Working Paper

Virtualizing active networks for Telecom environments

Author(s):
Brunner, Marcus; Stadler, Rolf

Publication Date:
1999

Permanent Link:

https://doi.org/10.3929/ethz-a-004287839 →

Virtualizing Active Networks for Telecom Environments

Marcus Brunner[1] and Rolf Stadler[2]

[1]Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology Zurich (ETH)
Gloriastr. 35, CH-8092 Zurich, Switzerland
Email: brunner@tik.ee.ethz.ch

[2]Center for Telecommunications Research (CTR)
and Department of Electrical Engineering
Columbia University, New York, NY 10027-6699
Email: stadler@ctr.columbia.edu

Abstract

The paper addresses the question on how the benefits of active networking, such as customized packet processing inside the network and flexible service management, can be realized in a telecom environment. More precisely, we ask: How can a network provider, whose infrastructure is based on active networking technology, support a large number of customers, all of which independently install and run their own customized active services in the provider's domain? Our approach is based on network virtualization, and the goal of this paper is to demonstrate that virtualization of active networks can be achieved with considerable benefits for customers and providers and with limited costs. Our work uses the concept of the Virtual Active Network (VAN), a generic service that is offered by an active network provider to customers. We have realized the VAN concept and implemented a VAN provisioning and management architecture on ANET, an active networking platform we have developed. ANET is an all-software, functional prototype of an active network, which allows for experimentation with great flexibility. Further, we have worked out a design for a VAN-enabled node operating system for a high-performance active network node which is currently being built by our laboratory at ETH Zurich in collaboration with a group at Washington University in St. Louis.

Keywords

Active Networks, Service Provisioning, Network Architectures, Virtual Networks

1 Introduction

Recent research in the area of active networking has demonstrated the potential of this new technology. From the service point of view, active networking allows customized packet processing inside the network, on a per-packet, per-flow, or per-service basis. Customized packet processing can be applied, e.g, to application-aware routing, information caching, multi-party communications, and packet filtering [1]. From the man-

agement perspective, active networking technology enables rapid service deployment and flexible service management [2].

While suggesting attractive benefits, active networking also poses serious challenges that must be overcome for this technology to gain wide-spread acceptance. This paper focuses on one of these challenges, namely, on the problem of engineering a multi-user multi-services active network environment. We formulate the problem in the following way: *How can a network provider, whose infrastructure is based on active networking technology, support a large number of customers, all of which independently install and run their own customized active services in the provider's domain?* (Note that the term customer has different meanings here. It can refer to a business unit representing a multitude of employees, or it can refer to a user community, or even a value-added service provider.) To solve this problem, we need the capabilities to (1) enable each customer to create, run and manage his/her own customized active service, and (2) isolate the customers to avoid interference among each other.

Current approaches to active networking address this problem by introducing a *trust relationship* between the party which installs the software in the network and the owner of the network ([3], [4], and [5]). The software is trusted by the network owner in the sense that its execution is assumed to consume no more than a certain amount of resources, and that its functions inside the network do not interfere with other network services. The difficulties with this approach are that (1) only few parties implementing services might be able to build up such a trust relationship with network owners (which might lead to monopolies and thus prevent a wider market for active network services from being created) and (2) even trusted parties will (unintentionally) produce faulty service code with potentially serious consequences for network operation, ranging from overconsumption of network resources to breaking down the whole network.

We take a different approach. We propose to isolate customers from one another through *network virtualization*. The concept we have developed to provide this capability is the *Virtual Active Network (VAN)*. In the same way as an active network can be understood as a generalization of a traditional network, a VAN can be seen as a generalization of a traditional Virtual Private Network (VPN). Similar to a traditional VPN, a VAN can be used by a customer to run network services, using a provider's physical infrastructure. In contrast to a traditional VPN, however, a VAN gives a customer a much higher degree of flexibility and controllability.

Network virtualization does have its costs. Realizing the VAN concept includes building a system for the network provider to create, operate and maintain such VANs. Further, mechanisms must be introduced on the provider's active network nodes to partition resources and police their consumption. *The goal of this paper is to demonstrate that virtualization of active networks can be achieved with considerable benefits for customers and provider and that its costs are limited.*

For this reason, we have realized the VAN concept and implemented a VAN provisioning and management architecture on ANET, an active networking platform developed in our laboratory at the TIK, ETH Zurich. The ANET platform is an all-software, functional prototype of an active network, which allows us to evaluate and demon-

strate active networking concepts. Further, we have worked out a design for a VAN-enabled node operating system for ANN, a high-performance active network node, which is currently being built in our laboratory, in collaboration with Washington University in St. Louis [5]. This design will be further refined, and the VAN concept will be realized on ANN, as part of the FAIN project in the Fifth (EC) Framework Program [6]. (Progress in this work will be reported in the final version of the paper.)

The paper is organized as follows. Section 2 introduces a framework for active networks in a telecom environment. The VAN concept is discussed in Section 3. Section 4 describes the realization of the VAN on the ANET prototype and describes scenarios of VAN provisioning and service management on this platform. Section 5 outlines the design space and our design decisions for realizing VANs on ANN. Section 6 surveys related work. Section 7 summarizes the contributions of this paper and gives an outlook on further work.

2 A Framework for Active Networks in Telecom Environments

2.1 The Interaction

Figure 1 shows the interaction taking place between a customer domain and a provider domain for the purpose of service provisioning, service delivery, and service management. Depending on the type of service, customers and providers interact in two fundamentally different ways. The first way is characterized by a provider offering functionality in its domain through a service interface. The second way of interaction shown in Figure 1 relates to the case where a customer out-sources control and management of a specific service to a provider, which installs and runs the service in the customer domain. The customer is not involved in service installation, upgrade, and management, but can concentrate on its core business instead. The provider customizes the service according to the customer's requirements.

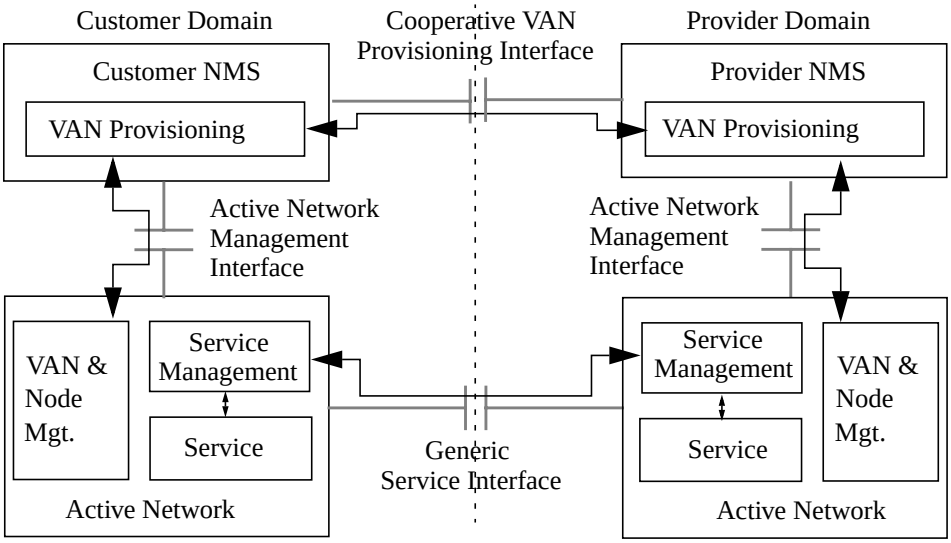


Figure 1: Customer-Provider Interaction in an Active Telecom Environment

In an active networking environment, the above described two ways of interactions between a customer and a provider can be realized in a flexible way with respect to service abstractions and control capabilities for the customer in the provider's domain and vice versa. In the following, we outline our framework for interaction in an active networking environment, which we have first proposed in [2].

Figure 1 shows the interaction between a customer domain and a provider domain for service provisioning, service delivery and service management in our framework. We propose that the provisioning of a specific (active) network service X is split into two different operations:

- the provisioning of a generic service, which we call the VAN service, is performed via the cooperative VAN provisioning interface, in cooperation with the provider; and
- the installation of service X is performed via the generic service interface, without further interaction with the provider. The same interface is used managing service X during its lifetime.

2.2 The Interfaces

The introduction of active networking technology in telecom environments, characterized by the use of active networking nodes as network elements, will change the role of the interfaces the following ways.

First, using active networking nodes enables the definition of a *generic service interface* for network services, based on the concept of active packets. In addition, this service interface can be used by the customer for service management interactions, i.e., for operations related to the installation, supervision, upgrading and removal of a specific service. Therefore, service management operations can be performed by a customer in a flexible way without interaction with the provider's management system.

Second, the management interface, i.e., *the cooperative VAN provisioning interface*, can be restricted to the task of service provisioning. Similar to the service interface, the management interface can be kept generic; it relates to a generic service abstraction that allows for installing and running a large class of network services.

Third, the *active network management interface* relates to the tasks of single-domain service provisioning and network element management. Further, the active network management interface becomes generic through the introduction of active packets replacing the standardized management protocols implementing the interaction between the network management system and the network elements to be managed. This allows each domain to implement its own management system. Additionally, the process of configuring the network element on behalf of service provisioning and the process of monitoring of the network elements can profit from the active network technology in terms of information aggregation, customized filtering, and the delegation of network control.

2.3 Relation to Traditional Telecom Environments

In a traditional telecom environment, the process of service provisioning and the resulting service abstractions are service-specific, whereas in an active networking environment, both the provisioning process and the service abstraction can be realized to be truly generic. Provisioning, say, a virtual network service in a traditional telecom environment, includes setting up Virtual Links between customer premises networks and allocating resources to these Virtual Links. For the customer, the service abstraction consists of a set of links, associated with bandwidth and QoS guarantees.

In contrast, the provisioning of a (generic) service in our framework for an active telecom environment gives the customer a more complex, but also more powerful service abstraction. We call this service abstraction a *Virtual Active Network (VAN)*. The VAN concept and its benefits are the subject of Section 3.

3 The Virtual Active Network (VAN)

A *Virtual Active Network (VAN)* can be described as a graph of virtual active nodes interconnected by Virtual Links. Virtual active nodes are in this paper called *Execution Environments (EEs),* following the terminology of the AN working group [7]. An Execution Environment has resources attached to it in form of processing and memory resources, provided by the underlying active networking platform. Similarly, a Virtual Link has bandwidth allocated to it. We envision that a single (physical) active node can run several virtual active nodes belonging to different VANs, and a single (physical) network link can support several Virtual Links for different VANs. (The term Virtual Active Network, as defined in this paper, is also used by other authors in a different way [8].)

A VAN can further be seen as a *service* offered by a provider to a customer. Via the cooperative VAN provisioning interface shown in Figure 1, a customer and a provider negotiate the initial configuration of this service according to the customer's requirements, and they can re-negotiate the VAN resources during the life time of the service.

The abstraction that a VAN provides is that of an active network. This means that the user of a VAN (or: the customer in our terminology) has the same capabilities as the user of a "real" active network. For instance, users of VANs can install and run active network services on VANs and thus can take full advantage of the features of active networking technology.

Figure 2 shows an active network with five nodes in a provider domain. On this network, two VANs have been installed, one for customer 1 and one for customer 2. The figure also shows the *management VAN*, which interconnects the management EEs. It is used by the provider for VAN provisioning and supervision (see Section 4).

What are the key implications of the VAN concept for customer and provider? First, the VAN provides a *generic service abstraction* for an active telecom environment. From a provider's point of view, the VAN is the entity according to which active network resources are partitioned and according to which the customers, using the provider's infrastructure, are isolated from one another. The VAN is further the (only)

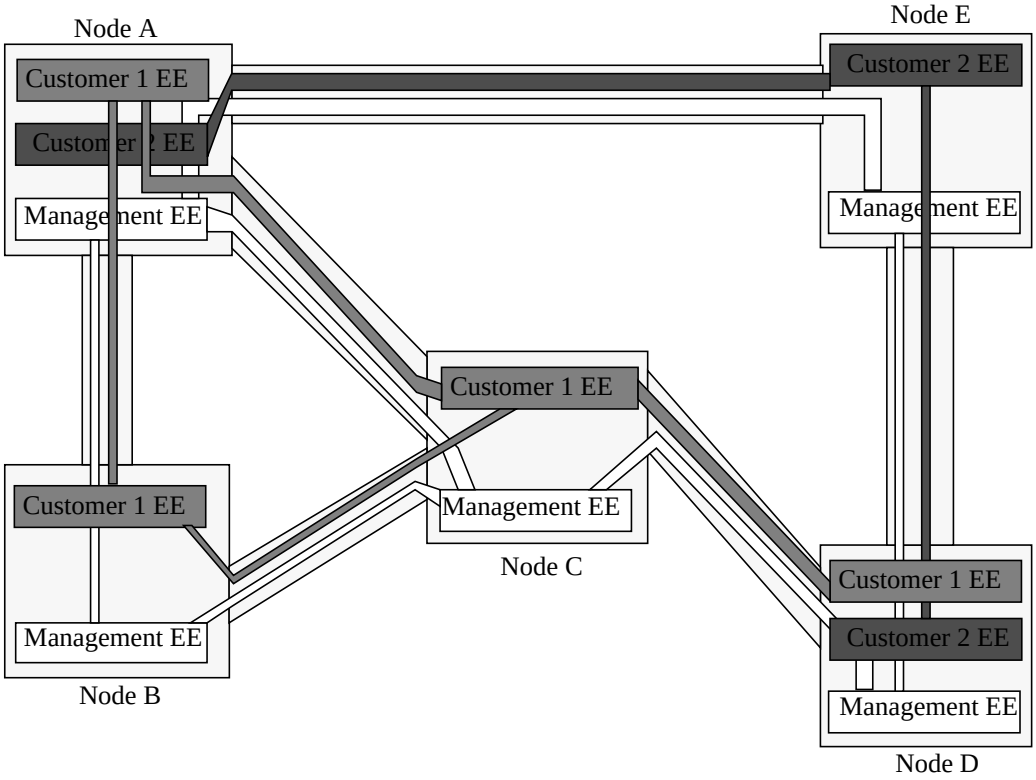


Figure 2: An active network with nodes A, ..., E, on which VANs for Customers 1 and 2 have been installed.

object that is shared between provider and customer, and it is the object of negotiation between the two parties. Specifically, the provider is not concerned about which specific service(s) a customer is running on its VAN. The task of the provider is solely to monitor and police the use of resources on the VAN level and to ensure that the QoS, as agreed upon between customer and provider, can be guaranteed.

Second, from a customer's perspective, the VAN concept allows for installation and *management of active network services, without interaction with the provider.* (As mentioned above, all interactions between customer and provider relate strictly to the VAN.) The customer can run a large variety of active network services on the VAN. These services are only restricted by the specific Execution Environment(s) the VAN supports. (Developing and defining Execution Environments for active networks is currently subject of intensive research. In our work, we base on the current state of the AN working group [7]).

Third, --as a general benefit of active networking-- the VAN concept enables *rapid deployment of new network services*. Deploying and upgrading network services is difficult and time consuming in today's networks, due to the closed, integrated architecture of network nodes. With the concept of a VAN, which divides the active network resources into partitions for different customers, the installation of any customer-specific service becomes feasible, and, as explained before, it can be accomplished by the customer alone, without interaction with the VAN provider.

Lastly, customers can run a mix of different network services on a single VAN. This allows customers to perform *dynamic re-allocation of VAN resources* to the various services, according to their own control objectives and traffic characteristics--again, without interaction with the VAN provider.

As mentioned before, the VAN concept can be compared to that of a Virtual Path (VP)-based Virtual Private Network (VPN). Similar to a VAN, a VP-based VPN provides customers with a service abstraction, on which they can run their own services, such an IP-based data service or a real-time service. Since a VP is a simple abstraction, a customer's ability to control traffic inside the provider's domain is very limited. (See [2] for a discussion of this point.) A VAN, on the other hand, is a much more complex abstraction than a VP, and, consequently, gives customers extensive control capabilities inside the provider's domain. In a similar way as dynamic bandwidth provisioning can be performed in a VP-based VPN [2], we envision that VAN resources can be renegotiated during the life-time of a particular VAN via the VAN management interface shown in Figure 1.

Figure 3 gives an operating system point of view of an active network node in a telecom environment. A node operating system layer configures and provides access to the node's resources, such as links, processing and memory resources. This layer runs the Execution Environments, separates them from each other, and polices the use of the resources consumed by each Execution Environment.

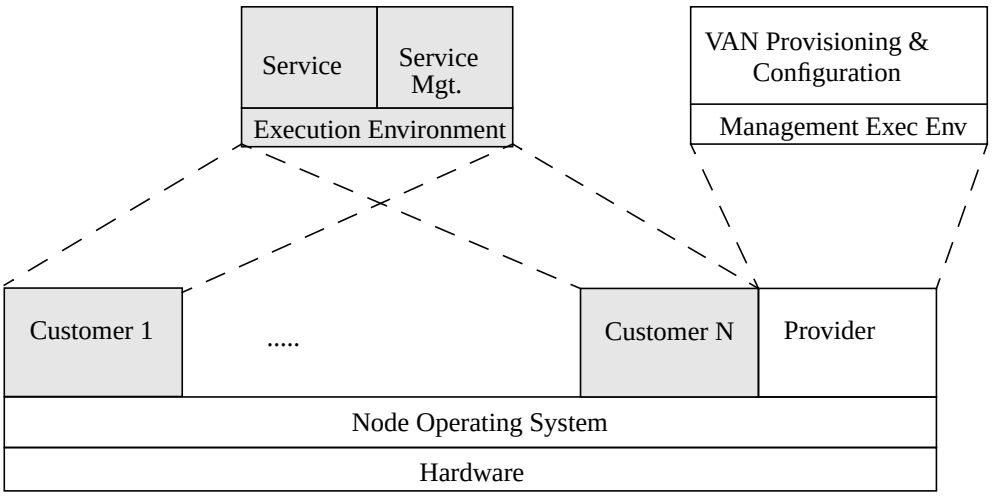


Figure 3: Architecture of an Active Network Node in a Telecom Environment

Figure 3 specifically shows the case where a provider offers Virtual Active Networks to several customers (Customer 1,..., Customer N). The figure shows one node of such a VAN. Each customer runs its service in a separate Execution Environment. A special Execution Environment called the Management Execution Environment runs the provider's VAN provisioning and configuration system, which creates Execution Environments for customers and is able to modify and terminate them. The interface the node operating system has to provide is discussed in Section 5. The VAN provisioning and configuration system is controlled by the provider's management system via the

exchange of active packets. The provider's management system, as shown in Figure 1, maintains a global view of the provider's domain for the purpose of VAN provisioning.

Our approach, as illustrated in Figure 3, is compliant with the architecture of an active network node developed by the AN Working Group [7]. The difference between Figure 3 and [7] is that we explicitly assign each Execution Environment to a particular VAN, i.e., to a particular customer, whereas AN Working Group argues only for different types of Execution Environments, which is also supported in our design.

4 Realizing VAN Provisioning and Management on the ANET Platform

We have built an active networking platform, called ANET, in order to test, evaluate and demonstrate active networking services and management concepts. The core of this platform consists of a cluster of Ultra-SPARCs, interconnected via an Ethernet LAN. Each active network node runs on a separate workstation. On top of this infrastructure, we have implemented traffic generators, traffic monitors, a VAN management system, and a service management system.

All software components of the ANET platform are written in Java. We chose Java because of its strengths as a prototyping language for networking environments, and because Java directly supports the realization of active packets through the concept of mobile code. Additional Java features which we take advantage of include object serialization, thread support, and safe memory access achieved by the type-safety of the language. In our ANET implementation, an active network node is implemented entirely in software, which gives us the flexibility of experimenting with different designs. Performance issues, such as realizing a high throughput of packets on a network node, are beyond the scope of the ANET work. However, we are currently realizing, as described in Section 5, the key capabilities of our active network management framework on ANN [5], a high-performance active networking platforms.

The active network node architecture given in Figure 3 is realized on ANET as follows. The in-bound links deliver the incoming packets to the appropriate Execution Environments according to a multiplexing identifier in the packet or in underlying/link layer protocol headers. The node operating system schedules the Execution Environments, taking into account the processing resources allocated to each of the Execution Environments. The out-bound links run the packet schedulers on the outgoing multiplexers and transmit the packets produced by the Execution Environments to neighboring nodes.

The VAN management system, which allows for VAN provisioning and configuration of the node operating systems of a single domain, has a centralized structure in our ANET implementation. One part of the system is the VAN management station, which uses active packets, sent to the Management Execution Environments, to create Execution Environments, to install Virtual Ports, and to configure cut-through links, in order to set up a new VAN. Management operations are executed via the configuration and management interface of the node operating system, described in Section 5. The configuration and management interface further gives access to statistical information of

the node operation system, the installed Execution Environments and Virtual Links, which is used for VAN monitoring.

Also the (customer-operated) service management system is implemented in a centralized way, and its main component is the service management station. The service management system uses active installation packets for the creation of a service and active monitoring packets to enable supervision of the service.

The complexity of the software we have built to date (active network node, Execution Environments, VAN management system, and service management system) is in the order of 400 Java classes with 30'000 lines of code.

In the following, we illustrate some of the design principles and capabilities of our ANET platform by describing a series of demonstrations we can perform.

4.1 Demonstrating VAN Provisioning and Supervision

Figure 4 shows the situation at the start of the demonstration. The provider network consists of three active network nodes. The provider management system is connected to one of these nodes. Three customers are involved in this demonstration. Customer A and B have two (active) customer premises networks each, and customer C has three. The Virtual Active Networks for customer A and customer B have been setup by the VAN management system. The view of the VAN management station at this point in the demonstration is displayed in Figure 5a. It shows the VANs for customer A and B and the management VAN. Further, customer premises networks are represented by a star labeled customer A, B, and C.

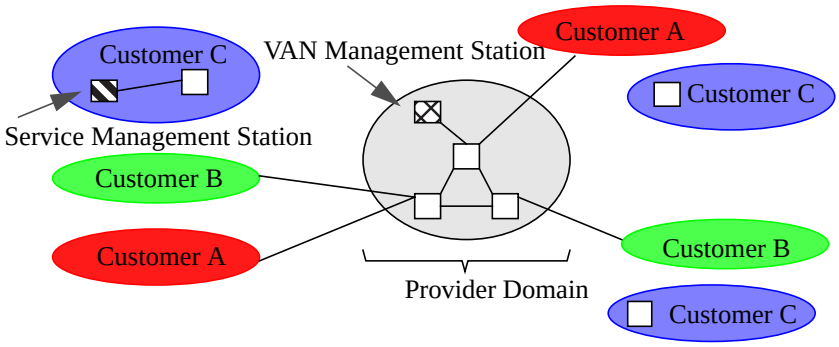
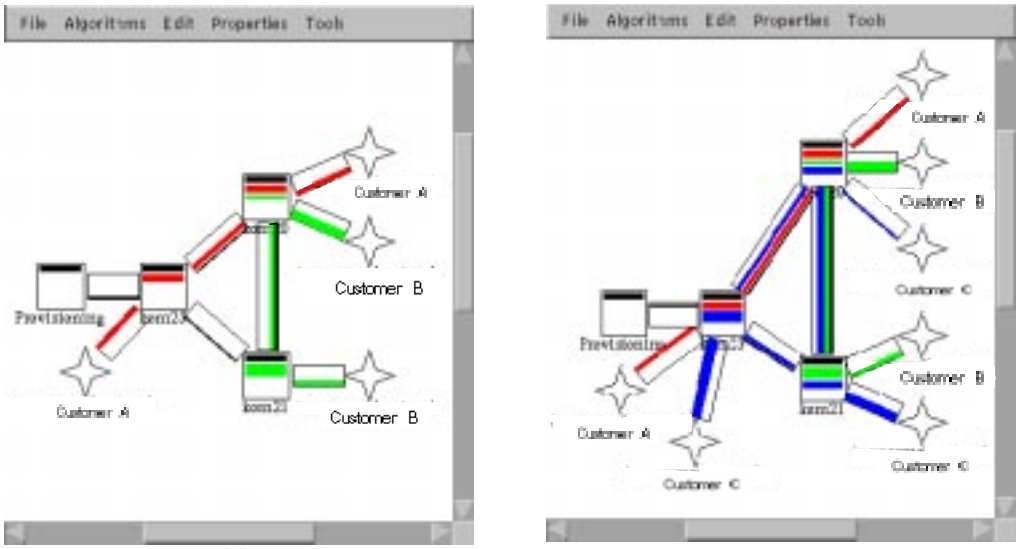


Figure 4: Situation at the Start of the Demonstration

The VAN provisioning capability is demonstrated by setting up a new VAN for customer C. The process of provisioning a VAN includes the installation of Execution Environments on all three nodes, connecting the Execution Environments by setting up Virtual Links between them and configure a Virtual Port in the direction of the active customer networks. The view of the VAN management station after the VAN for customer C has been provisioned is shown in Figure 5b.

The service management system in the domain of customer C has now the view of a VAN spanning over the provider domain and all domains of customer C. Figure 6 shows the view of seven nodes connected over links. Three of them are in the provider



(a) (b)

Figure 5: VAN provisioning as displayed on the provider's VAN management station: **before (a)** and **after (b)** provisioning a new VAN.

domain and four in the domain of customer C. Note that, on this level of abstraction, nodes in the customer domain do not differ from nodes in the provider domain. The service management system has the view of one active network, on which services can now be installed and supervised.

Figure 6 shows a window from the service management station of a customer. It includes a snapshot of a Execution Environment of a VAN. In our current ANET implementation, the service management system displays the configuration and the state of Execution Environments in VAN nodes. The figure shows the buffers of the CPU scheduler, the memory, the in-bound, and the out-bound links. Three buffers are associated with the CPU scheduler: the default buffer for (active) packets, a second buffer for packets that belong to service management functions (e.g., filters for detecting specific events), and a third buffer for packets of a mechanism that routes the packets of the service management system. This is the basic configuration of an active network node, after the VAN has been set up by the provider and the service management system has been initialized by the customer. At this point, the service management system is ready to install specific network services and service management functions on the VAN.

4.2 Customer-controlled Service Installation, Upgrade, and Supervision

On the ANET platform, we can demonstrate the installation, upgrade, and supervision of an IP-service. (We have chosen the IP-service because a well-known service is better understandable, than an sophisticated active service.) Installing an IP service on a active network node is achieved by configuring a virtual router inside the node's Execution Environment. The service management system sends a sequence of active packets to the Execution Environment. Processing these packets results in installing an IP routing table, creating output buffers for the virtual out-bound links, setting up packet

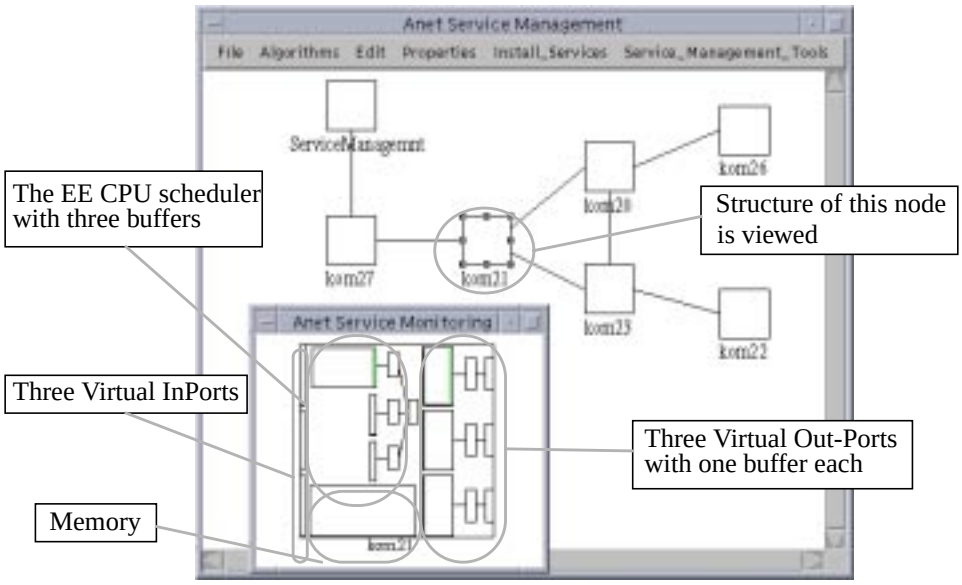


Figure 6: View of the Service Management Station after Provisioning a VAN for a Particular Customer

schedulers that operate on these buffers, installing function code for routing and management operations, configuring service-specific control parameters and management parameters, etc. After that, the IP service is initialized, which includes starting the routing protocol.

Upgrading the IP-service to an IP-service supporting several traffic classes with different QoS requirements is accomplished in our ANET system by reconfiguring the virtual routers inside the Execution Environments. The service management system sends an active packet to each Execution Environment of the VAN. The processing of this packet results in upgrading the packet classifier (to detect the class of a packet), setting up buffers for each traffic class, and substituting the packet scheduler with a scheduler for multi-class traffic.

Figure 7 shows the structure of the Execution Environment after installing an IP-service and upgrading it to a multi-class IP-service. Compared to Figure 6, the structure of the output-buffers has changed to contain two buffer partitions, and the CPU scheduler part has grown by two additional components, one for the IP routing protocol and one for the management of the multi-class IP-service.

In our ANET implementation, the service management system can change the partitioning of the output buffers, by sending active packets to the virtual routers installed on the ANET platform. Further, it can monitor the buffer usage, by configuring the active management component to send packets back to the service management station in regular time intervals. This way, we can perform service management operations the same way as a customer would do while managing its IP service on a Customer Premises Network (CPN).

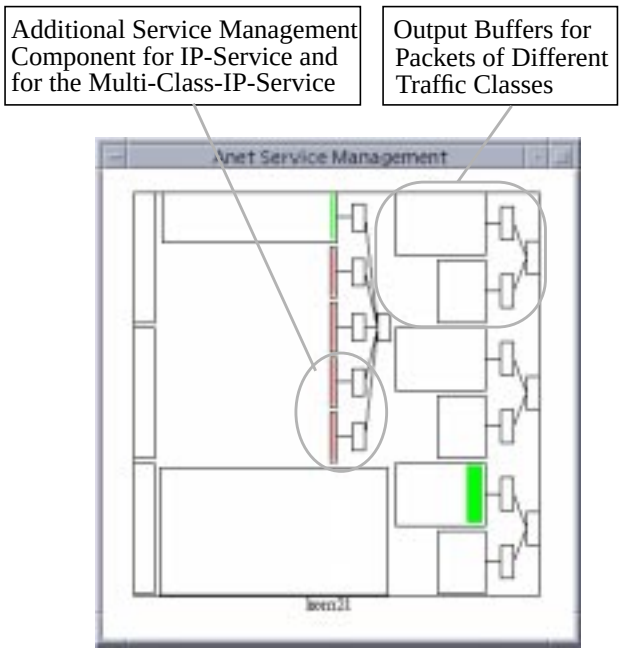


Figure 7: Node Structure after Installing an IP-Service and Upgrading to a Multi-Class-IP-Service

5 Realizing the VAN Concept on a High-Performance Active Networking Platform

In the previous section, we have shown how VAN management is implemented on the ANET platform, a functional active networking testbed. Our current work is focused on the realization of the VAN concept on ANN (Active Network Node), a high-performance active networking platform, which is being built in our laboratory (TIK, ETH Zurich) in collaboration with Washington University in St. Louis [5]. The heart of ANN is one of the fastest active network nodes today. It is based on a Gigabit ATM switching hardware and uses a node operating system that efficiently process active packets in the kernel.

As part of the ANN project, a first version of a node OS has been developed. Its design aims at efficient packet processing and at providing QoS per flow, realized by per-flow scheduling at the output ports. In addition, dynamic code loading from trusted servers is supported.

In order to support VANs on ANN--while maintaining the high throughput of the ANN nodes--the current design of the node OS needs some modifications and enhancements. Instead of resource allocation per flow, resource allocation per EE is required. Second, mechanisms for partitioning and isolating CPU and memory need to be added to the kernel. Lastly, the node OS must support an interface for creating and modifying virtual active nodes and Virtual Links on that particular node. We call the new node OS that supports the above features the *VAN-enabled node OS* for ANN.

In the remainder of this section, we present the current hardware and software architecture of ANN, and we discuss the design space our design decisions for engineering a VAN-enabled node OS.

5.1 The ANN Node Architecture

The hardware of an active network node is shown in Figure 8. The node consists of a set of Active Network Processing Elements (ANPE, four in Figure 8) connected to the ATM switching fabric. The switch supports eight ports with rates up to 2.4 Gb/s on each port. The ANPE comprises a general-purpose processor (Intel Pentium™), a large FPGA (100,000 gates), and memory (64 MB). The ANEPs are connected to the switch backplane via the ATM Port Interconnect Controller (APIC) chip [5]. Each ANEP independently runs the ANN node OS.

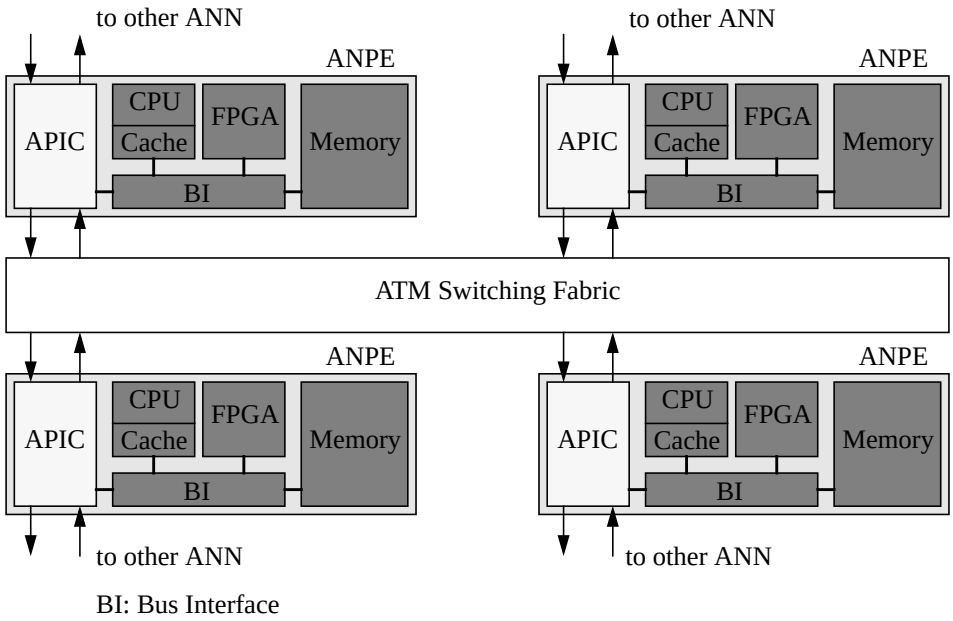


Figure 8: Hardware Architecture of an Active Network Node (ANN)

The ANN node OS is based on the NetBSD kernel. Modules implementing service-specific packet processing functions, called *active plug-ins* in the ANN context, are downloaded from code servers and installed on ANN nodes via the NetBSD mechanism for loading kernel modules. A download is triggered by an active packet with a reference to a module not currently resident in the kernel. The active plug-in is then initialized on a per-flow basis. A selector is chosen to label the flow and is propagated to the upstream node. The upstream node puts the selector into every subsequent packet of the flow, which allows the downstream node to efficiently lookup the plug-in instance to process the packet. Since the plug-in is executed in the kernel, it has access to all kernel data.

5.2 The VAN-enabled Node Operating System

This section describes the functionality needed in the node OS in order to realize the VAN concept on a high-performance active networking platform. On a conceptual level, the functional requirements can be summarized in the following four points.

- Resource Partitioning: Resources, such as CPU-cycles, memory, and transmission bandwidth of the outgoing physical links have to be partitioned among different EEs, i.e., customers.
- Resource Policing and Isolation: An EE must be prevented from consuming more resources than granted by the VAN provider. Further, its allocated memory has to be protected from unauthorized read and write.
- Demultiplexing and Multiplexing: Active packets on an incoming physical link must be demultiplexed, identified and forwarded to their corresponding EEs. Furthermore, active packets leaving the EEs need to be multiplexed onto the outgoing physical links.
- Cut-through Links: The node operating system needs to support cut-through links, allowing packets to pass through the node on a fast path, without being processed.

What are the resource control and multiplexing mechanisms offered by the current ANN OS, and what are the design choices for the new mechanisms necessary to realize a VAN-enabled node OS?

Bandwidth Partitioning

In the current ANN OS, partitioning and isolation of transmission bandwidth is supported by two packet schedulers on the physical output ports--a modified Deficit Round Robin and a Hierarchical Fair Service Curve scheduler [9][10]. In order to support VANs, we will use this functionality, but the bandwidth on the output port will be allocated per EE, not per flow as in the current version of the ANN OS.

Partitioning CPU Resources

Efficient partitioning of the CPU resources is hard. In the current ANN node OS, CPU resources are not explicitly allocated to flows, and active packets are processed on a first-come-first-serve basis. On the other hand, the ANN node OS loads the code for packet processing from a trusted server whose code has known resource consumption. In the VAN context, the code that is executed in an EE is provided by the customer and, therefore, may not be trusted by the provider. Therefore, a VAN-enabled node OS must support hard guarantees to EEs as far as CPU allocation is concerned.

In order to design an efficient CPU allocation mechanism for a node OS, we must consider the following. First, for a typical active service, most active packets are expected to need only a small number of CPU-cycles to be processed. The rest of the packets, which mostly relate to service control and management tasks, are expected to need much more processing time per packet. Second, an active packet is processed in the context of an EE. Third, the CPU resource is allocated per EE, i.e., the CPU is partitioned among the EEs.

To achieve high CPU utilization, the time for context switches between EEs as well as the number of context switches (per time interval) must be minimized. A way to keep

the time for a context switch short is to switch between EEs only after an active packet has been completely processed and, therefore, the execution context for this packet must not be stored. For this reason, the VAN-enabled node OS must be non-preemptive for “most” packets. To keep the number of context switches small, as many packets as possible from the same Execution Environment should be processed without interruption. However, there must be a time limit after which a context switch must occur, in order to guarantee the maximum waiting time for an EE to get access to the CPU. We introduce the *Maximum Processing Unit (MPU)* as the maximum time interval the node OS gives to an EE without interrupting its execution. (The MPU concept is similar to that of the Maximum Transmission Unit (MTU) as the maximum length of an IP packet on a path without fragmentation [11]).

Note that the MPU, together with the CPU scheduling policy, influence the delay for active packets to traverse an active node. This is important with respect to end-to-end QoS requirements for an active service on a VAN. For this reason, we believe that the VAN-enabled node OS must support the MPU concept. Practical values for MPUs will have be determined by experiments.

Partitioning Memory

Memory partitioning is not supported in the current ANN OS. It is needed in the VAN-enabled OS and can be realized in a straightforward way by assigning a block of memory to each EE.

Memory isolation is not supported in the current ANN OS. Since the ANN OS is supposed to execute only trusted code, this is not a big problem. However, memory isolation is required for the VAN-enabled node OS, since the service code is not assumed to be trusted.

The following memory isolation mechanisms have been developed in the operating system research.

- The type-safe programming languages in combination with static typing ensures the integrity of the running system. It has a low run-time penalty, because only few checks have to be executed at run time. On the other hand, the mechanism needs more compile time.
- The hardware address space protection, which is typically implemented as memory page protection in virtual memory systems or with memory segment registers, relies on hardware support.
- Sandboxing introduces for many operations, such as jump and store, additional code for address checking, which leads to a decrease in run-time performance [12].

For the VAN-enabled OS, we will use the hardware address space protection mechanism, which is supported by the Pentium processor on the ANEP card.

Multiplexing/Demultiplexing

Since the multiplexing id has only local, per-link significance, the id and the table to lookup the associated EE can be quite small--equal to the number of EEs supported on an active node. In the ANN OS, multiplexing is performed by setting up ATM Virtual Channels (VC) between Execution Environments and by dispatching an incoming

active packet to a specific EE according to its Virtual Channel Identifier (VCI). The VAN-enabled OS will make use of the same mechanisms.

Cut-through Links

Cut-through links are needed, since a VAN generally does not map one-to-one onto the underlying hardware topology. Using cut-through links, a Virtual Link of a VAN can run through one or more physical active nodes, without active packets being executed on those nodes. In the VAN-enabled OS, cut-through links will be realized by setting up VCs between EEs.

5.3 Techniques for VAN Management

VAN management includes the creation, modification, monitoring, and deletion of VANs on an active network infrastructure. The network management system (NMS) shown in Figure 1 maps the (global) VAN management operations onto (local) operations to be executed on the active network nodes. A VAN-enable node OS needs to support these operations on the local level.

The active network management interface--the interface between the NMS and the active network nodes (Figure 1)--can be realized in two ways. The traditional way is based on the MIB concept. In this case, a manager process in the NMS interacts with agents on the active network nodes via SNMP or CMIP. These agents accesses the node OS via the local VAN management interface. The second way makes use of active networking technology. Here, the VAN management operations are mapped onto active packets that operate on the management VAN (Figure 2). These active packets execute commands in the Management EEs, which, as above, provide access to the node OS via the local VAN management interface.

We favor the active networking based solution. It allows for efficiently monitoring of VANs through information aggregation and customized event filtering [13]. Additionally, the provider can profit from active networking technology in terms of simplified updating of the VAN management functions. Therefore, we have applied the active networking approach for realizing VANs on the ANET platform and will do so on ANN.

5.4 The Local VAN Management Interface

The local VAN management interface provides methods for setting up Execution Environments, configuring Virtual Links, creating cut-through links, and monitoring the resource consumption of the Execution Environments and Virtual Links. Table 1 lists functions in the local VAN management interface. The functions have Java-language like notation with input parameters in brackets and return values written in front.

The local VAN management interface contains additional functions, not shown in Table 1, to receive information about the state of the node OS and the current resource consumption of the EEs, Virtual Ports, and cut-through links. All monitoring functions address the managed object via an id, e.g. eeid, in_portid. A detailed description of the syntax and semantics of the local VAN management interface can be found in [14].

`vin_portid install_Virtual_InPort (in_portid, eeid, inmid);`
`vout_portid install_Virtual_OutPort (out_portid, eeid, outmid, bandwidth);`
`void remove_VirtualPort (vportid);`
`eeid install_EE (ee, cpu_resource, memory);`
`void remove_EE (eeid);`
`ctId install_ThroughLink (in_portid,inmid,out_portid,outmid,qlen,bandwidth)`
`void remove_ThroughLink(cut_through_id);`
`...`

Table 1: Local VAN Management Interface

Initializing a new Execution Environment on an active network node can be performed in two different ways. First, the node OS knows a set of predefined and pre-installed types of EEs. In this case, only the requested type has to be specified in the *install_EE* function. The second way, which we have implemented in the ANET prototype and which we will also implement in the VAN-enabled node OS, explicitly loads the EE code. This scheme allows the local VAN management system to initialize new, previously unknown types of EEs on an active node.

5.5 Mapping Execution Environments onto Multi-Processor Nodes

In a multi-processor distributed memory model, such as the ANN hardware architecture (Figure 8), it has to be decided on which processor a EE is installed and running. Strict partitioning of memory and CPU resources allows the ANN node to install a EE to one processor, because no interaction between EEs of different customers is foreseen in our architecture.

A VAN-enabled OS for ANN can be designed in two ways. First, the physical ANN node can be seen as a single active node (Figure 9). Here, we need to introduce a master VAN management module, which communicates with the local VAN management interfaces on each ANPE. The interface to the master VAN management module is the same as described in Section 5.4. The module provides transparent access to the local copies of the VAN-enabled node OS running on the ANPEs.

In the second design, each ANPE appears as one active network node (for the purpose of VAN management), and these nodes communicate with each other via the switch fabric. Each ANPE independently runs a VAN-enabled node OS. With this design, the decision to choose a particular ANPE for VAN creation is delegated to the network management system (Figure 1). We favor this approach for realizing the VAN concept on ANN, because the intermediate level of control, the master VAN manager, is not needed.

6 Related Work

The Switchware project [4] takes a language based approach. The memory access is controlled via a type-safe Programming Language for Active Networks (PLAN). Multiplexing is implicitly build into the active packet carrying the code to evaluate on intermediate active nodes. The executing code calls in a controlled way previously

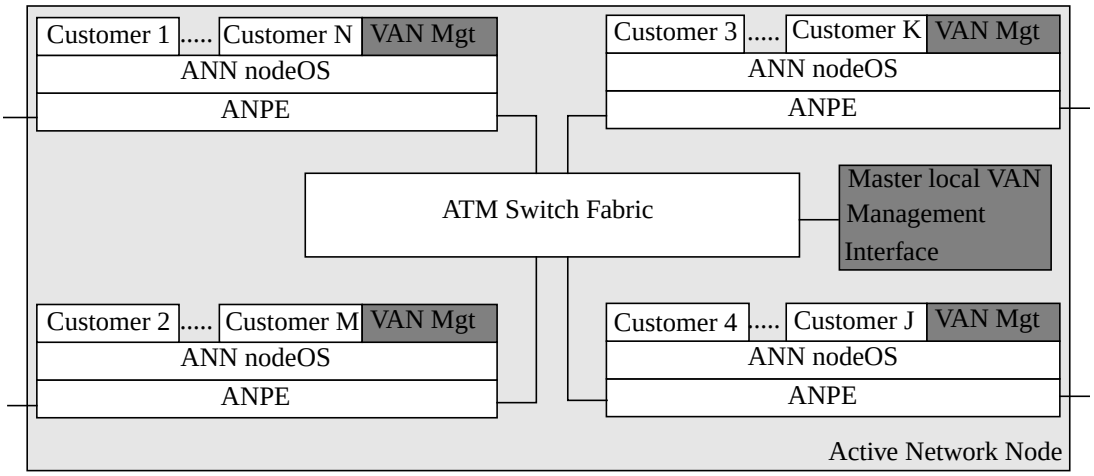


Figure 9: Design where an ANN node is seen as a single Active Network Node

installed routines on the network node. Switchware does not support resource partitioning, nor the notion of a Virtual Active Network for the programming of the network, as our work does.

The Resource Controlled Active Network Environment (RCANE) [15] supports an active network programming model over the Nemesis Operating System [16]. It provides control and accounting of system resources. RCANE uses PLAN as a type-safe and verifiable programming language to protect memory between Execution Environments. CPU scheduling is accomplished using a modified Earliest Deadline First (EDF) algorithm [17] to schedule Virtual Processors. A Virtual Processor may run different threads of an Execution Environment. In contrast, our approach tries to avoid preemption to minimize the overhead of saving thread contexts and we can use the same type of schedulers for CPU and for packet scheduling.

The Lancaster Active Router Architecture (LARA) [18] has a hardware architecture similar to ANN. Network ports have a CPU each and are interconnected by a Bus backplane instead of an ATM backplane. The node operating system, based on a Linux kernel, holds some data structures on behalf of the Execution Environments running on top of them. This node OS may be enhanced towards a VAN-enable node OS with the mechanisms described in the previous section.

The Netscript project [8] has the abstraction of a Virtual Active Network as the object to program. Additionally, they deal with automated generation of MIB to manage active services via standardized management protocols. Contrary to the Netscript project, our work leaves open the question of service instrumentation in an active network environment, but it focuses on a flexible framework for supporting interactions between customers and providers with the VAN abstraction as a key concept.

The adaptable network control and reporting system (ANCORS) [19] is a system to assess, control, and design active networks. It supports the deployment and system management of legacy software and new active network applications in a network. In contrast to our work, code for the services are loaded only from trusted code servers

and deployment and control commands are only accepted if they arrive from a known set of IP addresses. Further, ANCORS uses the UNIX process abstraction running on PCs to isolate customers from each other, where we run a node OS on an active networking hardware platform.

The Genesis project [20] brings up the notion of *virtual programmable networks* as the entity to bind resources to it, and they describe the life-cycle to install such a virtual programmable network. First, the Genesis approach is derived from the programmable networking (build on top of switchlets [21]), which concentrate the virtualization to the control plane. Our paper also includes the data path, which makes the network active on the data, control, and management plane. Second, the described life-cycle virtual networks, does not include any customer-provider interaction, as our paper does.

Virtual Networks for the Internet are proposed in [22]. They use the concept of Virtual Networks to partition the physical network resources, where the resulting partitions may implement their own, independent control and processing mechanisms. End-user traffic is classified and assigned to one of the Virtual IP Networks by edge routers, according to a programmable policy. In contrast, our work does build on active networking technology, where not only the classification policy is programmable, but the service itself is programmable and the customized service can be installed by a customer into the Virtual Active Network.

7 Conclusion and Further Work

One of the key problems in the field of active networking is the prospect of users loading and executing untrusted code in the network. In this work, we took the approach of network virtualization to address this problem. We introduced the Virtual Active Network (VAN) concept and focused on VAN setup and management in an active telecom environment. Further, we showed the potential that active networking opens up if the VAN concept is realized.

We illustrated the properties of a VAN by describing a series of scenarios conducted on the ANET active networking platform. Further, we have given the design for the VAN concept to be realized on ANN, a high-performance active network platform. This design will be further refined will be realized on ANN, as part of the FAIN project in the Fifth (EC) Framework Program. Any progress in this work will be reported in the final version of the paper.

8 References

[1] D. Tennenhouse, J. Smith, W. Sincoskie, D. Weatherall, G. Minden, “A Survey of Active Network Research,” IEEE Communications Magazine, Vol. 35(1), 1997.

[2] M. Brunner, R. Stadler, “The Impact of Active Networking Technology on Service Management in a Telecom Environment,” IFIP/IEEE International Symposium on Integrated Network Management (IM ‘99), Boston, USA, 1999.

[3] D. Weatherall, J. Guttag, D. Tennenhouse, “ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols,” IEEE Conference on Open Architec-

ture and Network Programming (OPENARCH'98), San Francisco, USA, April 1998.
[4] S. Alexander, W. Arbaugh, M. Hicks, P. Kakkar, A. Keromytis, J. Moore, C. Gunter, S. Nettles, J. Smith, "The Switchware Active Network Architecture," IEEE Network, Vol. 12(3), 1998.
[5] D. Decasper, G. Parulkar, S. Choi, J. DeHart, T. Wolf, B. Plattner, "A Scalable, High Performance Active Network Node," IEEE Network, Vol. 13(1), 1999.
[6] The European Commission, "Fifth (EC) Framework Program," http://www.cordis.lu/fp5.
[7] AN Architecture Working Group, "Architectural Framework for Active Networks," K. Calvert (editor), 1998.
[8] Y. Yemini, S. da Silva, "Towards Programmable Networks," IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'96), L'Aquila, Italy, 1996.
[9] M. Shreedar, G. Varghese, "Efficient Fair Queuing using Deficit Round Robin," SIGCOMM'95, 1995.
[10] I. Stoica, H. Zhang, T. Ng, "A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Services," SIGCOMM'97, 1997.
[11] J. Mogul, S. Deering, "Path MTU Discovery," Request for Comments 1191, 1990.
[12] R. Wahbe, S. Lucco, T. Anderson, S. Graham, "Efficient software-based fault isolation," Symposium on Operating System Principles, 1993.
[13] M. Brunner, "A Service Management Toolkit for Active Networks," TIK-Report No. 78, http://www.tik.ee.ethz.ch/tik/research/publications/Publications.html, 1999.
[14] M. Brunner, "Service Management in a Telecom Environment based on Active Network Technology," Ph.D. thesis, Swiss Federal Institute of Technology Zurich, Switzerland, 1999.
[15] P. Menage, "RCANE: A Resource Controlled Framework for Active Network Services," First International Working Conference on Active Networks (IWAN'99), Berlin, 1999.
[16] I. Lesli, "The Design and Implementation of an Operating System to Support Distributed Multimedia Applications," IEEE Journal on Selected Areas in Communications, Vol. 14(7), September 1996.
[17] C. Liu, J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment," Journal of the ACM, Vol 20(1), February 1973.
[18] R. Cardoe, J. Finney, A. Scott, W. Shepherd, "LARA: A Prototype System for Supporting High Performance Active Networking," First International Working Conference on Active Networks (IWAN'99), Berlin, 1999.
[19] L. Ricciulli, P. Porras, "An Adaptable Network COntrol and Reporting System (ANCORS)," IFIP/IEEE International Symposium on Integrated Network Management (IM '99), Boston, USA, 1999.
[20] A. Campbell, M. Kounavis, D. Villela, H. De Meer, K. Miki, J. Vicente, "The Genesis Kernel: A Virtual Network Operating System for Spawning Network Architectures," IEEE Conference on Open Architecture and Network Programming, (OPENARCH'99), 1999.

[21]J. van der Merwe, I. Leslie, “Switchlets a Dynamic Virtual ATM Networks,” Fifth IFIP/IEEE International Symposium on Integrated Network Management (IM’97), San Diego, USA, 1997.
[22]J. Redlich, M. Suzuki, S. Weinstein, “Virtual Networks for Customizable Traffic Treatments,” First International Working Conference on Active Networks (IWAN’99), Berlin, 1999.