedicated hardware used ? (e.g. smart cards)*

No.

SC24.) *What are the requirements of the AN components from the underlying infrastructure ? (e.g. administrator access is required for full resource handling.)*

No.

B.2.4 Flexibility Criteria

FC1.) *Capsules vs. Plug-ins: how 'late' can we bind functionality? Capsules offer a larger degree of flexibility, how do plug-in platforms compensate? Can we mix both models of Active Networking?*

SwitchWare can use both capsules and plugins. Capsules are created using the PLAN programming language and plugins are created with the ALIEN Library Extensions.

FC2.) *At what layer can we program down to? In software radios we can reprogram all the way down to the link-layer*

We can program down to the network layer (Layer 3).

FC3.) *How restricted is the behaviour of the mobile code? (related to security)*

See SC3 and others.

FC4.) *Sandboxing: Does the platform support sandboxing, if so how restrictive is it?*

See SC3 and others.

FC5.) *Language-based control (e.g. static typing): How does this restrict our expressiveness with the language?*

PLAN restricts what resources we are allowed to access using a combination of static typing and runtime checks. We cannot alter the nodes' behaviour using PLAN. If we used ALIEN Library plugins, then we can access the nodes' internals, thus increasing our expressiveness and also our ability to cause damage.

FC6.) *Backwards compatibility: Can different versions of the platform inter-operate with other versions?*

N/A

FC7.) *What technology is used for installation/de-installation of services (e.g. mobile agents, RPC-paradigm etc)*

Mobile Agent paradigm for both capsules and plugins.

FC8.) *Is the system scalable and in what levels.*

N/A

B.2.5 Miscellaneous

MC1.) *Is the interoperability issue touched ? What foreign EEs can the platform host ?*

SwitchWare nodes do not interoperate with other active network platforms.

MC2.) *Can the platform inter-operate with test bed platforms e.g., ABONE and ANON?*

Yes, ABONE.

MC3.) Robustness & fault tolerance ?

N/A

MC4.) *Performance of the system. Does it require special hardware support for the "best-performance" configuration?*

SwitchWare can be run on the P4 network element to increase performance

MC5.) *Which Software License is the code published under? (E.g., GPL, MPL, BSD etc.)*

No licence, no fee.

MC6.) *Code availability (E.g., can the code be downloaded now?)*

From the project's homepage.

MC7.) *Code maturity (Is the code release quality? Beta? Alpha?)*

University prototype of good quality.

MC8.) *Is the code actively maintained?*

No. ANTS is no product, thus maintenance depends on the intentions of the authors.

MC9.) *What are the dependencies of the code? Does it require Java? A specialised NodeOS?*

Java and Unix.

MC10.) *What features does the NodeOS provide?*

Whatever features Unix does provide. SwitchWare has also been ported to the Nemesis platform which has advanced resource control features.

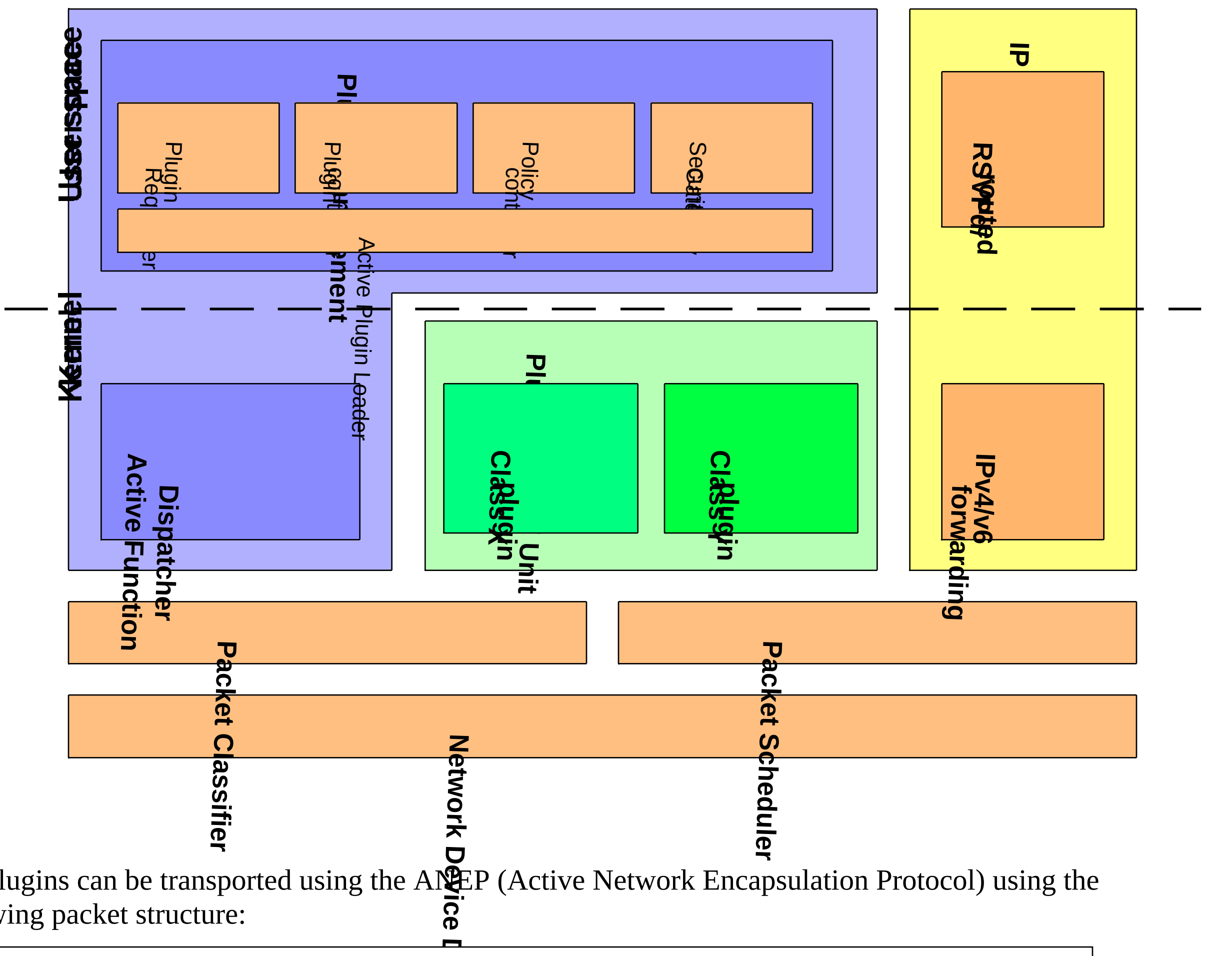
MC11.) *Which NodeOS API does the platform match, if at all?*

Nothing particular.

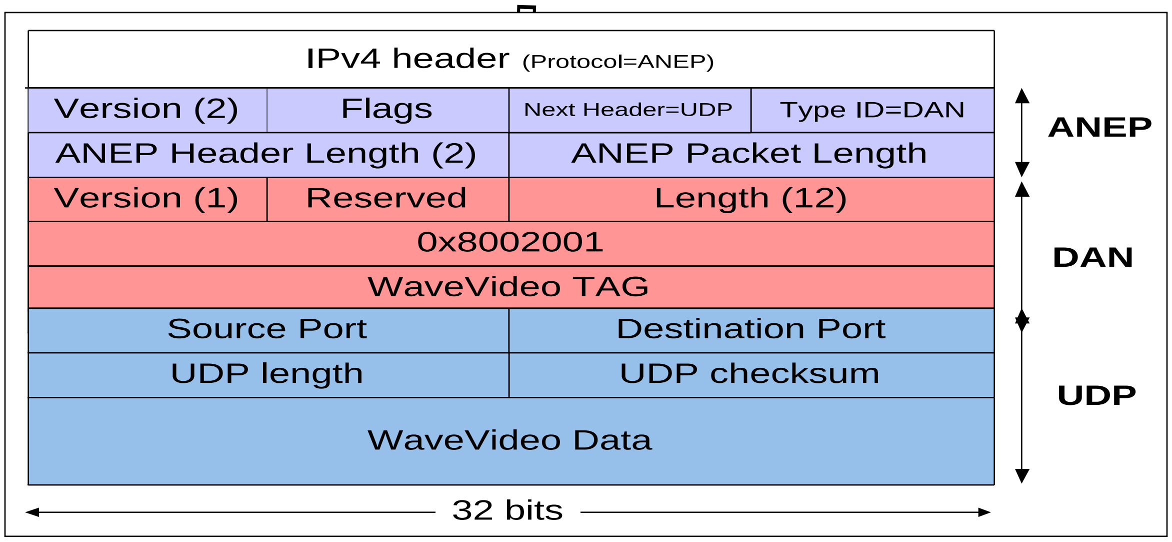
B.3 ANN Evaluation

B.3.1 Concepts and Architecture

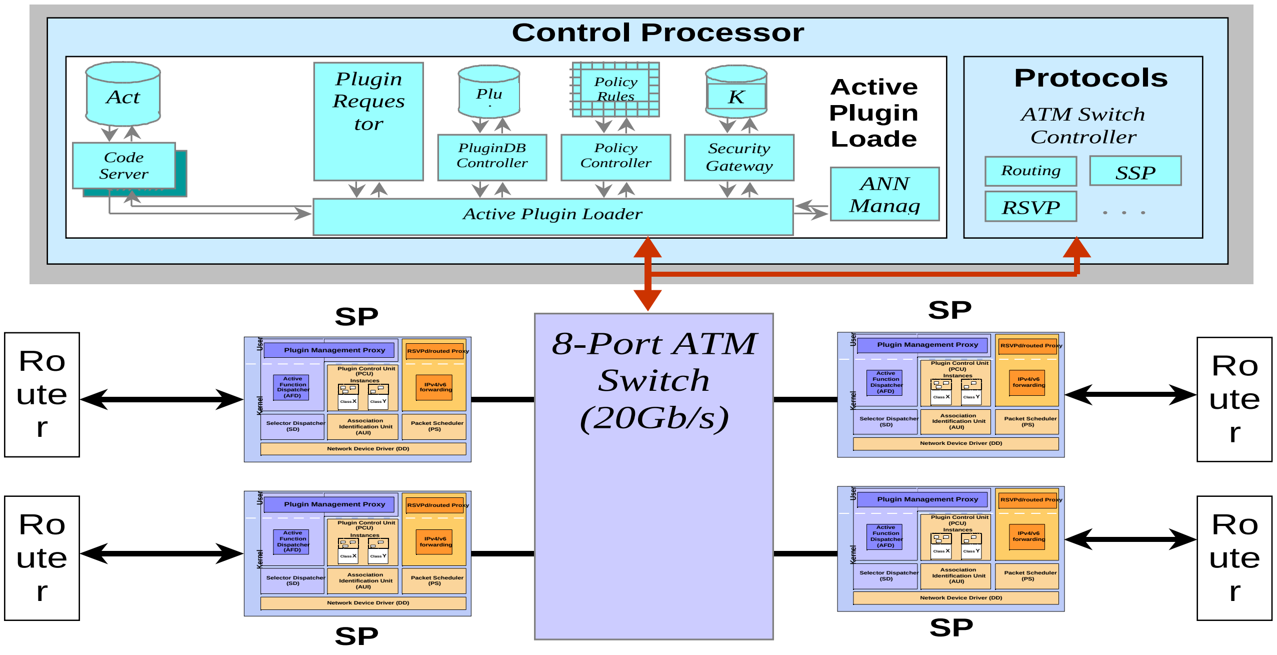
ANN (Active Network Node) is a Execution Environment, which runs in kernel space and uses the active extensions (plugins) approach, as opposed to capsules, for code distribution. The system architecture can be seen below:



The plugins can be transported using the ANEP (Active Network Encapsulation Protocol) using the following packet structure:



The system is designed for maximum performance, it achieves this partly by running in the kernel, but also by running on top of dedicated hardware i.e., the Washington University switch (WUGS). The architecture of the ANN with the EE and the hardware platform looks like:



B.3.2 Performance Criteria

PC1.) *CPU cycles for packet forwarding (active & data): some Active Network platforms (e.g. MBASH) use their own packet format for data packets (SAPF) which results in faster forwarding speeds than traditional IP forwarding. How many cycles does it take to forward an active packet versus a data packet in the platform?*

N\A

PC2.) *Packet size (active & data): If the platform uses special packet formats (see below) instead of running over UDP/IP for example, then what are the comparative sizes of these formats?*

ANN is running over ANEP/UDP/IP.

PC3.) *Packet format (ANEP, SAPF etc.): Active Network Encapsulation Protocol (ANEP) and Simple Active Packet Format (SAPF) are examples of active packet formats used by various projects to identify packets containing active code and/or to handle data packets without the overhead caused by TCP/UDP and IP.*

ANEP Version 1 or higher.

PC4.) *Implementation language of runtime/execution environment: If the runtime/EE is written in Java or another interpreted language, then it will incur a performance penalty, whilst allowing greater portability.*

C.

PC5.) *Implementation language of Node OS: In the cases where a dedicated Node OS is used, then the same notes as the above point apply.*

There is no dedicated NodeOS. ANN runs on NetBSD, which is implemented in C.

PC6.) *Overhead of inter-active code communication: In some AN platforms, the active code can communicate with each other using something like the Tuple-space in the Linda programming language.*

There is no specific mechanism for inter-active code communication.

PC7.) *Overhead of spawning a new thread: Often AN platforms see themselves as spawning new threads of control on remote systems. What is the cost of such an operation in terms of CPU cycles, memory usage, network usage etc?*

There is no particular concept for spawning new threads on remote systems.

PC8.) *Overhead of mobile code isolation (runtime checking): Some platforms implement isolation of active code, by performing runtime checks on the memory addressing performed by the active code. What is the overhead of putting and getting information from and to the shared memory area?*

These active packets ship NetBSD kernel modules, there are no checks.

PC9.) *Code shipping vs. code caching: Or capsules vs. plug-ins, which approach is used? Can the platform support both approaches? What is the overhead (in terms of network usage, CPU usage, memory usage etc.) of one approach compared to the other - in the cases where the platform supports both modes? Can the platform automatically switch from one mode to the other. E.g., load a class on-demand and then cache it for future use.*

ANN uses the active extensions/plugins paradigm exclusively.

B.3.3 Security Criteria

SC1.) *Cryptographic subsystem: Does the platform have such a subsystem? Can it provide cryptographic routines to the active code in the system?*

The DAN (Distributed Code Caching) architecture has an RSA cryptographic subsystem, these routines are not exported to the code running on the ANN.

SC2.) *Authentication, Authorisation, Auditing: How does the AN platform perform authentication? Does it use digital signatures and certificates to authenticate any incoming code?*

Code is authenticated with RSA keys.

How is authorisation performed?

N\A.

Does the system authorise the code as if it was a user on the system?

No.

How fine-grained is the privilege model in the platform?

There is no privilege model at all.

Does it use capabilities?

No.

Does the system provide logging facilities for auditing purposes?

No. There is no specialised logging facilities.

SC3.) *Language protection (SafetyNet, PLAN): Does the language used to write the active code that the platform processes provide any form of protection? In Java there is protection against accessing data beyond the bounds of an array, SafetyNet and PLAN are dedicated languages for programming ANs and they provide many language-based security features.*

The plugins are NetBSD kernel modules written in C. There is no protection or isolation provided by the language.

SC4.) *Sandboxing of EE: Does the EE run in a sandbox to protect the hosting node from being corrupted by malicious or buggy code?*

No.

Does the EE run as 'root' on a UNIX machine?

Yes.

SC5.) *Protection of mobile code: How does the platform protect the active code executing on it?*

There is no protection.

Does it provide isolation properties?

No.

Does it provide code mangling (cf. Fritz Hohl - University of Stuttgart) or encrypted execution of functions (Sander

& Tschudin - Berkeley University)?

No.

SC6.) *Protection of hosts: How does the platform protect itself from damage from malicious or buggy code?*

The platform offers no protection. All plugins are executed in the kernel, hence there is no OS protection.

SC7.) *Security in ANs can be seen in 3 different levels (personal opinion: (a) EE-security, (b) Node Security, (c) Active Code-security. The question that arises is not only what is offered but also in which level and how far this security support goes in matter of configurability and openness (e.g. can I plug in my encryption algorithms or do I have to use only the provided ones?).*

There is RSA crypto used for signing and encryption. This is not pluggable or extensible

SC8.) *Secure communication/Confidentiality. Is it supported between AN nodes ? (e.g. with the usage of SSL/TLS or other proprietary protocol).*

No.

Does the EE provide such services to the active code?

No.

Does the NodeOS provide it also to EEs (for secure inter-EE communication)?

No.

SC9.) *Integrity. Is it guaranteed? Will modifications e.g. in the active code be detected ?*

RSA signing of shipped code provides this functionality.

SC10.) *Accountability and Non-repudiation. Is the EE able to provide proof to third parties of the activities that took place?*

No.

SC11.) *Availability. Is there a way to ensure the availability of the basic services to the users even in case of denial of service attacks? E.g. QoS enabled connections between AN nodes, Economy-based resource consumption etc.*

No. It is rather easy to jeopardise the system.

SC12.) *Authentication. What means are there to authenticate the active code? Digital Signature validation ?*

See SC7.

Login/pass approach ?

No.

SC13.) *Access Control & Authorisation. How is the access to node's resources mediated and controlled ?*

The current implementation of ANN contains no mechanisms to control access to a node’s resources.

What policy system is used ? Is the policy syntax a standard one (e.g. Java policy syntax) or a EE-specific one ?

There is an ANN specfic policy system for deciding which active code should be accepted at a node and which should not.

Runtime Authorisation or static ?

There is no authorisation at all, neither static nor runtime.

SC14.) *EE/Node authentication. Is it possible to authenticate the EE or the AN node before visiting and executing in it (remotely from another node).*

No.

SC15.) *Is there a secure way to install/de-install/find/get active code and/or services to the AN node ?*

No.

SC16.) *Is there a way to revoke active code that travels within the AN infrastructure ?*

No.

SC17.) *Does the NodeOS provide sandboxing between EEs ?*

No.

SC18.) *Is there a network wide management of the security ? (e.g. network-wide policy change)*

No. The current implementation of ANN doesn’t contain a management component at all.

SC19.) *Is there a secure schema to ensure the survivability of Audit logs?*

There are no audit logs, see SC2 & SC7.

SC20.) *Does the EE take any measures to provide safe code execution ? Are there any AN specific language-based techniques used (SafetyNet, PLAN) or general ones e.g. Java?*

No.

SC21.) *What security services does the NodeOS provide to the EE it hosts ?*

N\A.

What security services do the EEs provide to the active code that execute ? (e.g. Cryptographic support)

None.

SC22.) *Does active code take any countermeasures to preserve the privacy of its data, detect malicious nodes and unwanted modifications to its code ? (e.g. partial result encapsulation, execution tracing, computing with encrypted data etc ?)*

No.

SC23.) *Is there dedicated hardware used ? (e.g. smart cards)*

No.

SC24.) *What are the requirements of the AN components from the underlying infrastructure ? (e.g. administrator access is required for full resource handling.)*

Root access is required in order for the NetBSD kernel modules to be inserted.

B.3.4 Flexibility Criteria

FC1.) *Capsules vs. Plug-ins: how 'late' can we bind functionality? Capsules offer a larger degree of flexibility, how do plug-in platforms compensate? Can we mix both models of Active Networking?*

ANN uses the plugin paradigm for code distribution.

FC2.) *At what layer can we program down to? In software radios we can reprogram all the way down to the link-layer*

Network layer 3.

FC3.) *How restricted is the behaviour of the mobile code? (related to security)*

See SC3 and others.

FC4.) *Sandboxing: Does the platform support sandboxing, if so how restrictive is it?*

See SC3 and others.

FC5.) *Language-based control (e.g. static typing): How does this restrict our expressiveness with the language?*

No restrictions (see SC3).

FC6.) *Backwards compatibility: Can different versions of the platform inter-operate with other versions?*

N\A.

FC7.) *What technology is used for installation/de-installation of services (e.g. mobile agents, RPC-paradigm etc)*

The plugins are located on a code server where they can be fetched..

FC8.) *Is the system scalable and in what levels.*

In principle, there are no obvious impediments for scalability.

B.3.5 Miscellaneous

MC1.) *Is the interoperability issue touched ? What foreign EEs can the platform host ?*

No, none.

MC2.) *Can the platform inter-operate with test bed platforms e.g., ABONE and ANON?*

No.

MC3.) Robustness & fault tolerance ?

... were obviously no issue during the development of ANN. Its a prototype intended to serve as a proof of concepts.

MC4.) *Performance of the system. Does it require special hardware support for the "best-performance" configuration?*

When the ANN system is located with a WUGS router, then it will reach its maximum performance.

MC5.) *Which Software License is the code published under? (E.g., GPL, MPL, BSD etc.)*

Research license, no fee.

MC6.) *Code availability (E.g., can the code be downloaded now?)*

Contact ETH Zurich.

MC7.) *Code maturity (Is the code release quality? Beta? Alpha?)*

University prototype of good quality.

MC8.) *Is the code actively maintained?*

The ANN project is still under active development.

MC9.) *What are the dependencies of the code? Does it require Java? A specialised NodeOS?*

NetBSD and (optionally) WUGS router..

MC10.) *What features does the NodeOS provide?*

Whatever features NetBSD does provide.

MC11.) *Which NodeOS API does the platform match, if at all?*

Nothing particular.

B.4 BANG evaluation

This should be taken from the second iteration of the R23.1 list of evaluation platforms. The current list of platforms is (subject to alteration):

- BANG : design and implement the prototype of an active network platform on top of high-speed IP router (Hitachi GigabitRouter2000, GR2k); offer sufficient flexibility for network control and management while retain high performance for forwarding

BANG phase one developed AN platform using mobile agents for code transport

BANG phase two developed DPE-based AN platform to support active QoS services and distributed object-oriented (multimedia) applications

B.4.1 Criteria

- Initial set of criteria from Zurich kick-off meeting. This was a rough draft by Jonathan Smith made during the actual project meeting. It is supposed to give a high-level overview of what categories are important for evaluation criteria.

Category	**Examples**	**Time to Avail**	**Cost**	**Porting Time**	**User Population**	**Market Support**	**IPR**	**Exten-sibility**	**Usa-bility**
HW/Plat-form	GR2000	now	-	-	large	Supported Product	Hitachi	-	-
Node OS	JVM upon Linux or other OSes, Jonathan ORB	now	-	-	Large	Supported Product Open source product	-	Yes	high
EE Progr. Languages	Java, C	now	-	none	large	supported	-	-	high
Package	OrbAN & interface modules	now	none	-	internal	no	Hitachi	Yes	-
Mgmt. Infra-structure	CORBA	now	-	-	-	supported	-	-	-
Node/EE Interfaces	generic abstraction to be aligned with P1520	-	-	-	-	not yet	-	Yes	-

Note: usability is hard to measure due to lack of trials.

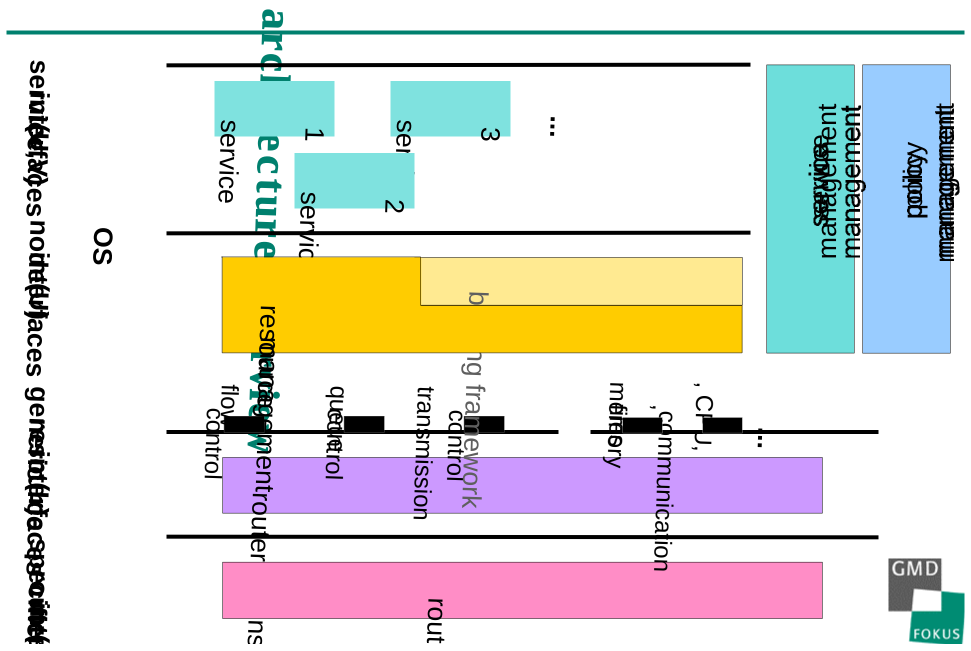
Since we do not have one particular goal of the FAIN project with which we can evaluate candidate platforms against, we define sets of criteria which we can then use to evaluate candidate platforms. We define the following sets:

- Hardware platform
- Performance
- Security
- Flexibility

B.4.2 Hardware platform

- Concept & Overview
- Hitachi Gigabit router GR2000 with flexible CLI interface for QoS configuration.
- in 1st phase, mobile agents used for distributing code.

- focused on an EE for running & managing service components/plug-ins. For the purpose, ORB-based DPE is enhanced with flexible binding framework, resource management framework, and component management capability. P1520 layered reference model provides the architectural rationale.
- IEEE P1520 L interface was trailed with generic resource abstractions.
- Product family
- BANG developed platform prototype for Hitachi, as a research trial. It can be seen a value-added module for GR2000 in support of service provisioning. IPR belongs to Hitachi Ltd, Japan.
- Functional Architecture
    - for details, see R23.1 review on BANG, the layered node architecture mainly contains binding framework, resource control framework, active component management, and policy management.



- Hardware environments to support the platform
- GR2000 as programmable network element, PC as proxy running active platform in control/mgmt. plane.
- Software environments to support the platform
- Jonathan ORB & Grasshopper upon JVM, Linux soft-router for service modules, e.g. DiffServ, metering.
- Interface and instruction set
- Command line interface with a complete configuration commands.
- generic resource abstractions including queuing control, transmission control, and flow control. The interface enables QoS functions such as advanced reservation for Intserv, Diffserv PHB/PDB, filtering, etc.

- Development environment
- JDK1.2, Jonathan ORB2.0, Grasshopper
- Sample Application
- advanced reservation for Web TV; Diffserv for VoIP; mobile agent for DoS isolation.
- Performance & Scalability
- see section 4 & 5.
- Miscellaneous

B.4.3 Performance Criteria

PC1.) *CPU cycles for packet forwarding (active & data): some Active Network platforms (e.g. MBASH) use their own packet format for data packets (SAPF) which results in faster forwarding speeds than traditional IP forwarding. How many cycles does it take to forward an active packet versus a data packet in the platform?*

out of band signalling is used for distributing code. No active packet is supported yet. But ANEP protocol stack can be very straightforwardly plugged in to the platform to have such capability.

PC2.) *Packet size (active & data): If the platform uses special packet formats (see below) instead of running over UDP/IP for example, then what are the comparative sizes of these formats?*

code are transmitted via binding which is GIOP over TCP/IP.

PC3.) *Packet format (ANEP, SAPF etc.): Active Network Encapsulation Protocol (ANEP) and Simple Active Packet Format (SAPF) are examples of active packet formats used by various projects to identify packets containing active code and/or to handle data packets without the overhead caused by TCP/UDP and IP.*

no active packets.

PC4.) *Implementation language of runtime/execution environment: If the runtime/EE is written in Java or another interpreted language, then it will incur a performance penalty, whilst allowing greater portability.*

Java with Java-C bridge.

PC5.) *Implementation language of Node OS: In the cases where a dedicated Node OS is used, then the same notes as the above point apply.*

There is no dedicated NodeOS. JVM upon Linux is seen as a logic Node OS.

PC6.) *Overhead of inter-active code communication: In some AN platforms, the active code can communicate with each other using something like the Tuple-space in the Linda programming language.*

CORBA binding

PC7.) *Overhead of spawning a new thread: Often AN platforms see themselves as spawning new threads of control on remote systems. What is the cost of such an operation in terms of CPU cycles, memory usage, network usage etc?*

normally the same cost as incurred by JVM.

PC8.) *Overhead of mobile code isolation (runtime checking): Some platforms implement isolation of active code, by performing runtime checks on the memory addressing performed by the active code. What is the overhead of putting and getting information from and to the shared memory area?*

1) restrict processing resource control with Java security manager Java, overhead is bound with Java;

2) resource access control for managing router resources. overhead is minimal as policy is used and enforced in advance. Current isolation relies on the loyalty of users, i.e. they send traffic as specified in the traffic specification. To achieve real isolation requires metering of traffic and corresponding shaping function.

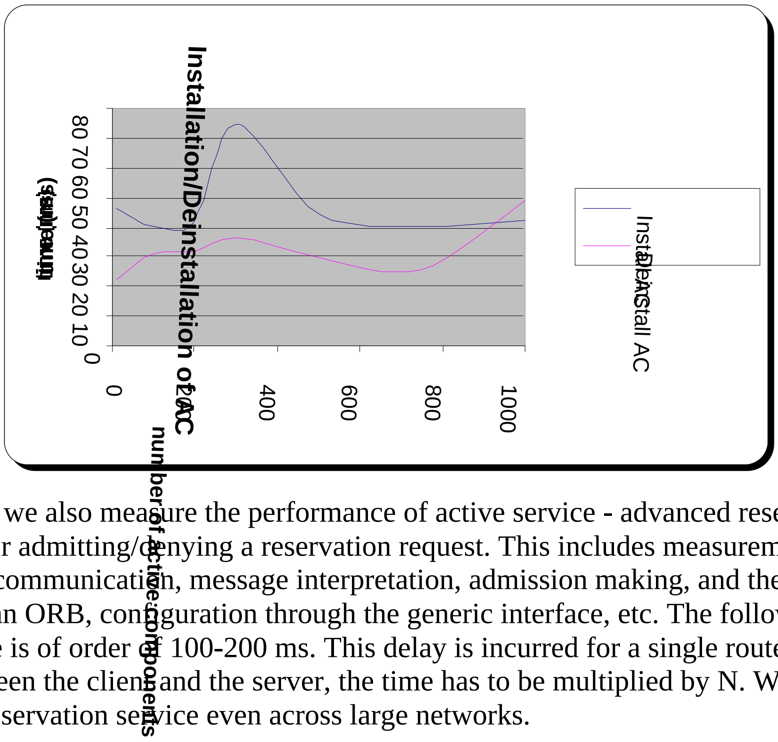
PC9.) *Code shipping vs. code caching: Or capsules vs. plug-ins, which approach is used? Can the platform support both approaches? What is the overhead (in terms of network usage, CPU usage, memory usage etc.) of one approach* *compared to the other - in the cases where the platform supports both modes? Can the platform automatically switch from one mode to the other. E.g., load a class on-demand and then cache it for future use.*

caching & shipping is supported by mobile agents & active component manager in DPE EE. plug-in is used. life cycle control is supported to smoothly change the state of code on demand. So it is naturally supported to load a class-on demand and cache for future use.

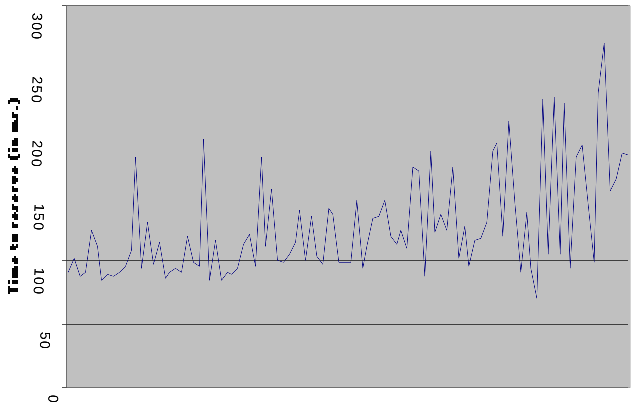
B.4.4 Performance Measurements

BANG deliverables D9, D10, D11 give full details. Here we just gives simple examples to show the performance implication of CORBA-based service implementation, and Java-based EE implementation.

We measured the performance for component management. The following figure depicts the performance of installing/deinstalling active code.



In BANG project, we also measure the performance of active service - advanced reservation, which is mainly the time for admitting/denying a reservation request. This includes measurement of the time used for binding, communication, message interpretation, admission making, and the overhead caused by Java or Jonathan ORB, configuration through the generic interface, etc. The following figure shows the time to reserve is of order of 100-200 ms. This delay is incurred for a single router. If there are N active nodes between the client and the server, the time has to be multiplied by N. We think it is acceptable for a reservation service even across large networks.



Another important measurement relates to the scalability of active service platform. Resource control framework develops a dynamic partitioning module and access control scheme to enlarge the scalability, i.e. to support as many service components as possible under certain resource constraints. The following figure shows the performance improvement of dynamic partitioning vs. static resource allocation. And it also helps to improve the performance of active service, e.g. the admission ratio for reservation requests.

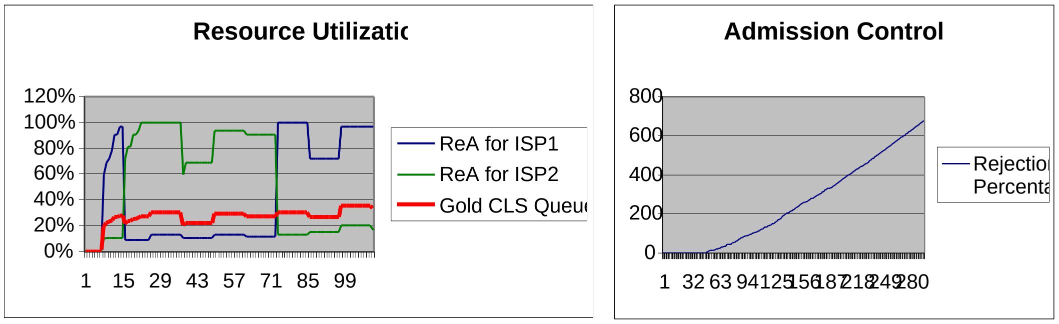


Figure 7 - Static Partitioning using Random Service Pattern for Advanced Reservation Services

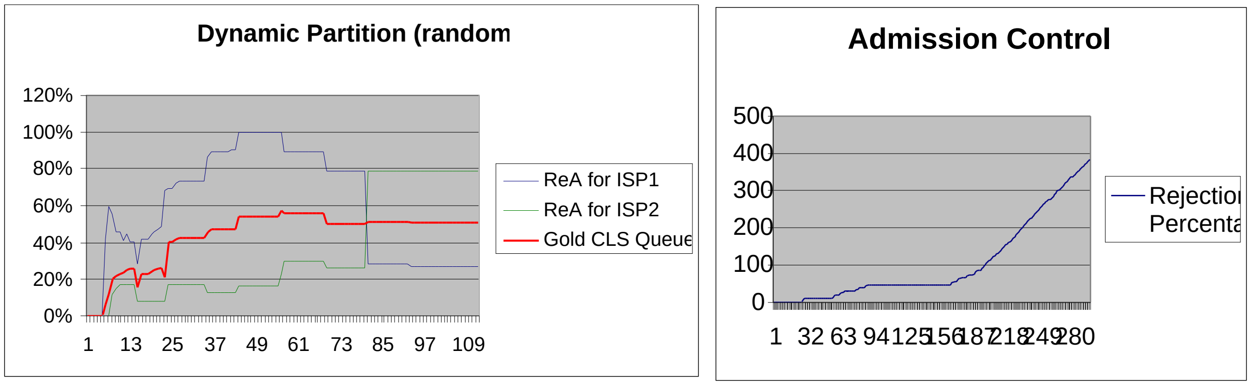


Figure 8 - Dynamic Partitioning using Random Service Pattern for Advanced Reservation Services

B.4.5 Security criteria

SC1.) *Cryptographic subsystem: Does the platform have such a subsystem? Can it provide cryptographic routines to the active code in the system?*

There is no cryptographic subsystem. Cryptographic routines can be provided by an BANG code extension

SC2.) *Authentication, Authorisation, Auditing: How does the AN platform perform authentication? Does it use digital signatures and certificates to authenticate any incoming code?*

certificate was used. Signature is planned in FAIN.

How is authorisation performed?

Authorisation is done in security policies. Code is stored in trusted code server.

Does the system authorise the code as if it was a user on the system?

in mobile agent case, yes.

How fine-grained is the privilege model in the platform?

There is no privilege model at all.

Does it use capabilities?

No.

Does the system provide logging facilities for auditing purposes?

Yes. in component manager package.

SC3.) *Language protection (SafetyNet, PLAN): Does the language used to write the active code that the platform processes provide any form of protection? In Java there is protection against accessing data beyond the bounds of an array, SafetyNet and PLAN are dedicated languages for programming ANs and they provide many language-based security features.*

BANG uses Java to write the active code and relies mainly on the security mechanisms of Java.

SC4.) *Sandboxing of EE: Does the EE run in a sandbox to protect the hosting node from being corrupted by malicious or buggy code?*

in mobile agent EE, Yes.

Does the EE run as 'root' on a UNIX machine?

No.

SC5.) *Protection of mobile code: How does the platform protect the active code executing on it?*

Java.

Does it provide isolation properties?

Nothing beyond the isolation properties provided by the Java virtual machine. It is clear that additional security mechanisms are necessary to ensure secrecy, integrity, and availability for all the users in the network. These mechanisms have not yet been implemented and are currently being investigated.

Does it provide code mangling (cf. Fritz Hohl - University of Stuttgart) or encrypted execution of functions (Sander & Tschudin - Berkeley University)?

No.

SC6.) *Protection of hosts: How does the platform protect itself from damage from malicious or buggy code?*

The NodeOS is protected from damage from the BANG platform, which runs in user space. DPE EE is protected by resource access control.

SC7.) *Security in ANs can be seen in 3 different levels (personal opinion: (a) EE-security, (b) Node Security, (c) Active Code-security. The question that arises is not only what is offered but also in which level and how far this security support goes in matter of configurability and openness (e.g. can I plug in my encryption algorithms or do I have to use only the provided ones?).*

security functions can be deployed as active code (plug-ins) and runs as active service with privileged access.

SC8.) *Secure communication/Confidentiality. Is it supported between AN nodes ? (e.g. with the usage of SSL/TLS or other proprietary protocol).*

SSL supported

Does the EE provide such services to the active code?

Yes

Does the NodeOS provide it also to EEs (for secure inter-EE communication)?

Java provides secure sockets. The current EE - Jonathan ORB doesn’t use secure sockets

SC9.) *Integrity. Is it guaranteed? Will modifications e.g. in the active code be detected ?*

no. planned in FAIN with signed code.

SC10.) *Accountability and Non-repudiation. Is the EE able to provide proof to third parties of the activities that took place?*

auditing is supported, however in a coarse granularity. The usability need to be studied.

SC11.) *Availability. Is there a way to ensure the availability of the basic services to the users even in case of denial of service attacks? E.g. QoS enabled connections between AN nodes, Economy-based resource consumption etc.*

The DoS isolation application enables this. However, no node-local function is engineered.

SC12.) *Authentication. What means are there to authenticate the active code? Digital Signature validation ?*

authentication is performed only on the user identity rather than the code.

Login/pass approach ?

Yes

SC13.) *Access Control & Authorisation. How is the access to node's resources mediated and controlled ?*

access control by security policy and resource policy. The control is enforced during runtime.

What policy system is used ? Is the policy syntax a standard one (e.g. Java policy syntax) or a EE-specific one ?

Java policy is extended with quantitative factors.

Runtime Authorisation or static ?

static.

SC14.) *EE/Node authentication. Is it possible to authenticate the EE or the AN node before visiting and executing in it (remotely from another node).*

Yes, but not implemented.

SC15.) *Is there a secure way to install/de-install/find/get active code and/or services to the AN node ?*

yes. by authorisation and authentication

SC16.) *Is there a way to revoke active code that travels within the AN infrastructure ?*

in principle, yes. But a better caching framework is needed.

SC17.) *Does the NodeOS provide sandboxing between EEs ?*

Java-based EE is fairly portable without restriction on Node OS:

SC18.) *Is there a network wide management of the security ? (e.g. network-wide policy change)*

No, planned in FAIN.

SC19.) *Is there a secure schema to ensure the survivability of Audit logs?*

Persistence can be supported.

SC20.) *Does the EE take any measures to provide safe code execution ? Are there any AN specific language-based techniques used (SafetyNet, PLAN) or general ones e.g. Java?*

BANG is based on Java.

SC21.) *What security services does the NodeOS provide to the EE it hosts ?*

Linux

What security services do the EEs provide to the active code that execute ? (e.g. Cryptographic support)

Java

SC22.) *Does active code take any countermeasures to preserve the privacy of its data, detect malicious nodes and unwanted modifications to its code ? (e.g. partial result encapsulation, execution tracing, computing with encrypted data etc ?)*

Nothing beyond Java.

SC23.) *Is there dedicated hardware used ? (e.g. smart cards)*

GR2000

SC24.) *What are the requirements of the AN components from the underlying infrastructure ? (e.g. administrator access is required for full resource handling.)*

Identity of operators of active service is required, and checked against resource policies. This is a mandatory requirement by resource control framework.

B.4.6 Flexibility criteria

FC1.) *Capsules vs. Plug-ins: how 'late' can we bind functionality? Capsules offer a larger degree of flexibility, how do plug-in platforms compensate? Can we mix both models of Active Networking?*

ANEP stack can be easily integrated to support capsules. Plug-ins are supported now.

FC2.) *At what layer can we program down to? In software radios we can reprogram all the way down to the link-layer*

Currently, layer3-7

FC3.) *How restricted is the behaviour of the mobile code? (related to security)*

See SC3 and others.

FC4.) *Sandboxing: Does the platform support sandboxing, if so how restrictive is it?*

See SC3 and others.

FC5.) *Language-based control (e.g. static typing): How does this restrict our expressiveness with the language?*

No restrictions

FC6.) *Backwards compatibility: Can different versions of the platform inter-operate with other versions?*

No.

FC7.) *What technology is used for installation/de-installation of services (e.g. mobile agents, RPC-paradigm etc)*

mobile agents and component manager by binding.

FC8.) *Is the system scalable and in what levels.*

scalable on the level of the number of runtime service instance.

B.4.7 Miscellaneous

MC1.) *Is the interoperability issue touched ? What foreign EEs can the platform host ?*

CORBA is expected to provide fundamental interoperability on the communication among active service/code; IEEE P1520 L interface aims for the hardware, i.e. router. so that active services can be used to control hardware from different vendors.

MC2.) *Can the platform inter-operate with test bed platforms e.g., ABONE and ANON?*

Not tested. to be explored in FAIN.

MC3.) Robustness & fault tolerance ?

persistence can be integrated, if this is a key goal of FAIN

MC4.) *Performance of the system. Does it require special hardware support for the "best-performance" configuration?*

focus on control & mgmt. performance is acceptable. No special hardware support. Maybe for metering purpose, this is needed. But this depends on the performance requirements

MC5.) *Which Software License is the code published under? (E.g., GPL, MPL, BSD etc.)*

license of the OrbAB platform is up to Hitachi, Grasshopper is free, and Jonathan ORB is open source and release via GNU-GPL.

MC6.) *Code availability (E.g., can the code be downloaded now?)*

depends on Hitachi decision

MC7.) *Code maturity (Is the code release quality? Beta? Alpha?)*

research prototype.

MC8.) *Is the code actively maintained?*

Probably yes in FAIN

MC9.) *What are the dependencies of the code? Does it require Java? A specialised NodeOS?*

Java, no special NodeOS

MC10.) *What features does the NodeOS provide?*

-

MC11.) *Which NodeOS API does the platform match, if at all?*

Nothing particular.

B.4.8 Conclusion

BANG is a conceptually very interesting project. Its implementation provides a suitable test bed and serves very well as proof of concept. Its goal, architecture and design is quite aligned with FAIN. It can be a good test platform for investigating AN on control & management plane, and integration with other EE. The following features of BANG platform are particularly interesting for FAIN:

- distributed object-oriented platform for active services
- focus on dynamic service provision and policy-based resource management
- integration of modules in forwarding, control & management planes
- fine grained resource access control
- lifecycle control of service components

B.5 JanOS evaluation

B.5.1 Performance Criteria

PC1.) *CPU cycles for packet forwarding (active & data): some Active Network platforms (e.g. MBASH) use their own packet format for data packets (SAPF) which results in faster forwarding speeds than traditional IP forwarding. How many cycles does it take to forward an active packet versus a data packet in the platform?*

PC2.) *Packet size (active & data): If the platform uses special packet formats (see below) instead of running over UDP/IP for example, then what are the comparative sizes of these formats*

PC3.) *Packet format (ANEP, SAPF etc.): Active Network Encapsulation Protocol (ANEP) and Simple Active Packet Format (SAPF) are examples of active packet formats used by various projects to identify packets containing active code and/or to handle data packets without the overhead caused by TCP/UDP and IP.*

Janos uses UDP/IP or Ethernet channels both with ANEP v1.0.

PC4.) *Implementation language of runtime/execution environment: If the runtime/EE is written in Java or another interpreted language, then it will incur a performance penalty, whilst allowing greater portability.*

Runtime is implemented in JAVA. EE can be written in C.

PC5.) *Implementation language of Node OS: In the cases where a dedicated Node OS is used, then the same notes as the above point apply.*

NodeOS is implemented in C or parts of it in Java. In Java or C are implemented most parts of the NodeOS interface specification specified in [1].

PC6.) *Overhead of inter-active code communication: In some AN platforms, the active code can communicate with each other using something like the Tuple-space in the Linda programming language.*

Depends on EE and/or AC implementation. For example in KaffeOS, see [2], there exist share spaces but it is not stated in the paper how much is overhead of inter-active code communication. Software is not yet available.

PC7.) *Overhead of spawning a new thread: Often AN platforms see themselves as spawning new threads of control on remote systems. What is the cost of such an operation in terms of CPU cycles, memory usage, network usage etc?*

PC8.) *Overhead of mobile code isolation (runtime checking): Some platforms implement isolation of active code, by performing runtime checks on the memory addressing performed by the active code. What is the overhead of putting and getting information from and to the shared memory area?*

Mobile code is isolated in JanosVM (KaffeOS, isolation per process). Penalty is cca. 2-8% regarding to normal execution (SpecJVM98 benchmarks, compared to normal execution in Kaffe), see [2]. Same can be possibly achieved in implementation in C with Flask security architecture, see [3]. Flask is based on Fluke, with strong emphasis on processes separation, see [4] as one of the final reports. Results of both project should be available also in security-enhanced Linux. Software is not yet available.

PC9.) *Code shipping vs. code caching: Or capsules vs. plugins, which approach is used? Can the platform support both approaches? What is the overhead (in terms of network usage, CPU usage, memory usage etc.) of one approach compared to the other - in the cases where the* *platform supports both modes? Can the platform automatically switch from one mode to the other. E.g., load a class on-demand and then cache it for future use.*

Only supported EE is ANTS; so at the moment platform supports capsules and caching. Anything else is possible, also something like plugins in C (like separated EE with only one functionality but with standard NodeOS interface as defined in [1], its functionality is linked to the program as a library) or Java.

B.5.2 Security Criteria

SC1.) *Cryptographic subsystem: Does the platform have such a subsystem? Can it provide cryptographic routines to the active code in the system?*

There are no special cryptographic subsystem in Janos; but there is no reason that they cannot be added in software or hardware, in same manner as to any Linux system.

SC2.) *Authentication, Authorisation, Auditing: How does the AN platform perform authentication? Does it use digital signatures and certificates to authenticate any incoming code? How is authorisation performed? Does the system authorise the code as if it was a user on the system? How fine-grained is the privilege model in the platform? Does it use capabilities? Does the system provide logging facilities for auditing purposes?*

SC3.) *Language protection (SafetyNet, PLAN): Does the language used to write the active code that the platform processes provide any form of protection? In Java there is protection against accessing data beyond the bounds of an array, SafetyNet and PLAN are dedicated languages for programming ANs and they provide many language-based security features.*

Janos can use Java security features.

SC4.) *Sandboxing of EE: Does the EE run in a sandbox to protect the hosting node from being corrupted by malicious or buggy code? Does the EE run as 'root' on a UNIX machine?*

No, but it can be done. In KaffeOS, Java code or threads can run in VM separated like as Unix processes and they can be tightly controlled.

SC5.) *Protection of mobile code: How does the platform protect the active code executing on it? Does it provide isolation properties? Does it provide code mangling (cf. Fritz Hohl - University of Stuttgart) or encrypted execution of functions (Sander & Tschudin - Berkeley University)?*

The right question is if the code can be protected from EE or/and NodeOS. No.

SC6.) *Protection of hosts: How does the platform protect itself from damage from malicious or buggy code?*

KaffeOS can protect its resources somehow better from buggy or malicious code than other VM because of processes separation and running more processes in a single VM, see [2,page 10] for example.

SC7.) *Security in ANs can be seen in 3 different levels (personal opinion).*

- *EE-security,*
- *Node Security,*
- *ActiveCode-security*

The question that arises is not only what is offered but also in which level and how far this security support goes in matter of configurability and openness (e.g. can I plug in my encryption algorithms or do I have to use only the provided ones?).

Plus levels: active network and language security.

SC8.) *Secure communication/Confidentiality. Is it supported between AN nodes ? (e.g. with the usage of SSL/TLS or other proprietary protocol). Does the EE provide such services to the active code? Does the NodeOS provide it also to EEs (for secure inter-EE communication)?*

There is no secure communication between ANNs or between EEs. To be checked.

SC9.) *Integrity. Is it guaranteed? Will modifications e.g. in the active code be detected ?*

No.

SC10.) *Accountability and Non-repudiation. Is the EE able to provide proof to third parties of the activities that took place?*

No.

SC11.) *Availability. Is there a way to ensure the availability of the basic services to the users even in case of denial of service attacks? E.g. QoS enabled connections between AN nodes, Economy-based resource consumption etc.*

KaffeOS, if used, can offer some good properties in case of DoS.

SC12.) *Authentication. What means are there to authenticate the active code? Digital Signature validation ? Login/pass approach ?*

No. Can depend on EE. See ANTS [5].

SC13.) *Access Control & Authorisation. How is the access to node's resources mediated and controlled ? What policy system is used ? Is the policy syntax a standard one (e.g. Java policy syntax) or a EE-specific one ? Runtime Authorisation or static ?*

SC14.) *EE/Node authentication. Is it possible to authenticate the EE or the AN node before visiting and executing in it (remotely from another node).*

No.

SC15.) *Is there a secure way to install/deinstall/find/get active code and/or services to the AN node ?*

No.

SC16.) *Is there a way to revoke active code that travels within the AN infrastructure ?*

No.

SC17.) Does the NodeOS provide sandboxing between EEs ?

SC18.) *Is there a network wide management of the security ? (e.g. network-wide policy change)*

No.

SC19.) *Is there a secure schema to ensure the survivability of Audit logs?*

No.

SC20.) *Does the EE take any measures to provide safe code execution ? Are there any AN specific language-based techniques used (SafetyNet, PLAN) or general ones e.g. Java?*

Only general one, JAVA, possibly in future in the OS (Flask, security-enhanced Linux).

SC21.) *What security services does the NodeOS provide to the EE it hosts ? What security services do the EEs provide to the active code that execute ? (e.g. Cryptographic support)*

Services related to security defined in [1]. It seems there is no intention in short term to support something like security spec described in [6].

SC22.) *Does active code take any countermeasures to preserve the privacy of its data, detect malicious nodes and unwanted modifications to its code ? (e.g. partial result encapsulation, execution tracing, computing with encrypted data etc ?)*

SC23.) *Is there dedicated hardware used ? (e.g. smartcards)*

No, but there is no reason why they cannot be used.

SC24.) *What are the requirements of the AN components from the underlying infrastructure ? (e.g. administrator access is required for full resource handling.)*

B.5.3 Flexibility Criteria

FC1.) *Capsules vs. Plugins: how 'late' can we bind functionality? Capsules offer a larger degree of flexibility, how do plugin platforms compensate? Can we mix both models of Active Networking?*

Both models can be used.

FC2.) *At what layer can we program down to? In software radios we can reprogram all the way down to the link-layer*

From link layer up.

FC3.) *How restricted is the behaviour of the mobile code? (related to security)*

FC4.) *Sandboxing: Does the platform support sandboxing, if so how restrictive is it?*

FC5.) *Language-based control (e.g. static typing): How does this restrict our expressiveness with the language?*

FC6.) *Backwards compatibility: Can different versions of the platform inter-operate with other versions?*

To early to say, but big plus is NodeOS interface spec support.

FC7.) *What technology is used for installation/deinstallation of services (e.g. mobile agents, RPC-paradigm etc)*

FC8.) *Is the system scalable and in what levels.*

No data, except the same as for EE used (for example ANTS, see [5]).

B.5.4 Miscellaneous

MC1.) *Is the interoperability issue touched ? What foreign EEs can the platform host ?*

It seems that the platform can host many EEs.

MC2.) *Can the platform inter-operate with testbed platforms e.g., ABONE and ANON?*

At least with Abone at some extent.

MC3.) *Robustness & fault tolerance ?*

MC4.) *Performance of the system. Does it require special hardware support for the "best-performance" configuration?*

No real performance data, no special hardware required.

MC5.) *Which Software License is the code published under? (E.g., GPL, MPL, BSD etc.)*

GPL.

MC6.) *Code availability (E.g., can the code be downloaded now?)*

Some code yes and some code not. JanosVM (KaffeOS?) and security-enchanced Linux are not available. Moab (C implementation of the [1]) and Java code implementing the same are available.

MC7.) *Code maturity (Is the code release quality? Beta? Alpha?)*

Alpha. But the code does at the moment exactly what is intended to do. Projects are enhancing steadily.

MC8.) *Is the code actively maintained?*

Yes.

MC9.) *What are the dependencies of the code? Does it require Java? A specialised NodeOS?*

The Code is dependent and independent at the same time. Available code can run on Linux, FreeBSD, Solaris or on the OSkit. There is not quite clear what does dependence on the OSkit means. For example, we can see in some slides that security features will be implemented on the OSkit level but on the other hand they will be possibly implemented also in security-enhanced Linux or the KaffeOS. Also performance issues are not clear in this sense. Some opinions were also given that it is better to start prototyping with the OSkit and than develop your own set of libraries when appropriate. Penn can possibly have more information on this (SANE). Security features in Java depends on availability of the KaffeOS.

MC10.) *What features does the NodeOS provide?*

A large subset of NodeOS interface spec, see [1].

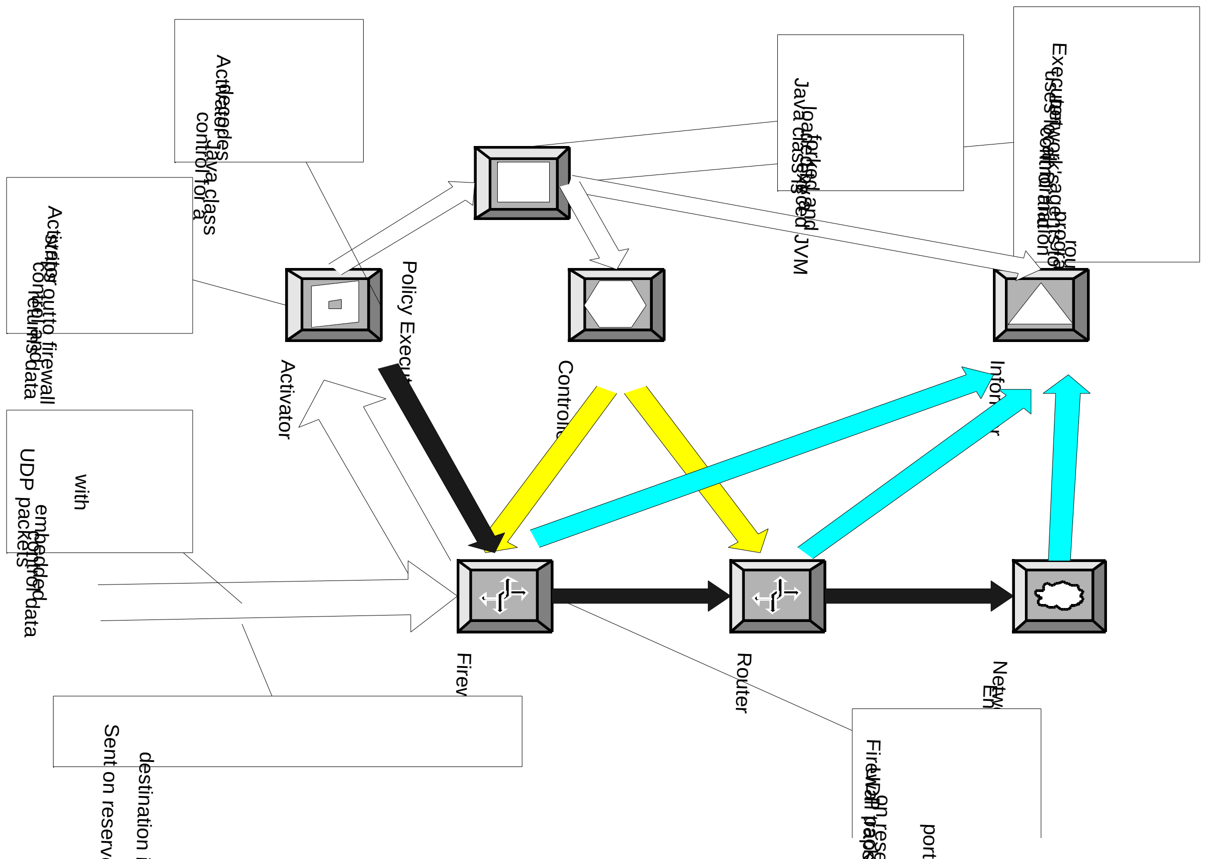
MC11.) *Which NodeOS API does the platform match, if at all?*

Main platform target is to match the NodeOS interface as defined in [1].

B.6 ABLE evaluation

B.6.1 Concepts and Architecture

The ABLE Active Network platform is a programmable platform, which links a legacy router with an Active Network platform where active code can be executed to alter the behaviour of the router. The EE is based on a Java VM which can talk to the legacy router via SNMP. Incoming active packets containing Java code are diverted by the legacy router to the Active Engine (EE). The ABLE architecture is as follows:



The ABLE system has the concept of a "session". This is a soft-state mechanism that enables long lasting applications typically associated with network management to have some state in the network devices. Everytime that a session wishes to open a connection it must register with the security stream module so that the session can monitor the network usage of its streams. The connections opened can either be explicitly addressed to a particular machine or they can use the "blind addressing" mode of ABLE where packets can just be sent to local active nodes, leveraging off of the active node network architecture.

The ABLE architecture distributes Java active code using ANEP over UDP/IP, this enables the system to support both the capsule paradigm and the plugin paradigm through the session soft-state mechanism.

B.6.2 Performance Criteria

PC1.) *CPU cycles for packet forwarding (active & data): some Active Network platforms (e.g. MBASH) use their own packet format for data packets (SAPF) which results in faster forwarding speeds than traditional IP forwarding. How many cycles does it take to forward an active packet versus a data packet in the platform?*

PC2.) *Packet size (active & data): If the platform uses special packet formats (see below) instead of running over UDP/IP for example, then what are the comparative sizes of these formats?*

ABLE is running over UDP/IP.

PC3.) *Packet format (ANEP, SAPF etc.): Active Network Encapsulation Protocol (ANEP) and Simple Active Packet Format (SAPF) are examples of active packet formats used by various projects to identify packets containing active code and/or to handle data packets without the overhead caused by TCP/UDP and IP.*

ANEP Version 1.

PC4.) *Implementation language of runtime/execution environment: If the runtime/EE is written in Java or another interpreted language, then it will incur a performance penalty, whilst allowing greater portability.*

Java VM written in C + assembly language.

PC5.) *Implementation language of Node OS: In the cases where a dedicated Node OS is used, then the same notes as the above point apply.*

There is no dedicated NodeOS. FreeBSD is implemented in C.

PC6.) *Overhead of inter-active code communication: In some AN platforms, the active code can communicate with each other using something like the Tuple-space in the Linda programming language.*

Sessions can communicate with each other via a broker, this incurs a degree of overhead.

PC7.) *Overhead of spawning a new thread: Often AN platforms see themselves as spawning new threads of control on remote systems. What is the cost of such an operation in terms of CPU cycles, memory usage, network usage etc?*

There is no particular concept for spawning new threads on remote systems.

PC8.) *Overhead of mobile code isolation (runtime checking): Some platforms implement isolation of active code, by performing runtime checks on the memory addressing performed by the active code. What is the overhead of putting and getting information from and to the shared memory area?*

There are no additional checks beyond the checks done by the Java virtual machine.

PC9.) *Code shipping vs. code caching: Or capsules vs. plug-ins, which approach is used? Can the platform support both approaches? What is the overhead (in terms of network usage, CPU usage, memory usage etc.) of one approach compared to the other - in the cases where the platform supports both modes? Can the platform automatically switch from one mode to the other. E.g., load a class on-demand and then cache it for future use.*

ABLE supports both the capsule and the plugin programming models. The system can easily switch between the two modes by the user specifying whether they wish to create a soft-state session, which has more persistence than an ephemeral capsule session.

B.6.3 Security Criteria

SC1.) *Cryptographic subsystem: Does the platform have such a subsystem? Can it provide cryptographic routines to the active code in the system?*

There is no cryptographic subsystem.

SC2.) *Authentication, Authorisation, Auditing: How does the AN platform perform authentication? Does it use digital signatures and certificates to authenticate any incoming code?*

There is no authentication.

How is authorisation performed?

There is a limit imposed by the brokers and the Java VM as to what a piece of code can perform.

Does the system authorise the code as if it was a user on the system?

No.

How fine-grained is the privilege model in the platform?

There is no privilege model at all.

Does it use capabilities?

No.

Does the system provide logging facilities for auditing purposes?

No.

SC3.) *Language protection (SafetyNet, PLAN): Does the language used to write the active code that the platform processes provide any form of protection? In Java there is protection against accessing data beyond the bounds of an array, SafetyNet and PLAN are dedicated languages for programming ANs and they provide many language-based security features.*

ABLE active code is Java code and it is subject to all the restrictions of the Java language.

SC4.) *Sandboxing of EE: Does the EE run in a sandbox to protect the hosting node from being corrupted by malicious or buggy code?*

Unix respective Java mechanisms.

Does the EE run as 'root' on a UNIX machine?

N\A.

SC5.) *Protection of mobile code: How does the platform protect the active code executing on it?*

Java.

Does it provide isolation properties?

Nothing beyond the isolation properties provided by the Java virtual machine.

Does it provide code mangling (cf. Fritz Hohl - University of Stuttgart) or encrypted execution of functions (Sander & Tschudin - Berkeley University)?

No.

SC6.) *Protection of hosts: How does the platform protect itself from damage from malicious or buggy code?*

The NodeOS is protected from damage from the ABLE platform or from active code by the fact that both run as user level programs under the Unix operating system.

The ABLE platform is protected from damage from malicious or buggy capsule code only by the mechanisms provided by Java.

SC7.) *Security in ANs can be seen in 3 different levels (personal opinion: (a) EE-security, (b) Node Security, (c) Active Code-security. The question that arises is not only what is offered but also in which level and how far this security support goes in matter of configurability and openness (e.g. can I plug in my encryption algorithms or do I have to use only the provided ones?).*

A user can use the Java Cryptographic Extensions to extend the cryptographic capabilities of the ABLE system.

SC8.) *Secure communication/Confidentiality. Is it supported between AN nodes ? (e.g. with the usage of SSL/TLS or other proprietary protocol).*

No.

Does the EE provide such services to the active code?

No.

Does the NodeOS provide it also to EEs (for secure inter-EE communication)?

No.

SC9.) *Integrity. Is it guaranteed? Will modifications e.g. in the active code be detected ?*

No.

SC10.) *Accountability and Non-repudiation. Is the EE able to provide proof to third parties of the activities that took place?*

No.

SC11.) *Availability. Is there a way to ensure the availability of the basic services to the users even in case of denial of* *service attacks? E.g. QoS enabled connections between AN nodes, Economy-based resource consumption etc.*

No.

SC12.) *Authentication. What means are there to authenticate the active code? Digital Signature validation ?*

No.

Login/pass approach ?

No.

SC13.) *Access Control & Authorisation. How is the access to node's resources mediated and controlled ?*

The ABLE platform uses the broker system to control access to the router resources. If a session takes a large portion of the system resources or if the session does not show activity within an ageing period it will be removed in order to protect the system.

What policy system is used ? Is the policy syntax a standard one (e.g. Java policy syntax) or a EE-specific one ?

There is no policy system and therefore no policy syntax.

Runtime Authorisation or static ?

There is no authorisation at all, neither static nor runtime.

SC14.) *EE/Node authentication. Is it possible to authenticate the EE or the AN node before visiting and executing in it (remotely from another node).*

No.

SC15.) *Is there a secure way to install/de-install/find/get active code and/or services to the AN node ?*

No.

SC16.) *Is there a way to revoke active code that travels within the AN infrastructure ?*

No.

SC17.) *Does the NodeOS provide sandboxing between EEs ?*

ABLE provides standard Java sandboxing and isolation. If two ABLE platforms are being run on the same node then they will run in separate Java VMs for isolation.

SC18.) *Is there a network wide management of the security ? (e.g. network-wide policy change)*

No.

SC19.) *Is there a secure schema to ensure the survivability of Audit logs?*

There is no such schema.

SC20.) *Does the EE take any measures to provide safe code execution ? Are there any AN specific language-based techniques used (SafetyNet, PLAN) or general ones e.g. Java?*

ABLE Active Engine is a Java VM.

SC21.) *What security services does the NodeOS provide to the EE it hosts ?*

Its Unix; see SC7.

What security services do the EEs provide to the active code that execute ? (e.g. Cryptographic support)

None.

SC22.) *Does active code take any countermeasures to preserve the privacy of its data, detect malicious nodes and unwanted modifications to its code ? (e.g. partial result encapsulation, execution tracing, computing with encrypted data etc ?)*

Nothing beyond Java.

SC23.) *Is there dedicated hardware used ? (e.g. smart cards)*

No.

SC24.) *What are the requirements of the AN components from the underlying infrastructure ? (e.g. administrator access is required for full resource handling.)*

No.

B.6.4 Flexibility Criteria

FC1.) *Capsules vs. Plug-ins: how 'late' can we bind functionality? Capsules offer a larger degree of flexibility, how do plug-in platforms compensate? Can we mix both models of Active Networking?*

ABLE supports both paradigms for maximum flexibility.

FC2.) *At what layer can we program down to? In software radios we can reprogram all the way down to the link-layer*

ABLE is based on the usage of standard protocols as a transport mechanism for capsules (ANEP over UDP/IP).

FC3.) *How restricted is the behaviour of the mobile code? (related to security)*

See SC3 and others.

FC4.) *Sandboxing: Does the platform support sandboxing, if so how restrictive is it?*

See SC3 and others.

FC5.) *Language-based control (e.g. static typing): How does this restrict our expressiveness with the language?*

Nothing beyond Java.

FC6.) *Backwards compatibility: Can different versions of the platform inter-operate with other versions?*

This issue has not so far been addressed.

FC7.) *What technology is used for installation/de-installation of services (e.g. mobile agents, RPC-paradigm etc)*

Mobile Agents paradigm.

FC8.) *Is the system scalable and in what levels.*

The system is scalable. There are no obvious architectural bottlenecks.

B.6.5 Miscellaneous

MC1.) *Is the interoperability issue touched ? What foreign EEs can the platform host ?*

ABLE does not interoperate with other active network platforms.

MC2.) *Can the platform inter-operate with test bed platforms e.g., ABONE and ANON?*

No.

MC3.) Robustness & fault tolerance ?

Not addressed

MC4.) *Performance of the system. Does it require special hardware support for the "best-performance" configuration?*

No. Although the system was tested on Cisco and Lucent routers, no “best-performance” configuration is mentioned.

MC5.) *Which Software License is the code published under? (E.g., GPL, MPL, BSD etc.)*

N\A.

MC6.) *Code availability (E.g., can the code be downloaded now?)*

N\A.

MC7.) *Code maturity (Is the code release quality? Beta? Alpha?)*

University prototype of good quality.

MC8.) *Is the code actively maintained?*

No. ANTS is no product, thus maintenance depends on the intentions of the authors. See section B.1.4.2.

MC9.) *What are the dependencies of the code? Does it require Java? A specialised NodeOS?*

Java and Unix.

MC10.) *What features does the NodeOS provide?*

Whatever features Unix does provide.

MC11.) *Which NodeOS API does the platform match, if at all?*

Nothing particulars.

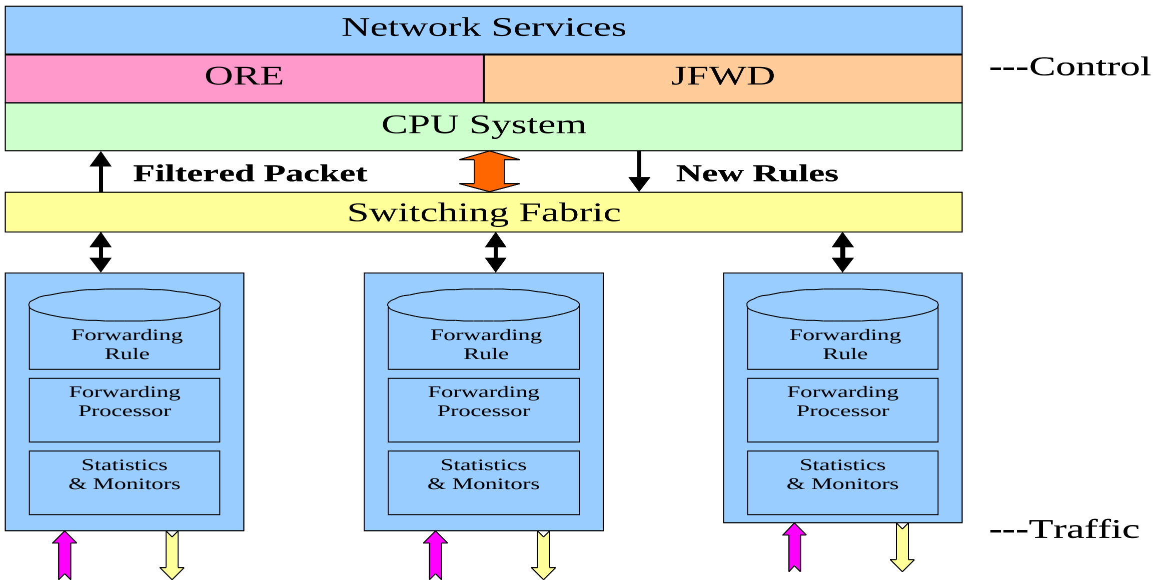
B.7 OPENET architecture

B.7.1 Concepts and Architecture

The primary goal of the Openet project is to build a platform for implementing programmable services on a commercial router. The project aimed to achieve the goals of both high performance and programmability. High performance was addressed by preserving the router hardware fast path for datagram forwarding and programmability was achieved by using APIs into the router so that active code could control the router to provide services.

The Openet project is based upon the Accelar layer 3 routing switch., which uses application-specific integrated circuits (ASICs) to provide fast path forwarding functionality. Such an architecture differentiates between the control plane and the forwarding plane, thus allowing the system to perform data path forwarding at line speeds.

The Accelar architecture is as follows:



The ORE (Oplet Runtime Environment) part of the Openet system is the substrate for downloading, installing, executing and managing networking services. Oplets, in the context of the Openet system are self-contained downloadable units that embody a set of services.

B.7.2 Performance Criteria

PC1.) *CPU cycles for packet forwarding (active & data): some Active Network platforms (e.g. MBASH) use their own packet format for data packets (SAPF) which results in faster forwarding speeds than traditional IP forwarding. How many cycles does it take to forward an active packet versus a data packet in the platform?*

Packet forwarding is implemented in hardware, thus CPU cycles are not consumed. Forwarding is performed at Gigabit line speeds.

PC2.) *Packet size (active & data): If the platform uses special packet formats (see below) instead of running over UDP/IP for example, then what are the comparative sizes of these formats?*

Openet is running over IPv4, therefore it is limited by the IPv4 specification.

PC3.) *Packet format (ANEP, SAPF etc.): Active Network Encapsulation Protocol (ANEP) and Simple Active Packet Format (SAPF) are examples of active packet formats used by various projects to identify packets containing active code and/or to handle data packets without the overhead caused by TCP/UDP and IP.*

IPv4.

PC4.) *Implementation language of runtime/execution environment: If the runtime/EE is written in Java or another interpreted language, then it will incur a performance penalty, whilst allowing greater portability.*

Java

PC5.) *Implementation language of Node OS: In the cases where a dedicated Node OS is used, then the same notes as the above point apply.*

The NodeOS is VxWorks, it is implemented in C.

PC6.) *Overhead of inter-active code communication: In some AN platforms, the active code can communicate with each other using something like the Tuple-space in the Linda programming language.*

Services in Openet can be comprised of multiple pieces of active code which are controlled by the ORE.

PC7.) *Overhead of spawning a new thread: Often AN platforms see themselves as spawning new threads of control on remote systems. What is the cost of such an operation in terms of CPU cycles, memory usage, network usage etc?*

There is no particular concept for spawning new threads on remote systems.

PC8.) *Overhead of mobile code isolation (runtime checking): Some platforms implement isolation of active code, by performing runtime checks on the memory addressing performed by the active code. What is the overhead of putting and getting information from and to the shared memory area?*

The Java VM provides runtime checking of code.

PC9.) *Code shipping vs. code caching: Or capsules vs. plug-ins, which approach is used? Can the platform support both approaches? What is the overhead (in terms of network usage, CPU usage, memory usage etc.) of one approach compared to the other - in the cases where the platform supports both modes? Can the platform automatically switch from one mode to the other. E.g., load a class on-demand and then cache it for future use.*

Openet currently only supports the plugin approach.

B.7.3 Security Criteria

SC1.) *Cryptographic subsystem: Does the platform have such a subsystem? Can it provide cryptographic routines to the active code in the system?*

There is a Public Key Infrastructure, which is used to protect the integrity of the system.

SC2.) *Authentication, Authorisation, Auditing: How does the AN platform perform authentication? Does it use digital signatures and certificates to authenticate any incoming code?*

Oplets carry a digital signature.

How is authorisation performed?

By the Java VM.

Does the system authorise the code as if it was a user on the system?

No.

How fine-grained is the privilege model in the platform?

There is no privilege model at all.

Does it use capabilities?

No.

Does the system provide logging facilities for auditing purposes?

The system provides logging facilities.

SC3.) *Language protection (SafetyNet, PLAN): Does the language used to write the active code that the platform processes provide any form of protection? In Java there is protection against accessing data beyond the bounds of an array, SafetyNet and PLAN are dedicated languages for programming ANs and they provide many language-based security features.*

Language safety is punted to the Java programming language, therefore it has all the safety properties of the Java language.

SC4.) *Sandboxing of EE: Does the EE run in a sandbox to protect the hosting node from being corrupted by malicious or buggy code?*

Traditional Java mechanisms.

Does the EE run as 'root' on a UNIX machine?

No.

SC5.) *Protection of mobile code: How does the platform protect the active code executing on it?*

Java.

Does it provide isolation properties?

Nothing beyond the isolation properties provided by the Java virtual machine.

Does it provide code mangling (cf. Fritz Hohl - University of Stuttgart) or encrypted execution of functions (Sander & Tschudin - Berkeley University)?

No.

SC6.) *Protection of hosts: How does the platform protect itself from damage from malicious or buggy code?*

The ORE controls thread spawning to prevent Denial of Service attacks.

SC7.) *Security in ANs can be seen in 3 different levels (personal opinion: (a) EE-security, (b) Node Security, (c) Active Code-security. The question that arises is not only what is offered but also in which level and how far this security support goes in matter of configurability and openness (e.g. can I plug in my encryption algorithms or do I have to use only the provided ones?).*

Theoretically the system can be made pluggable with the usage of the Java Cryptographic Extensions (JCE).

SC8.) *Secure communication/Confidentiality. Is it supported between AN nodes ? (e.g. with the usage of SSL/TLS or other proprietary protocol).*

No.

Does the EE provide such services to the active code?

No.

Does the NodeOS provide it also to EEs (for secure inter-EE communication)?

No.

SC9.) *Integrity. Is it guaranteed? Will modifications e.g. in the active code be detected ?*

Openet uses cryptographic mechanisms to derive a signature from the code.

SC10.) *Accountability and Non-repudiation. Is the EE able to provide proof to third parties of the activities that took place?*

Yes. Openet provides accountability and non-repudiation through its PKI.

SC11.) *Availability. Is there a way to ensure the availability of the basic services to the users even in case of denial of service attacks? E.g. QoS enabled connections between AN nodes, Economy-based resource consumption etc.*

As mentioned previously, the ORE controls thread spawning to ensure availability.

SC12.) *Authentication. What means are there to authenticate the active code? Digital Signature validation ?*

Digital Signatures are used.

Login/pass approach ?

No.

SC13.) *Access Control & Authorisation. How is the access to node's resources mediated and controlled ?*

Openet provides per-service resource controls.

What policy system is used ? Is the policy syntax a standard one (e.g. Java policy syntax) or a EE-specific one ?

There is no policy system and therefore no policy syntax.

Runtime Authorisation or static ?

There is no authorisation at all, neither static nor runtime.

SC14.) *EE/Node authentication. Is it possible to authenticate the EE or the AN node before visiting and executing in it (remotely from another node).*

No.

SC15.) *Is there a secure way to install/de-install/find/get active code and/or services to the AN node ?*

Service revocation etc. is managed by the ORE.

SC16.) *Is there a way to revoke active code that travels within the AN infrastructure ?*

Yes, through the ORE interface.

SC17.) *Does the NodeOS provide sandboxing between EEs ?*

The ORE controls EE instantiation and thus provides sandboxing between the nodes.

SC18.) *Is there a network wide management of the security ? (e.g. network-wide policy change)*

No.

SC19.) *Is there a secure schema to ensure the survivability of Audit logs?*

There is no secure schema, although there are logging capabilities.

SC20.) *Does the EE take any measures to provide safe code execution ? Are there any AN specific language-based techniques used (SafetyNet, PLAN) or general ones e.g. Java?*

Openet uses Java for safe code execution.

SC21.) *What security services does the NodeOS provide to the EE it hosts ?*

Security services are provided through Java.

What security services do the EEs provide to the active code that execute ? (e.g. Cryptographic support)

None.

SC22.) *Does active code take any countermeasures to preserve the privacy of its data, detect malicious nodes and unwanted modifications to its code ? (e.g. partial result encapsulation, execution tracing, computing with encrypted data etc ?)*

Nothing beyond Java.

SC23.) *Is there dedicated hardware used ? (e.g. smart cards)*

No.

SC24.) *What are the requirements of the AN components from the underlying infrastructure ? (e.g. administrator access is required for full resource handling.)*

No.

B.7.4 Flexibility Criteria

FC1.) *Capsules vs. Plug-ins: how 'late' can we bind functionality? Capsules offer a larger degree of flexibility, how do plug-in platforms compensate? Can we mix both models of Active Networking?*

Openet currently only supports the plugin programming model, although apparently ANTS has been ported to Openet, thus facilitating the usage of the capsule based approach.

FC2.) *At what layer can we program down to? In software radios we can reprogram all the way down to the link-layer*

Network Layer 3.

FC3.) *How restricted is the behaviour of the mobile code? (related to security)*

See SC3 and others.

FC4.) *Sandboxing: Does the platform support sandboxing, if so how restrictive is it?*

See SC3 and others.

FC5.) *Language-based control (e.g. static typing): How does this restrict our expressiveness with the language?*

No restrictions (see SC3).

FC6.) *Backwards compatibility: Can different versions of the platform inter-operate with other versions?*

N\A.

FC7.) *What technology is used for installation/de-installation of services (e.g. mobile agents, RPC-paradigm etc)*

Java-based techniques.

FC8.) *Is the system scalable and in what levels.*

No obvious scalability problems.

B.7.5 Miscellaneous

MC1.) *Is the interoperability issue touched ? What foreign EEs can the platform host ?*

Openet can host any Java-based EE.

MC2.) *Can the platform inter-operate with test bed platforms e.g., ABONE and ANON?*

Yes, e.g., ABONE.

MC3.) Robustness & fault tolerance ?

Not in the scope of the project.

MC4.) *Performance of the system. Does it require special hardware support for the "best-performance" configuration?*

Requires a Nortel Accelar Switch to perform correctly.

MC5.) *Which Software License is the code published under? (E.g., GPL, MPL, BSD etc.)*

N\A.

MC6.) *Code availability (E.g., can the code be downloaded now?)*

N\A.

MC7.) *Code maturity (Is the code release quality? Beta? Alpha?)*

N\A.

MC8.) *Is the code actively maintained?*

Yes.

MC9.) *What are the dependencies of the code? Does it require Java? A specialised NodeOS?*

Java and VxWorks.

MC10.) *What features does the NodeOS provide?*

N\A.

MC11.) *Which NodeOS API does the platform match, if at all?*

P1520.

APPENDIX C: RCF

C.1 RCF Glossary

C.1.1 Acronym

Acronym		Full Spelling
ANoRMaM	**:**	Active Node Resource Management Module
AOCoM	**:**	Active Objects Control Module
AOR	**:**	Active Objects Repository
DIP	**:**	Destination IP address
EE	**:**	Execution Environment
LR(i)O	**:**	Logical Resource(i)’s Object
LR(i)CC	**:**	Logical Resource(i)’s Controlling Component
LR(i)MC	**:**	Logical Resource(i)’s Monitoring Component
PR(i)CC	**:**	Physical Resource(i)’s Controlling Component
PR(i)MC	**:**	Physical Resource(i)’s Monitoring Component
RCF	**:**	Resource Control Framework
Resource(i) CC	**:**	Resource(i)’s Control Component
Resource(i) CT	**:**	Resource(i)’s Control Table
ROCoM	**:**	Resources Objects Control Module
RR	**:**	Resource Repository
SIP	**:**	Source IP address
VE	**:**	Virtual Environment
VE(i)LR(j)I	**:**	Virtual Environment(i)’s Logical Resource(j)’s Instance
VE(i)M(j)I	**:**	Virtual Environment(i)’s Mechanism(j)’s Instance
VE(i)PR(j)I	**:**	Virtual Environment(i)’s Physical Resource(j)’s Instance
VE(i)R(j)I	**:**	Virtual Environment(i)’s Resource(j)’s Instance

C.1.2 Definitions

Word	Explanation
EE	It is an execution environment for execution of services. The EEs exist in the VEs. Multiple EEs could exist in every VE. EE in a VE depends on type of application that is installed into a VE. In other words, service provider installs EE for execution of applications. (Is it right?)
End-user	It requests service and interacts with the service.
Flow	It is identified by SIP, DIP, Port of SIP, Port of DIP and Protocol Type.
Resource(i)	It means one of physical resources or logical resources.
Security System	It controls security issues.
Service Provider	It provides service to end-users.
VE	It is a virtual environment that is composed of resources. Each service provider can create one VE dynamically and also remove it dynamically.

C.2 Example of Resource Usage in RCF

Figure C-1 shows the example of resources' usage. In this model, the filter module decides to receive the flow or not based on a filter table at first. For example, if there is some information about discarding the data because of security reason, that kind of data will be discarded. The shaper module controls input bandwidth based on the information that is stated by data sender. If the real input bandwidth exceeds the stated input bandwidth, the real flow is shaped into stated bandwidth. Then forwarding module decides to transmit the flow to the CPU or to the routing module based on a forwarding table. At the CPU, the flow is retransmitted to the proper computation module based on a computation table. Besides, the CPU capacity and threads are controlled by scheduler. The computation module uses the memory for data processing and uses the storage to store the data by way of data bus. The packet data is routed based on a routing table at the routing module. At the output bandwidth control module, the packet data is classified and is queued into buffers based on a classifier table. Besides, the output bandwidth is controlled based on information that is stated by data sender.

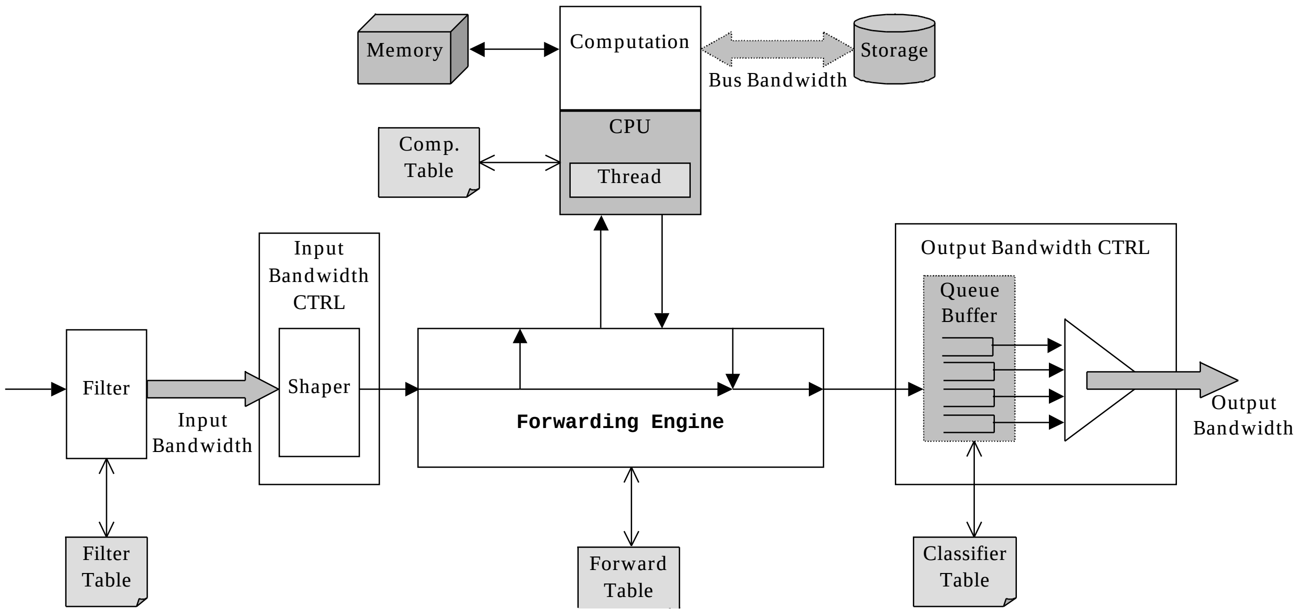


Figure C-1: Flow of data processing

C.3 RCF Scenarios

C.3.1 Creation of Object

C.3.1.1 Initialisation of RCF Block

Direction of Action	**Outline of Action**
/*Initialisation of RCF*/	
	Hardware Resources Checking
	Creating ANoRMaM
/*Initialisation of the ROCoM, the RR, the PRs' control and monitoring components, the LR and the LRs' control and monitoring components*/	
ANoRMaM → ROCoM	Initialise the ROCoM
ROCoM → RR	Creation of the RR
ROCoM → PR(i)CC/PR(i)MC	Initialise the control and monitoring components of the PRs
ROCoM → PR(i)MC	Asking the availability of the physical resources
PR(i)MC → ROCoM	Giving the availability of the physical resources
ROCoM → RR	Storing to the RR the availability of the PRs
ROCoM → LR(i)O	Creating LR(i)O
ROCoM → PR(i)CC	Request for allocation of PRs for the LRs objects
ROCoM → RR	Informing about the allocation of the PRs
ROCoM → LR(i)CC/LR(i)MC	Initialise the control and monitoring components of the LRs
/*Initialisation of the AOCoM and the AOR*/	
ANoRMaM → AOCoM	Initialise the AOCoM
AOCoM → AOR	Initialise the AOR

C.3.1.2 Creation of VE

Direction of Action	**Outline of Action**
/*Creation of a new VE(i)*/	
VE(i) creator → ANoRMaM	Request the creation of a new VE(i) with information of necessary resources
ANoRMaM → Security System	Check the justification of the VE(i) creator
Security System → ANoRMaM	Reply the justification of the VE(i) creator
ANoRMaM → ROCoM	Request to check if the VE(i) can be created.
ROCoM → RR	Check the resource availability from the Resource Repository (RR) and decide to accept the request from VE(i) creator or not.
RR → ROCoM	Return the resources availability
ROCoM → ANoRMaM	Return the result of the resource availability check.
if enough resources	
ANoRMaM → AOCoM	Integrated Request of creation the VE(i)
AOCoM → AOR	Create a unique ID for a new VE(i) and store it to the AOR
AOCoM → AOR	Initialise the proper Resource and Mechanisms Instances

AOCoM → ROCoM	Request for allocation a set of resources per VE Objects Instances
ROCoM → ANoRMaM	Ask if the VE has authorisation to allocate the amount of the resources
ANoRMaM → ROCoM	Gives the authorisation to the VE
ROCoM → LR(i)CC/PR(i)CC	Request to allocate the resources per consumer (VE(i)R(j)I/VE(i)M(j)I)
LR(i)CC/PR(i)CC → ROCoM	Confirmation of Allocation
ROCoM → AOCoM	Confirmation of Allocation
AOCoM→ VE(i)R(j)I/VE(i)M(j)I	Configuration of the Instances in order to be able to use the Resources
AOCoM → ANoRMaM	Confirmation of creation the new VE
ANoRMaM → VE(i) creator	Return the SUCCESS signal of a new VE(i) creation to the VE(i) creator
if not enough resource	
ANoRMaM → VE(i) creator	Return the FAILURE signal of a new VE(i) creation to the VE(i) creator
Endif	

C.3.1.2.1 Creation of Physical Resource Instance in a VE

Direction of Action	**Outline of Action**
Create(VE(i)PR(j)I)	/*Creation of VE(i)PR(j)I*/
ROCoM → PR(j)CC	Request to allocate the physical resources per consumer
PR(j)CC → Resource	Allocate the capacity of requested physical resource(j)
if SUCCESS	
Resource → PR(j)CC	Accept the allocation of the capacity of requested physical resource(j)
PR(j)CC → ROCoM	Inform about the succeed allocation
if FAILURE	(ex. Physical Problem)
Resource → PR(j)CC	Deny the allocation of the capacity of requested physical resource(j)
PR(j)CC → ROCoM	Inform about the failure of the allocation of physical resource(j)

C.3.1.2.2 Creation of Logical Resource Instance in a VE

Direction of Action	**Outline of Action**
Create(VE(i)LR(j)I)	/*Creation of VE(i)LR(j)I*/
ROCoM → LR(j)CC	Request to allocate the logical resources per consumer
LR(j)CC → LR(j)O	Allocate the capacity of requested logical resource(j)
if SUCCESS	
LR(j)O → LR(j)CC	Accept the allocation of the capacity of requested logical resource(j)
LR(j)CC → ROCoM	Inform about the succeed allocation
if FAILURE	(ex. Physical Problem)
LR(j)O → LR(j)CC	Deny the allocation of the capacity of requested logical resource(j)

LR(j)CC → ROCoM	Inform about the failure of the allocation of logical resource(j)

C.3.2 Removing of Object

C.3.2.1 Removing of VE

Direction of Action	**Outline of Action**
/*Remove VE(id)*/	
VE(id) remover → ANoRMaM	Request the removing of the VE(id)
ANoRMaM → Security System	Check the justification of the VE(id) remover
Security System → ANoRMaM	Reply the justification of the VE(id) remover
/*Get the Resource(j)I List*/	
ANoRMaM → AOCoM	Request the removing of the VE(id)
AOCoM → ROCoM	Request the removing of the VE(id)
ROCoM → RR	Get the lists of all the VE(id)R(j)I and VE(id)M(j)I that belong to the VE(id) from the RR
RR → ROCoM	Return the lists of all the VE(id)R(j)I and VE(id)M(j)I that belong to the VE(id)
/*Remove all the Instance*/	
ROCoM → LR(j)CC/PR(j)CC	Request to remove all the resources per consumer (VE(id)R(j)I/ VE(id)M(j)I)
LR(j)CC/PR(j)CC → ROCoM	Confirmation of Removing the resources
ROCoM → AOCoM	Confirmation of Removing the resources
AOCoM → AOR	Remove the ID number of VE(id)
AOR → AOCoM	Confirmation of Removing the ID
AOCoM → AnoRMaM	Confirmation of Removing the VE(id)
ANoRMaM → VE(id) remover	Confirmation of Removing the VE(id)

C.3.2.1.1 Removing of Physical Resource Instance in a VE

Direction of Action	**Outline of Action**
Remove(VE(id)PR(j)I)	/*Removing of VE(i)PR(j)I*/
AOCoM → VE(i)R(j)I	Remove the Instance
AOCoM → AOR	Inform the repository
AOCoM → ROCoM	Request for releasing the allocated resources to the VE(i)M(j)I
ROCoM → PR(j)CC	Request to remove all the resources per consumer (VE(id)R(j)I/ VE(id)M(j)I)
PR(j)CC → Resource	Release the allocated resources
Resource → PR(j)CC	Inform about releasing of allocated resources
PR(j)CC → ROCoM	Inform about releasing of allocated resources
ROCoM → RR	Inform the resource repository about the releasing

C.3.2.1.2 Removing of Logical Resource Instance in a VE

Direction of Action	Outline of Action
Remove(VE(id)LR(j)I)	/*Removing of VE(i)LR(j)I*/
AOCoM → VE(i)R(j)I	Remove the Instance
AOCoM → AOR	Inform the repository
AOCoM → ROCoM	Request for releasing the allocated resources to the VE(i)M(j)I
ROCoM → LR(j)CC	Request to remove all the logical resources per consumer (VE(id)R(j)I/VE(id)M(j)I)
LR(j)CC → LR(j)O	Release the allocated resources
LR(j)O → LR(j)CC	Inform about releasing of allocated logical resources
LR(j)CC → ROCoM	Inform about releasing of allocated logical resources
ROCoM → RR	Inform the resource repository about the releasing

C.3.2.1.3 Removing of VE RCF mechanism Instance

Direction of Action	Outline of Action
Remove(VE(id)LR(j)I)	/*Removing of VE(i)LR(j)I*/
AOCoM → VE(i)M(j)I	Remove the Instance
AOCoM → AOR	Inform the repository
AOCoM → ROCoM	Request for releasing the allocated resources to the VE(i)M(j)I
ROCoM → LR(j)CC	Request to remove all the logical resources per consumer (VE(id)R(j)I/VE(id)M(j)I)
LR(j)CC → LR(j)O	Release the allocated Logical resources
LR(j)O → LR(j)CC	Inform about releasing of the allocated logical resources
LR(j)CC → ROCoM	Inform about releasing of the allocated logical resources
PR(j)CC → PR(j)O	Release the allocated physical resources
PR(j)O → PR(j)CC	Inform about releasing of the allocated physical resources
PR(j)CC → ROCoM	Inform about releasing of the allocated physical resources
ROCoM → RR	Inform about releasing of the allocated logical and physical resources

C.3.3 Controlling of Resource

C.3.3.1 Resource Allocation

Direction of Action	Outline of Action
/* Resource Allocation */	
User → Resource(i) CC	Request the allocation of resource(i)
Resource(i) CC → Security Sys	Check the user
Security Sys → Resource(i) CC	Reply the result of checking the user
Resource(i) CC → Resource(i) CT	Check the resource(i) availability
Resource(i) CT → Resource(i) CC	Reply the result of resource availability
Resource(i) CC → Resource(i) CT	Allocate the resources(i)
Resource(i) CT → Resource(i) CC	Confirm the allocation of resource(i)
Resource(i) CC → User	Reply the result of the resource(i)’s allocation

C.3.3.2 Resource Access(Read/Write/Execute)

Direction of Action	Outline of Action
/* Resource Access */	
User → Resource(i) CC	Request to read/write/execute the data of resource(i)
Resource(i) CC → Resource(i) CT	Check the access right(read/write/execute)
Resource(i) CT → Resource(i) CC	Reply the result of access right
Resource(i) CC → Resource(i)	Access the resource(i) for reading/writing/executing the data
Resource(i) → Resource(i) CC	Reply the result of access from the resource(i) CC
Resource(i) CC → User	Reply the result of request from the user

C.3.3.3 Resource Release

C.3.3.3.1 Manual Resource Release

Direction of Action	Outline of Action
/* Manual Release */	
User → Resource(i) CC	Request the releasing of resource(i)
Resource(i) CC → Security Sys	Check the user
Security Sys → Resource(i) CC	Reply the result of checking the user
Resource(i) CC → Resource(i) CT	Release the resource(i)
Resource(i) CT → Resource(i) CC	Reply the result of resource(i)'s releasing
Resource(i) CC → User	Reply the result of resource(i)'s releasing

C.3.3.3.2 Automatic Resource Release

Direction of Action	Outline of Action
/* Automatic Release */	
Resource(i) MC → Resource(i) CC	Request the releasing of resource(i) when it is over-used.
Resource(i) CC → Resource(i) CT	Release the resource(i)
Resource(i) CT→ Resource(i) CC	Reply the result of resource(i)'s releasing
Resource(i) CC → Resource(i) MC	Reply the result of resource(i)'s releasing

C.3.4 Monitoring of Resource

Direction of Action	Outline of Action
VE(i)R/M(j)I → L/PR(j)	Install callback by specifying a high or low watermark
L/PR(j) → VE(i)R/M(j)I	Invoke callback when condition is met
VE(i)R/M(j)I → L/PR(j)	Get current resource usage

APPENDIX D: IDL MODULE SPECIFICATION

D.1 Resource Control Limits Interface Definition Language Module

```
//File: Limit.idl
//Active network node

#ifndef _LIMIT_IDL_
#define _LIMIT_IDL_

#include <orb.idl>

#include <idlj-extra.idl>

#include <TimeBase.idl>
#include <CosCollection.idl>
#include <RAD.idl>

#pragma prefix "eu.ist"

module Limit {

  // Interfaces to specify accepted languages.

  interface LimitLanguage {};
  interface BSD : LimitLanguage {};
  interface Posix : LimitLanguage {};

  typedef DfResourceAccessDecision::ResourceName
      ResourceName;

  // BSD sysctl

  // Resource names

  // Entities upon which resource limits may be set
  const string SELF = "proc/limit/self";
  const string CHILDREN = "proc/limit/children";
  const string BOTH = "proc/limit/both";

  // Entities upon which priorites may be set
  const string PROCESS = "proc/prio/process";
  const string PRGP = "proc/prio/pgrp";
  const string USER = "proc/prio/user";

  // Attributes for limits
  const long RLIM_INFINITY    = -1;
  const long RLIM_NOFILE = RLIM_INFINITY;

  // Attributes for priorities
  const long PRIO_MIN = -20;
  const long PRIO_MAX = 20;

  // Setting

  // Resource names that may be set

  const string CPU = "proc/cpu"; /* CPU time in seconds */
  const string FSIZE = "proc/fsize"; /* Maximum filesize */
  const string DATA = "proc/data"; /* max data size */
  const string STACK = "proc/stack"; /* max stack size */
  const string CORE = "proc/core"; /* max core file size */
  const string RSS = "proc/rss"; /* max resident set size */
  const string NPROC = "proc/nproc"; /* max number of processes */
  const string NOFILE = "proc/nofile"; /* max number of open files */
  const string MEMLOCK = "proc/memlock";
            /* max locked-in-memory address space*/
  const string AS = "proc/as";
            /* address space (virtual memory) limit */

  // The limit structure:
  // soft limit is current, hard limit is max

  struct Limits {
      long    cur;
      long    max;
  };

  struct LimitResource {
      ResourceName resource_name;
      Limits limits;
  };

  // This is a container for a set of LimitResource entities
  // and provides the activation method.

  exception InvalidLimit { LimitResource which; };
  exception InvalidEntity { ResourceName who; };

  interface LimitResources : CosCollection::Collection {
      attribute ResourceName entity_to_limit;
    // one of process, user, process group

      boolean isActive();
      // whether the collection of LimitSources is active.

      // Toggle switch
      // switches a set of Limits on or off
      boolean makeActive(in boolean active)
        raises (InvalidEntity, InvalidLimit);
  };

  // Getting

  // Resource usages that may retrieved

  const string UTIME = "proc/utime";
  const string STIME = "proc/stime";
  const string MAXRSS = "proc/maxrss";
  const string IXRSS = "proc/ixrss";
  const string IDRSS = "proc/idrss";
  const string ISRSS = "proc/isrss";
  const string MINFLT = "proc/minflt";
  const string MAJFLT = "proc/majflt";
  const string NSWAP = "proc/nswap";
  const string INBLOCK = "proc/inblock";
  const string OUBLOCK = "proc/oublock";
  const string MSGSND = "proc/msgsnd";
  const string MSGRCV = "proc/msgrcv";
```

```
  const string NSIGNALS = "proc/nsignals";
  const string NVCSW = "proc/nvcsw";
  const string NIVCSW = "proc/nivcsw";

  // Composite structure these should map back to the resource names.

  struct LimitTimeT {
      TimeBase::TimeT seconds;
      TimeBase::TimeT useconds;
  };

  struct Rusage
  {
      LimitTimeT ru_utime; /* user time used */
      LimitTimeT ru_stime; /* system time used */
      long ru_maxrss;          /* maximum resident set size */
      long ru_ixrss;      /* integral shared memory size */
      long ru_idrss;      /* integral unshared data size */
      long ru_isrss;      /* integral unshared stack size */
      long ru_minflt;          /* page reclaims */
      long ru_majflt;          /* page faults */
      long ru_nswap;      /* swaps */
      long ru_inblock;         /* block input operations */
      long ru_oublock;         /* block output operations */
      long ru_msgsnd;          /* messages sent */
      long ru_msgrcv;          /* messages received */
      long ru_nsignals;        /* signals received */
      long ru_nvcsw;      /* voluntary context switches */
      long ru_nivcsw;          /* involuntary context switches */
  };

};

#endif /* ifndef _LIMIT_IDL_ */
```

D.2 Demultiplexing Interface Definition Language Module

```
//File: Dmux.idl
//Packet Handling System

#ifndef _DMUX_IDL_
#define _DMUX_IDL_

#include <orb.idl>

#include <idlj-extra.idl>

#include <CosCollection.idl>
#include <RAD.idl>

#pragma prefix "eu.ist"

module Dmux {

  // Interfaces to specify accepted languages.

  interface PacketLanguage {};
  interface IPv4 : PacketLanguage {};
  interface IPv6 : PacketLanguage {};
  interface ANEP : PacketLanguage {};

  typedef DfResourceAccessDecision::ResourceName ResourceName;

  // Packet Demultiplexing

  // From ip_fw.h

  // If a packet meeting IPv4 packet specification appears in a
  // buffer, it can be acted upon and sent to a target.

  // The execution environment will appear as either the redirpt or as
  // the vianame.

  struct IPv4PacketSpecification {
      unsigned long src, dst; // Source and destination IP addr
      unsigned long smsk, dmsk; // Mask for src and dest IP addr
      unsigned long mark; // ID to stamp on packet
      unsigned short proto; // Protocol, 0 = ANY
      unsigned short flg; // Flags word
      unsigned short invflg; // Inverse flags
      unsigned short spts[2]; // Source port range.
      unsigned short dpts[2]; // Destination port range.
      unsigned short redirpt; // Port to redirect to.
      unsigned short outputsize; // Max amount to output to NETLINK
      string vianame;
    // name of interface "via", used by tunneling
      // and other EEs that operate in the kernel as pseudo-devices
      octet tosand, tosxor; // Revised packet priority
  };

  // Standard targets
  enum Target { Masquerade,   // Forward as if from this host
                        Redirect,   // Send to redirpt
                        Accept,           // Let packet through to buffer
                        Deny,       // Ignore the packet
                        Reject,           //  Ignore  the  packet,  but  log
failure
                                          // to sender
                        Return };   // Try the next chain

  // Packet de-multiplexing source. Packets meeting this criteria are
  // sent to an EE: either on an interface name or to a port.

  // Packet sources form a tree structure using a target

  interface PacketSource;

  union TargetSpecification switch(boolean) {
      case TRUE: Target default_target;
      case FALSE: PacketSource custom_target;
  };

  // This should really be a struct and it should be forward

  interface PacketSource {
      attribute TargetSpecification target;
      attribute IPv4PacketSpecification packet_spec;
  };

  // InvalidTarget: type is enumerated, but it may be inconsistent
  // (e.g. Masquerade in forward) or locked by another resource.

  exception InvalidTarget {
      TargetSpecification target;
  };

  // InvalidPacketSpecification: structure may be inconsistent

  typedef sequence<string> fieldNames;

  exception InvalidPacketSpecification {
      fieldNames incorrect_fields;
  };

  // This is a container for a set of PacketSource entities.

  interface PacketSources : CosCollection::Collection {
      attribute ResourceName packet_buffer;
    // one of input, forward, output

      boolean isActive();
      // whether the collection of PacketSources is active.

      boolean makeActive(in boolean active);
      // Toggle switch
      // switches a set of PacketSources on or off.
  };

  exception InvalidPacketSources { };

  // PacketSourceFactory: this creates a PacketSource. It can then be
  // added to a PacketSources entity.

  interface PacketSourceFactory {
      readonly attribute PacketLanguage packet_language;
      // Match this to that given in resource naming

      PacketSource createIPv4(in TargetSpecification target,
                                          in        IPv4PacketSpecification
packet_spec)
        raises (InvalidTarget, InvalidPacketSpecification);
  };

};

#endif /* ifndef _DMUX_IDL_ */
```

D.3 Resource Control Interface Definition Language Module

```
//File: Rcf.idl
//Active network node

#ifndef _RCF_IDL_
#define _RCF_IDL_
```

```
#include <orb.idl>

#include <idlj-extra.idl>

#include <CosCollection.idl>
#include <RAD.idl>
#include <dmux.idl>
#include <limit.idl>

#pragma prefix "eu.ist"

  /*

      SMI Network Management Private Enterprise Codes:

    Prefix: iso.org.dod.internet.private.enterprise (1.3.6.1.4.1)

      The FreeBSD Project - 1.3.6.1.4.1.2238

      const ResourceNamingAuthority =
            "iso.org.dod.internet.private.enterprise.TheFreeBSDProject";

      const ResourceNameComponent IPv4_input = {
            "net/ip_fwchains/input", MANAGEABLE
         };

      const ResourceNameComponent IPv4_forward = {
            "net/ip_fwchains/forward", MANAGEABLE
         };

      const ResourceNameComponent IPv4_output = {
            "net/ip_fwchains/output", MANAGEABLE
         };

  */

module Rcf {

  typedef DfResourceAccessDecision::ResourceName
      ResourceName;

  typedef DfResourceAccessDecision::ResourceNamingAuthority
      ResourceNamingAuthority;

  interface ResourceNamer {
      ResourceNamingAuthority getNamingAuthority();
  };

  // Dmux resource control

  // For IPv4 the names of the three packet buffers are all that is
  // needed.

  interface IPv4ResourceNamer : ResourceNamer {
      readonly attribute Dmux::PacketLanguage packet_language;

      readonly attribute ResourceName IPv4_input;
      readonly attribute ResourceName IPv4_forward;
      readonly attribute ResourceName IPv4_output;
  };
```

```
// Linux and FreeBSD systems can use sysctl(8)
// Make the name of the naming authority explicit.

interface BSDIPv4ResourceNamer : IPv4ResourceNamer {
    readonly attribute ResourceNamingAuthority sysctl;
};

// Limit resource control

interface LimitResourceNamer : ResourceNamer {
    readonly attribute Limit::LimitLanguage limit_language;
};

//

interface BSDLimitResourceNamer : ResourceNamer {
    readonly attribute ResourceName self;
    readonly attribute ResourceName children;
    readonly attribute ResourceName both;

    readonly attribute ResourceName process;
    readonly attribute ResourceName process_group;
    readonly attribute ResourceName user;
};


// Resource Access Control

// This is the base type. It is a sequence of <resource names,
// operation> and a set of attributes, which may be signed and/or
// encrypted.

typedef DfResourceAccessDecision::AccessDefinitionList
    AccessDefinitionList;
typedef DfResourceAccessDecision::AttributeList
    AttributeList;

struct ResourceProfile {
    AccessDefinitionList access_requests;
    AttributeList attribute_list;
};

// All resources that couldn't be released. Attributes will be null.

// All resources that couldn't be changed. Attributes may return how
// much can be used.

enum ResourceUseBreach { Unreleasable, Immutable, Other };

struct OtherResourceBreach {
    string why;
    ResourceProfile what;
};

union BadResourceProfile switch(ResourceUseBreach) {
case Unreleasable: ResourceProfile unreleased;
case Immutable: ResourceProfile immutable;
case Other: OtherResourceBreach other;
};
```

```
  typedef sequence<BadResourceProfile> BadResourceProfiles;

  exception InvalidResourceProfile {
      BadResourceProfiles bad_resource_profiles;
  };

  // Resource managers are given to EEs and VEs. Resources claims can
  // only be released within the original allocation profile - the
  // budget.

  // All resource control is advisory. Resource access control is
  // implemented elsewhere on use. If a resource is capable of being
  // controlled, then it would call:

  // DfResourceAccessDecision::AccessDecison.access_allowed()

  // when it is called.

  interface ResourceManager {
      readonly attribute ResourceProfile budget;

      ResourceProfile list();       // currently used

      // Give up a claim to some part of the budgetted resources.
      // budget is revised to reflect this.

      void release(in ResourceProfile to_be_released)
        raises (InvalidResourceProfile);
  };

  // This only creates resource profiles. It doesn't change the
  // allocation of resources, that is the job of ResourceManager.

  // This just checks you have the names of the resources correct.

  interface ResourceProfileFactory {
      ResourceProfile create(in ResourceProfile request)
        raises (DfResourceAccessDecision::ResourceNameNotFound,
                    DfResourceAccessDecision::InvalidResourceName);
  };

};

#endif /* ifndef _RCF_IDL_ */
```

D.4 Security Interface Definition Language Module

```
//File: Safe.idl
//Active network node

#ifndef _SAFE_IDL_
#define _SAFE_IDL_

#include <orb.idl>

#include <idlj-extra.idl>

#include <CosLicensingManager.idl>
```

```
#pragma prefix "eu.ist"

module Safe {

  // Licensing
  // This interface provides enough for checking an implementation
  // load and who the Principal is, who presumably takes
  // responsibility for it.

  typedef CosLicensingManager::ProducerSpecificLicenseService
      Licenser;
};

#endif /* ifndef _SAFE_IDL_ */
```

D.5 Active Network Node Interface Definition Language Module

```
//File: Fain.idl
//Active network node

#ifndef _FAIN_AN_IDL_
#define _FAIN_AN_IDL_

#include <orb.idl>

#include <idlj-extra.idl>

#include <CosCollection.idl>
#include <CosLicensingManager.idl>
#include <RAD.idl>
#include <safe.idl>
#include <dmux.idl>
#include <rcf.idl>

#pragma prefix "eu.ist"

module ActiveNode {

  // Introduce one interface for controlling a VE/EE.

  interface SuperStructure {};      // Abstract Base

  interface ExtendedSuperStructure : SuperStructure {
      readonly attribute Safe::Licenser licenser;
      readonly attribute Rcf::ResourceManager resource_manager;
  };

  union SuperStructureComponent switch(boolean) {
  case TRUE: Safe::Licenser licenser;
  case FALSE: Rcf::ResourceManager resource_manager;
  };

  exception InvalidSuperStructureComponent {
      string why;
      SuperStructureComponent what;
  };

  interface SuperStructureFactory {
      SuperStructure make(in Safe::Licenser licenser,
                                    in                  Rcf::ResourceManager
resource_manager
        ) raises (InvalidSuperStructureComponent);
  };

  // Abstract base class for EEs.

  interface EE {};

  // Unmanageable EE, so-called because it has been implemented
  // without a separate identity or isolated components. Typically a
  // kernel module.  Its VE loads it and unloads it. VE is unable to
  // manage it resources.

  /* pseudo */ interface UnmanageableEE : EE {};

  // Manageable EE. A user process. It will run under a user
  // identity. It will be subject to OS limits (Posix 1003.13). It may
  // be a Java Virtual Machine and also be under a SecurityManager.

  interface ManageableEE {
      attribute Dmux::PacketSources packet_sources;
      readonly attribute SuperStructure manager;
  };

  // Virtual Environment. A user space manager for EEs.
  // To the Supervisor, which created it, it is also an EE.

  exception InvalidUnmanageableEE { };
  exception InvalidCodeBase { };

  union CodeBase switch(boolean) {
  case TRUE: CORBA::InterfaceDef manageable;
  case FALSE: CORBA::NativeDef unmanageable;
  };

  // VE and Supervisor

  interface VE : ManageableEE {
      // If this is set, then it is a supervisor
      readonly attribute boolean privileged;

      // List of all the EEs and VEs, Manageable and Unmanageable
      // The VE is expected to maintain this collection for the
      // Supervisor. The Supervisor will maintain a list of VEs.

      readonly attribute CosCollection::Collection products;

      // One make operation
      EE make(in Dmux::PacketSources packet_sources,
                  in CodeBase code_base,
                  in SuperStructure manager)
        raises (Dmux::InvalidPacketSources,
                    InvalidCodeBase,
                    InvalidSuperStructureComponent);

      // This cannot destroy a ManageableEE - type trees diverge
      void destroy(inout UnmanageableEE ee)
        raises (InvalidUnmanageableEE);
  };
```

```
};

#endif /* ifndef _FAIN_AN_IDL_ */
```