A service management toolkit for active networks

A Service Management Toolkit for Active Networks

Marcus Brunner

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

Abstract

Active networking, where network nodes perform customized processing of packets, is a rapidly expanding field of research. This paper investigates the realization of service management on a Virtual Active Network (VAN), the key concept in our active network management framework [11]. A VAN can be seen as a thin (software) layer in an active networking environment, which creates a generic service abstraction, offered by the provider to the customer. The VAN concept transforms a multi-domain situation into a single domain for a customer. The customer can install, run and supervise network services into the VAN, without interaction with the provider. In this paper, we describe a service management toolkit, which facilitates designing and managing services taking advantage of active networking technology for code reuse, code distribution, flexible event filtering, and programmable information aggregation. Further, we show how the service management toolkit is realized on ANET, an active networking testbed that we are in the process of building.

Keywords

Active Network Management, Service Management, Network Monitoring, Event Handling.

1 Introduction

In this paper, we investigate the problem of service management in an active networking environment. Based on the fact that an active network node is able to run code, which is remotely installed or carried along with active packets, and that service functionality and management functionality can be accessed via a generic service interface, we explore the options for realizing the task of service management.

The generic service interface for network services is enabled by the concept of *active packets* and 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. The customer-provider interaction over the management interface, i.e., the interface through which the customer and the provider management systems cooperate, is restricted to the task of service provisioning, which includes setting up connectivity and securing resources. The management interface is kept generic; it relates to a generic service abstraction, which we call Virtual Active Network (VAN) [12] that allows for installing and running a large class of network services.

The VAN service abstraction transforms a physical multi-domain environment into a virtual single domain environment, which allows service management operations to be performed by a customer without interaction with the provider's management system. This opens up new possibilities but also more responsibility and complexity for the customer. A customer not only requests a new service from a service provider as in a traditional telecom environment, but he installs and supervises customized active services into a Virtual Active Net-

work, which probably spans over multiple provider domains. Additionally, service and management system of a service are tidily coupled, because they are introduced into the same VAN over the same interface. This tide coupling enables a fine-grained and customized control and management of services. Note, that the party we denote as a customer, could be a end-user of a service, or a service provider selling a service to end-users.

In order to facilitate the design and implementation of different services and service management systems, a modularization of the service and the service management system is recommended to reuse software and to enhance the flexibility by extending the software towards service-specific and the customer-specific requirements.

On the other hand, active networking technology is very well suited to overcome before mentioned problems. It really facilitates the implementation of service management systems in a extendable, reusable, and scalable way. The problem of performance bottlenecks and scalability in service supervision is attacked with programmable event filters, management by delegation, flexible grouping of nodes, and grouping of events into composite events. The installation, upgrade, removal of services is supported by the active networking technology through the possibility to efficiently implement customized multi-casting.

In this paper, we propose an flexible and extensible service management toolkit, which extracts and implements the service-independent functionality common to most service management system. The challenge of this is that (1) the service management system does not know in advance what services are installed and supervised, but it should be able do adapt to service-specific and customer-specific control and management needs without being totally replaced. Further, (2) a VAN spanning over many provider domains typically yields in a large number of EEs, which poses problems in the scalability in the number of EEs. And (3) the installed services may produce a large amount of management information, because not only transmission is a factor but also the execution of active packets in EEs.

The paper is organized as follows. Section 2 outlines our framework for service management and provisioning in a telecom environment that is based on active networking technologies. Section 3 describes a service management toolkit. The realization of the toolkit on our ANET active networking platform is presented in Section 4. Section 5 shows the power of the service management toolkit by describing the implementation of a active packet tracer. Section 6 surveys current efforts on active technologies for network and service management, and Section 7 summarizes the contributions of this paper and gives an outlook on further work.

2 Active Network Management Framework

In the following, we outline our management framework for interaction in an active networking environment, which we have first proposed in [11].

2.1 Service Provisioning in Active Network Environments

In an active networking environment, 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.

Figure 1 shows the interaction between a customer domain and a provider domain for service provisioning, service delivery and service management in our framework. The key differences between a traditional and an active environment becomes clear. First, the active

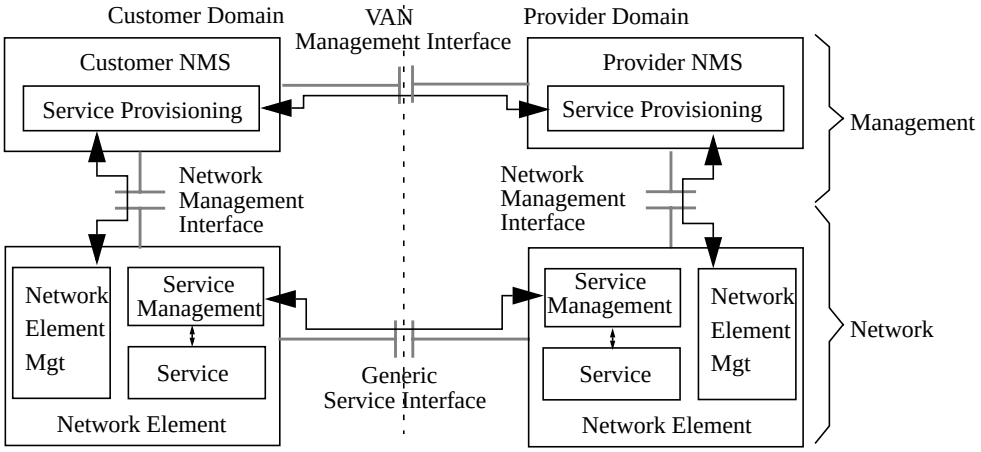

Figure 1: Management Interaction in an Active Telecom Environment

networking approach allows for a clear separation between service provisioning and service management. Second, service management can be realized via the service interface.

Service provisioning includes creating service connectivity and securing resources for the service. It is performed in a cooperative manner between the customer and the provider through a management interface. 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.

Service provisioning in our active network management framework gives the customer a more complex, but also more powerful service abstraction. We call this service abstraction a *Virtual Active Network (VAN)*. A VAN can be described as a graph of virtual active nodes interconnected by virtual links. Virtual active nodes can also be called *Execution Environments (EE)*, following the terminology of the AN working group [7]. An EE has resources attached to it in form of processing and memory resources, provided by the underlying active networking platform. Similarly, a virtual active link has bandwidth allocated to it. During the provisioning phase, the customer and the provider cooperatively set up such a graph and allocate resources to the VAN.

The VAN is a powerful abstraction in the sense that the customer can, within the limitations of the specific VAN topology and the allocated resources, install, configure and run network services without further interaction with the provider. It further divides the active network resources into partitions for different customers. For a deeper discussion of the Virtual Active Network and the setup of a VAN refer to [12].

2.2 Architecture of an Active Network Node

The architecture of an active network node has to provide mechanisms to provision a VAN and to separate VAN's of different customers. For VAN provisioning, it needs an access point for a VAN management system. The VAN separation mainly consists of resource control on a network node.

Figure 2 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 EEs, separates them from each other, and polices the use of the resources consumed by each EE.

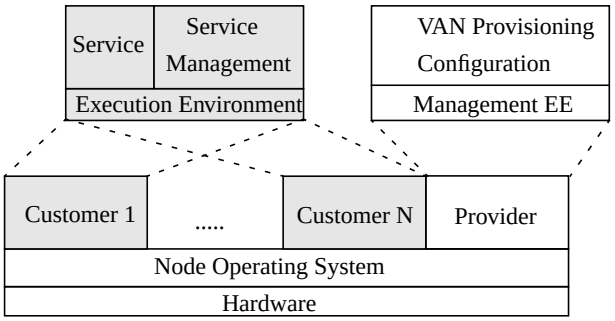

Figure 2: Architecture of an Active Network Node in a Telecom Environment

Figure 2 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 an active network. Each customer runs its services in a separate EE. Service management functionality is installed and runs together with the service in an EE. This allows the tide integration of service, service control and service management functionality. A privileged EE (Management EE) runs the provider's VAN provisioning and configuration system, which creates EEs for customers and is able to modify and terminate them.

2.3 Customer Service Management

Service management include the installation, running, supervision, and removal of a service on an active networking infrastructure. The service management system runs on one VAN, which possibly consists of interconnected VANs of different domains. The VAN concept transforms a multi-domain situation into a single-domain situation through the virtualization of the network.

Figure 3 shows a VAN spanned over two customer premises active networks and two pro-

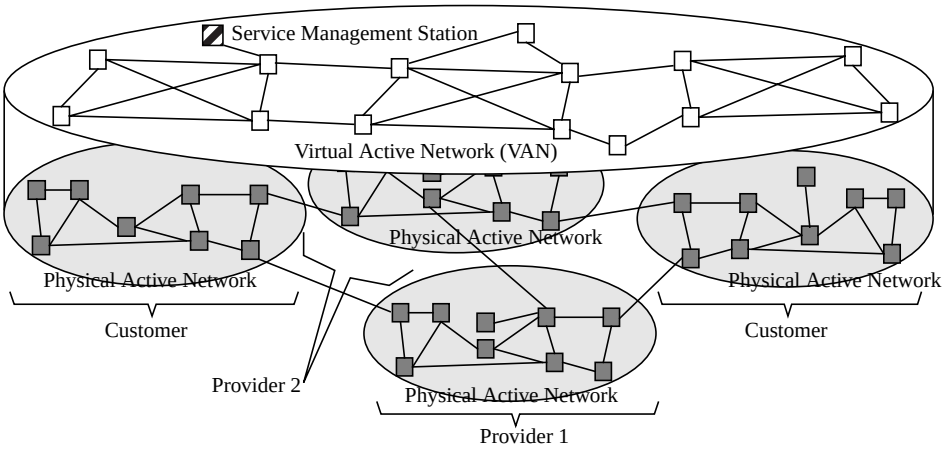

Figure 3: A Virtual Active Network over three Domains

vider networks. The service management system autonomously installs, updates and supervises one or more services on the VAN. Autonomous means, that no interaction with a provider is needed, that the reserved VAN resources are guaranteed, and that the services introduced by the system are not disturbed by malicious services of other customers. The service management system is controlled from a *Service Management Station (*Figure 3), which serves as the human-computer interface for the service management system. The station initiates service management actions and displays monitoring information.

3 A Service Management Toolkit

In such an active networking environment it is very promising to design a service management toolkit, which implements similarities and which is extensible to adapt to customer-specific or service-specific needs. A service designer and a service management system designer can take components out of the toolkit, may customize and use it. This is possible, because many services can be composed of similar components and many service management tasks need similar mechanisms, similar informations about services or similar information access mechanisms.

An *active network service*, as defined in our framework, consist of *stationary service components* installed on the EEs of a VAN, and active packets, which travel through the VAN accessing the stationary components when they execute on an EE. Further, the EEs are configured to meet the run-time requirements of an active service. Figure 4 shows active packets

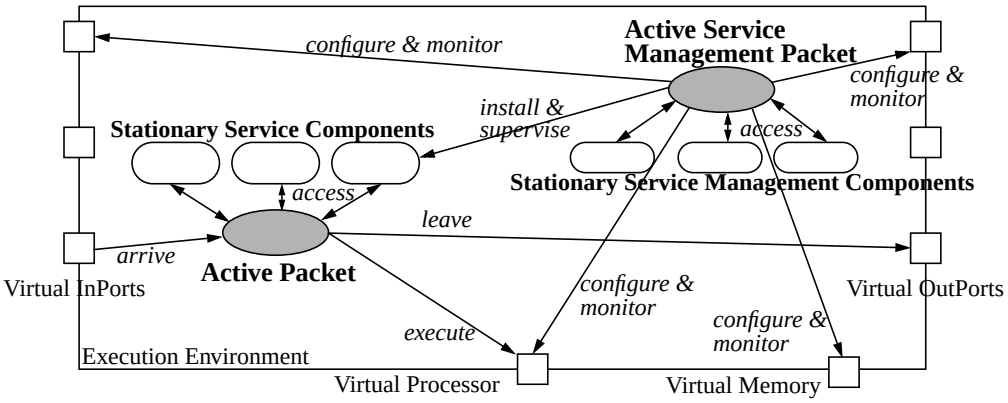

Figure 4: Active Packets and Service Components

arriving at a Virtual InPort of the EE, accessing stationary service components while executing, and leaving the EE on a Virtual OutPort.

Stationary service components include data structures and algorithms accessible over predefined interfaces. The code and initial values of the algorithms and data structures are brought into the EE during the service installation phase, and they are stored in the nodes memory. Many of the service components to be installed in the EE can be reused for many different services. We call this components *basic service components*.

The basic service components implement most commonly used data structures and functions, and are extensible for service-specific needs. Our service management toolkit makes available packet queues to store packets for later execution or transmission. Further, queue composition nodes to build hierarchical queueing systems are available. A queue composition node consists of many queues or of other queue composition nodes and a scheduler implementing a scheduling discipline. Additionally, packet classifiers are available, which decide into which queue a packet arriving on the Virtual InPort is queued. This allows a service to forward particular packets, which do not need any computation, directly to output buffers. Also a set of different schedulers are contained by the toolkit, which are used for packet scheduling or CPU scheduling. Note that a service designer can implement components not based on basic service components.

Active packets are the basic communication entity in terms of what information they transport, and they are the basic computation entity in terms of what functions they execute on the EEs. Active packets are used for two different tasks. First, they are the objects which imple-

ment an end-to-end networking service on behalf of a distributed application. Second, active packets are executing service control and management tasks on behalf of the service management system. In Figure 4, they are labeled *active service management packets*. Active service management packets prepare the EE to be used by an active service. The following sections describe the installation and monitoring of an active service.

3.1 Installing an Active Service

The service installation is initiated by the service management station, where the customer decides what service to install. The steps performed by the installation process of a service are as follows. First, after the decision about what service to install, service-specific configuration parameters need to be set. Second, code and parameters of service components are transported to the EEs of the VAN. Third, the service code and data structures are installed and configured into the EE in order to setup the functional and run-time behavior of the service.

Setting the configuration parameters in the service management station is service-specific, therefore the service package exports a user interface for setting the parameters, which is included into the service management station's user interface and is displayed after the decision about the service to install has been taken.

Distributing service code and parameters to each node of the VAN is for most service types service-independent, because most services need the same code with similar configuration parameters on all EEs. Different distribution algorithms and strategies can be chosen in terms of efficiency, scalability, and capabilities of the EEs. E.g., [13] shows that more efficient implementations of the installing process can be achieved if the EEs are able to clone the active installation packets, which yields to an efficient multi-casting of active packets. Service types using not the same code or the same configuration parameters need a more complex, customized installation algorithm, but can also profit from multi-casting capabilities of the active network. We choose broadcasting of active installation packets to all EEs of a VAN as the basic installation algorithm in our toolkit.

Active installation packets contain, beside the code and installation parameters, service-specific network information, which can be gathered during the installation process. E.g., installing an IP-service the routing algorithm is interested, what neighbors can be reached directly over the virtual links, which is easily gathered. The example shows that the broadcasting is enhanced by distributing service-specific information during the service installation process. Such a flexibility can be achieved by the use of active networking technology.

The execution of the active installation packets installs the service code and service-specific data structure on the EE, initializes the service, and calls the initialization procedure of the code to let the service perform service-specific run-time configuration. Specifically, the service-specific initialization consists of installing and configuring programmable classifiers on the Virtual InPorts, packet schedulers on Virtual OutPorts, memory management on Virtual memory, and CPU schedulers on the Virtual Processor. Additionally, active installation packets are used to install stationary service management components on the EEs, in order to setup service-specific and general service management functionality, which is mainly targeted to control and supervision of the service.

3.2 Support for Service Supervision on a Node

Service supervision can be understood as a normal service in terms of how the supervision components are installed and run on a VAN. Service supervision is composed of *service-dependent* and *service-independent* parts. Service-dependent parts supervise service components installed on a EE in a service-dependent manner, where as monitoring of the EE is service-independent.

Our toolkit supports service-dependent supervision in two different ways. First, the EE makes predefined mechanisms available, which helps designing service-dependent supervision components. One is an event collection point for events in the EE to support asynchronous communication between service components and their service management components. Others are predefined, extensible classes of events representing service actions such as error events, warnings, or debugging information. Second, basic service components have build-in support for service supervision. These basic components have open interfaces for synchronous access by service supervision components, and instrumented data structures and functions, which sent an event to the collection point on defined conditions. E.g., in the case where current number of bytes used in a buffer is monitored, the information is synchronously accessible over a defined interface to the buffer, further the code is instrumented to send an event to the collection point, when the buffer overflows.

Service-independent supervision components monitor the EE's behavior during the run-time of services and the re-configuration of the EE. Supervision support for EE components is implemented by instrumenting the EE at two places. First, the interface between services and the EE is instrumented such that each invocation of that interface generates an event. Second, the interface between the node operating system and the EE is instrumented to generate an event on each invocation. E.g., the EE creates an event on each arrival of a packet on a Virtual InPort.

3.3 Information Collection

Mechanisms known to increase the scalability and efficiency of distributed service supervision are filtering, information aggregation, and management by delegation. All this mechanisms are supported by active networking technology in very flexible and customized way.

In our toolkit filters are programs installed into the event collection point. This means during the service management components installation phase, also filter programs are installed into the event collector, to pass or drop events of a specific type. These allows to install very fine grained filters, which only pass events of a specific type of active packets with specific payload, or it may pass all events triggered by an action of an active packet type, e.g., all packets belonging to the routing algorithm. Predefined events together with pre-implemented, configurable filters are available to use and extended for service-specific monitoring. Filter are not only configured towards specific requirements, but are also replaced by new ones on the fly. Further, the collection point allows to install more than one filter to allow different service management components to get another trace of event.

The events passing the filter are stored and later fetched by a service management component called *Event Dispatcher,* which takes appropriate actions depending on the event's type and content. Typically, the action taken by the Event Dispatcher is sending the event to the service management station. Or in the case of delegated and automated service management, a re-configuration of the service or the EE is executed. As later discussed the events can also be sent to a representative of the particular node's group.

The process of collecting management information from all EEs of a VAN is the next step. Since a VAN may consist of a large number of EEs and Virtual Links, the service management system may not scale well in terms of overwhelming amount of management information to be transmitted and processed in the service management station. The amount of monitoring data is reduced by information aggregation, which is very-well supported by active networking technology.

In our toolkit, information is aggregated by assigning EEs to a group, which is represented by one of these EE (*representative*). The group and representative information is stored on each EE. All information collected in EEs of a group is sent to its representative, where a *concentrator* collects the information and compacts it with a service-specific aggregation function, which is a program installed during the service management system installation, and which can be replace on the fly during monitoring. The aggregated information is generally sent to the next representative in hierarchical grouping or to the service management station in flat grouping. In order to allow the concentrator to take another action depending on the result of the aggregated management information, the forwarding function can be replaced dynamically to execute a delegated management task. Which node is a suitable representative and how EEs are grouped depend on the application of the information aggregation process, and is therefore left open in the toolkit. E.g., for monitoring purposes a grouping of highly interconnected nodes or organizational related nodes may be suitable.

Figure 5 shows a VAN grouped into three groups of EEs with one representative each. On

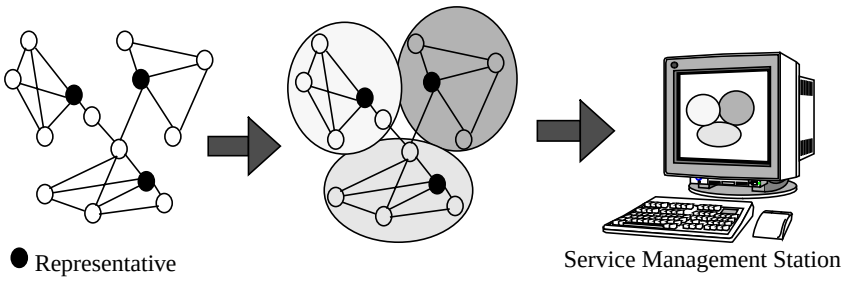

Figure 5: Grouping of Nodes

the service management station only the three groups are visible. The monitoring information of the services running on this VAN is aggregated in the representatives. The service management station only get the calculated result for each group.

The grouping is flexible and need to be configured by the implementor of a supervision system. Note that the mechanism for grouping and aggregation are separated and both are freely programmable, which allows a customer to choose an appropriate supervision scenario. Beyond the traffic reduction achieved by programmable filters and programmable concentrators, their is also the benefit of on demand creation of different views of the services running on the VAN, the view can range from very high-level overviews to very low-level views of one part of the system. This can be achieved in our service management toolkit by dynamically changing filter and concentrator functions.

4 Realizing the Service Management Toolkit on the ANET Platform

We have built an active networking platform, in order to test, evaluate and demonstrate active networking service and management concepts. The core of this platform consists of a cluster of Ultra-SPARCs, interconnected via an Ethernet. Each active network node runs on

a separate workstation. On top of this infrastructure, we have implemented a VAN management system to setup VANs, the service management toolkit, and some simple services and their management components.

All software components of our 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 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 currently beyond the scope of our work. However, we plan, in a future phase of this project, to realize the key capabilities of our system on one of the emerging high-performance active networking platforms, such as [6].

4.1 Realizing an Execution Environment for a Customer Service

When our VAN provisioning system creates an EE in an active network node, a structure shown in Figure 6 is initialized. This structure represents an EE in a VAN. It can be seen as

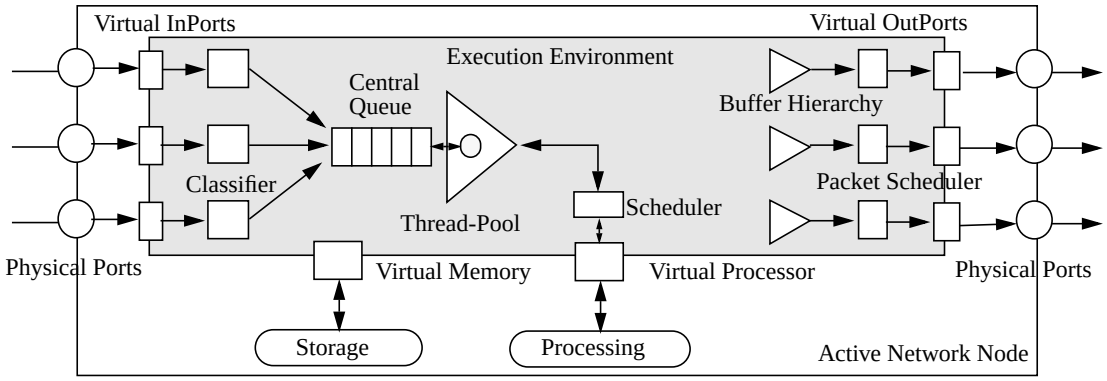

Figure 6: Initial Configuration of an Execution Environment

an initially empty wrap, which is filled with service functionality during the service installation phase. The EE contains Virtual InPorts and Virtual OutPorts (each of which corresponds to a link of the Virtual Active Network), a central queue for all incoming packets, a thread pool containing one thread, and storage for data and functions.

A packet classifier is associated with each virtual in-bound link to dispatch the incoming packets according to its content to different locations. Initially, the classifier puts all incoming packets to the central queue. The thread takes active packets out of the central queue, binds code to the packet, and processes them, using the basic instruction set of the EE. The first few packets configure the Execution Environment for the specific needs of the service. The Virtual OutPort packet scheduler and buffer hierarchy are empty at the start-up of the EE. Actually, the service management system has minimally to install one buffer.

The service management system can replace or re-configure all the components initially available on the Execution Environment. E.g., a classifier can be installed, that puts some packets in the central queue, and other directly to a buffer in the buffer hierarchy. As soon as more than one thread is in the thread pool, the processing scheduler need to be replaced with one implementing a scheduling policy, e.g. a round-robin or priority based policy.

Figure 6 can be interpreted as a virtual machine. It allows us to install and configure a service-specific, more complex virtual machine, by sending active packets that contain the code for the new virtual machine to the EE. This process can be compared to boot-strapping an operating system on a computer.

4.2 Configuration Interface

The configuration interface (Table 1) consists of functions to configure and extend the execution environment for specific needs of an active service, specifically the run-time behavior of an active service. These functions are called by service management active packets during the service installation process. The `installPacketScheduler` function installs a packet scheduler into a Virtual OutPort. The `installPacketSchedulerNode` installs a new scheduler node. A scheduler node contains either a queue, or again a packet scheduler scheduling packets from different queues. This allows a service management system to configure hierarchical scheduling or only parallel output queues. A similar interface, except that only one Virtual Processor exists per execution environment, is used for installing CPU schedulers. The `putClass` and `getClass` function allows a service to install code into the execution environment, and to get code for transporting it to other execution environments not configured to contain that particular code. Finally, the `installClassifier` function is used to install a classifier at the Virtual InPort to classify incoming packet and possibly put them in different queues waiting to get the processing resource.

```
void installPacketScheduler (Message caller, OutPort p, PacketScheduler sched);
boolean installPacketSchedulerNode (Message caller, OutPort p, SchedulerNode n);
void installCPUScheduler (Message caller, CPUScheduler sched);
void installCPUSchedulerNode (Message caller, CPUScheduler s, SchedulerNode n);
void putClass (Message caller, String name, byte[] code);
byte[] getClass (Message caller, String name);
void installClassifier (Message caller, InPort port, Classifier classifier);
```

Table 1: Configuration Interface

4.3 Service Supervision Support

In Section 3, we present different mechanisms to support service supervision. In this section the implementation of the mechanisms on the ANET platform is described. Figure 7 shows the Execution Environment part of the node architecture depicted in Figure 2. It shows an EE with service components and service management components installed, similar to the one shown in Figure 4 except that we have an operating system view of the EE. Service-Triggers use the information interface (Section 4.4) for sending service-specific or predefined events, representing service actions to the event collector. Open data structures in the service components are accessed synchronously by active service management packets to get service-specific information. Service-independent actions are recorded by the Interface-Triggers and the EE-Triggers. Interface-Triggers record actions of the service components at the interface between service components and the EE. EE-Triggers registers actions happening within the EE without any interaction with the service components. All the collected events are filtered according to the filter installed by the service management system. The events are time-stamped and stored in event queues, ready to be retrieved by event dispatchers.

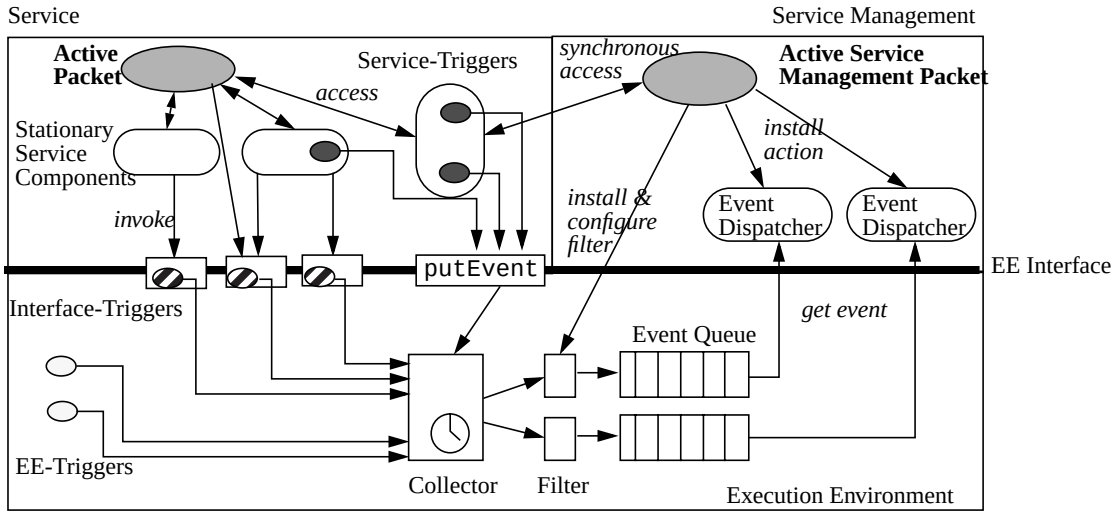

Figure 7: Event Mechanism to Support Service Supervision

4.4 Information Interface

The information interface Table 2 contains functions to get information about the resource

`void putEvent (AnetEvent evt);`
`void registerEventQueue (EventQueue queue, Filter filter);`
`void removeEventQueue (EventQueue queue);`
`ResourceAmount getOutPortResource (OutPort port);`
`ResourceAmount getMemoryResource ();`
`ResourceAmount getCPUResource ();`
`InPort getCorrespondingInPort (OutPort port);`
`OutPort getCorrespondingOutPort (InPort port);`
`Queue getCentralCPUQueue ();`

Table 2: Information Interface

consumption of the service running on the execution environment and it contains event mechanisms functions for monitoring purposes.

The `putEvent` function is used by service components to send service-specific events to the event collection point. The `registerEventQueue` installs an event filter together with an event queue into the event collection point. The events passing the filter are queued into the event queue and can be dequeued by a service management component.

The functions `getOutPortResource`, `getMemoryResource`, and `getCPUResource` allows a service to ask about the reserved resource for this execution environment. The reservation of this resource has been negotiated between the customer and the provider over the VAN management interface, and is further propagated to the nodes over the network management interface (see Figure 1).

Each Execution Environment has Virtual InPorts and Virtual OutPorts corresponding to a physical InPort and physical OutPort. Since most physical links are duplex links, a InPort and a OutPort can belong together. This correlation is used in different service control and management implementations, and is asked for by the functions `getCorresponding-InPort` and `getCorrespondingOutPort`.

A function called `getCentralCPUQueue` asks for the CPU-central queue for active packets, because the first active installation packets are stored in the CPU-central queue,

which need to run in this execution environment in order to bootstrap the service. Further, this allows a active packet to create new active packets, ask for the central CPU-queue, and put them into the CPU-central queue for processing.

5 Active Packet Tracer: An Example

We have built an active packet tracer on top of the service management toolkit, as an example application of the described supervision mechanisms and to show the flexibility gained with our service management toolkit. The active packet tracer allows a designer of an active service to follow active packets and monitor their execution on different EEs of a VAN. It shows the possibilities of event retrieval, flexible event filtering, and programmable information aggregation within the network, before displaying the information on the service management station. Further, the on the fly generation of different views can be demonstrated.

The installation of the active packet tracer service is implemented as follows. The active installation packets implementing the broadcasting mechanism distribute the code of the packet tracer service. Further, it initializes and installs a filter, which only filters out events originating from the packet tracer service itself. The filter can be configured in two different ways. First, it passes all events of services later running on the EE, if a full trace of this EE is needed. Second, it passes only events, which indicate the arrival, creation, leaving or termination of active packets. This is used in cases, where the designer is not interested on the specific execution trace of the active packet. Further, the user of the packet tracer specifies which nodes are grouped together, and which node is the representative of the group. This information is distributed to the appropriate EEs and a concentrator is installed in each representative. Note that the groups and representatives are chosen manually over the service management station user interface.

The function executed during the run-time of the tracer are the event trace recording and the concentration of the information arriving from different node. The event dispatcher cumulates the event trace for each active packet in a separate structure. If the active packet leaves the node or if it terminates, the event dispatcher sends the event trace to the group's representative. The leaving and termination of a packet is signaled by the pre-defined `PacketLeaveEvent` and `PacketTerminateEvent`. The concentrator on the representative compacts all active packet event traces such that only the arrival of the packet in the group, the leaving of the group, the creation within the group, and the termination within the group are contained in the event trace. This newly created event trace is sent to the service management station, where the event traces are displayed on the screen, as shown in Figure 8.

Our service management toolkit has the nice property, that filtering and concentration are reconfigured or exchangeable. This allows a observer of the active packet tracer to change the grouping or filtering in the nodes on the fly, an operation which is hardly possible with today's management systems.

Figure 8 shows the display on the service management station, after installing the active packet tracer and running the installation of a IP-Service. It shows a VAN with four nodes (SMgt., N0, N1, N2), where N1 and N2 are forming a group of nodes. The first trace originating in the service management station (SMgt) shows the service installation active packet flooding the VAN and installing IP-service components into the EE of the VAN. Typically, the event traces on an EE have been aggregated to an event trace containing only two events indicating that the active packet was arrived and that it left the particular EE. Further, one full event trace of an active packet running on the service management station is shown. The

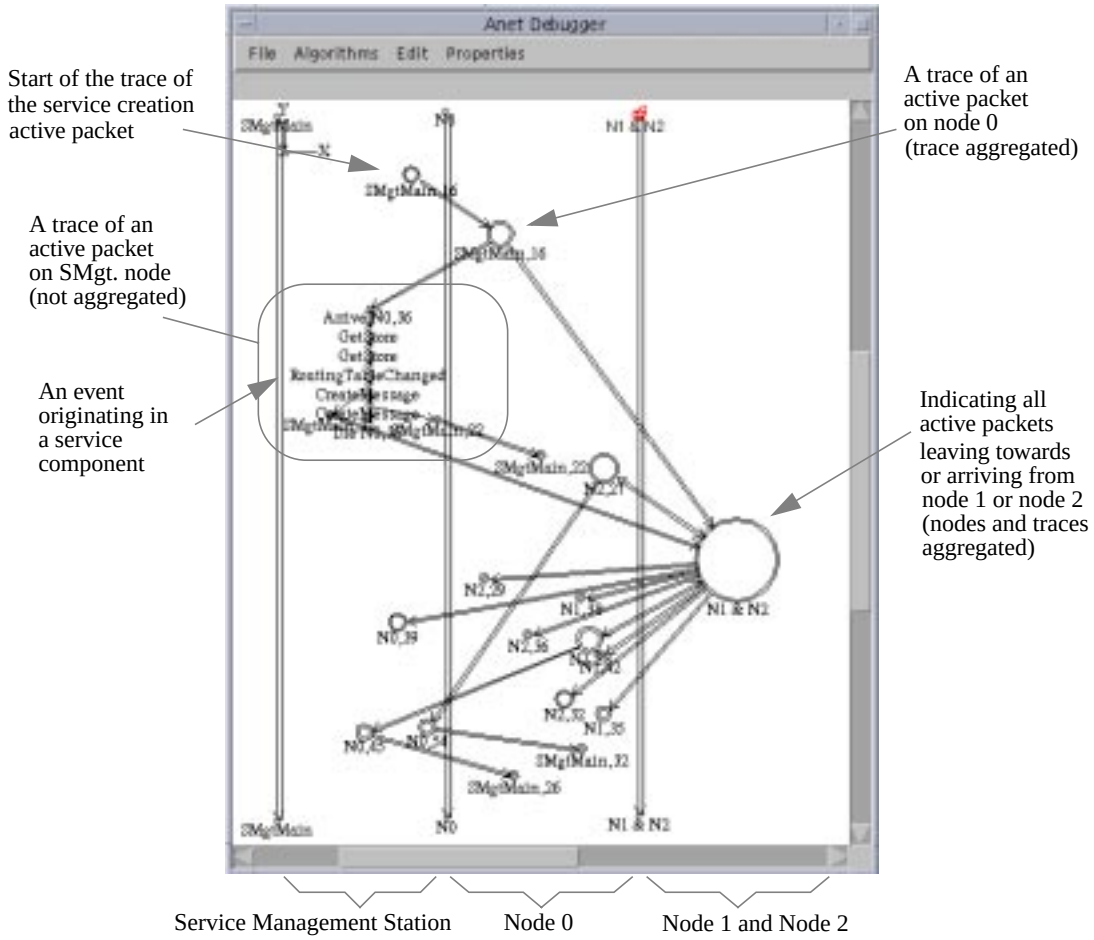

Figure 8: Example Application: Active Packet Tracer

full trace displays two accesses to memory, a routing table change, and the creation of two active packets, sent to node N0 and to one of the nodes in the group N1 & N2. Note that the routing table change event is service-specific, whereas the other events are caught by the interface-triggers (Figure 7).

6 Related Work

Network management research related to active technologies has only recently been started. Many approaches taken today can be seen as a generalization of the concept of *management by delegation* [1]. They are focusing on using mobile code for building an active management middle-ware, i.e., a software layer between the management applications and the managed objects (MOs) that represent physical or other local resources. In [2] new classes of MOs, called active managed objects (AMOs), are proposed for event discrimination and aggregation of monitoring data. The behavior of AMOs is defined in a scripting language. The scripts encoding this behavior stored as values of AMO attributes. Therefore, the behavior of AMOs can be changed using standard management protocols. A similar approach is pursued in [3], where the concept of a dynamic MO is proposed, in order to make Q-adapters programmable. A third approach replaces the management protocols with mobile agents [4][14], and the efficiency and scalability of this approach is shown in [13] and [15]. In contrast to these approaches, this paper assumes an *underlying* active networking infrastructure, and its contributions are based on exploiting the capabilities of active networking nodes.

The adaptable network control and reporting system (ANCORS) [16] 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.

The Netscript project [5] deals with management of active networks. In that project, a platform for programming network services is being built. These services can be automatically instrumented for management purposes, and corresponding MIBs can be generated. During operation, services can be managed through those MIBs. Contrary to the Netscript project, our work takes a framework approach to service instrumentation. It focuses on a flexible framework for supporting service management, with predefined instrumented service components customized through extending the components.

7 Conclusion

Rapid deployment of new services on an network infrastructure is the main driving force behind active networking research. In this work, we specifically focused on service management in an active network environment, which allows to isolate customers from each other by implementing the concept of a Virtual Active Network. Further, we showed the promising potential that active networking opens up in this area and how the technology supports mechanisms to execute service management in a very extendable, reusable, flexible, and scalable way.

The paper includes the following contributions. We have outlined a service management toolkit based on our active network management framework for a telecom environment with the benefits of

- executing service installation in provider domains without interaction with the provider;
- facilitating the design of a service-specific service management system by composing the system out of basic service management components, which can be extended and reused;
- introducing programmable event filters, management by delegation, flexible grouping of nodes represented by a representative, and grouping of events into composite events, in order to solve the problem of performance bottlenecks and scalability in service supervision;
- supporting the service supervision with mechanism to retrieve events from different services running in the EE in a very flexible and reusable way such as pre-defined events, an event collection point, and programmable filters;
- supporting the installation, upgrade, removal of services by extendable distribution mechanisms;
- supporting the collection of information from a Virtual Active Network taking advantage of easily aggregate information within the VAN using active networking technology to flexibly install and program concentrators in a customized way within the network.

Further, we have illustrated the toolkit by describing the realization of some key functions of service management on our active networking platform (ANET). And we described the implementation of an active packet tracer using and extending the service management toolkit in order to show the power of the architecture.

Our work to date opens up the way for further research. From the management perspective, some of the topics worth pursuing are:

- Automation of the instrumentation of code installed in the VAN for management pur-

poses. [5] proposes such an approach in their specific programming language.
- Automation of the grouping of nodes and events for the purpose of aggregating information within the VAN [9][10].
- Study in more depth event correlation in an active networking environment.

8 References

[1] Y. Yemini, G. Goldszmidt, S. Yemini, “Network Management by Delegation,” Second International Symposium on Integrated Network Management (ISINM’91), Washington DC, USA, 1991.

[2] A. Vassila, G. Pavlou, G. Knight, “Active Objects in TMN,” Fifth IFIP/IEEE International Symposium on Integrated Network Management (IM’97), San Diego, 1997.

[3] Y. Kiriha, M. Suzuki, I. Yoda, K. Yata, S. Nakai, “Active Q Adaptor: A Programmable Management System Integrator for TMN,” Ninth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM’98), 1998.

[4] M. Suzuki, Y. Kiriha, S. Nakai, “Delegation Agents: Design and Implementation,” Fifth IFIP/IEEE International Symposium on Integrated Network Management (IM 97), San Diego, USA, 1997

[5] Y. Yemini, S. da Silva, “Towards Programmable Networks,” Seventh IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM’96), L’Aquila, Italy, October 1996.

[6] D. Decaspar, G. Parulkar, S. Choi, J. DeHart, T. Wolf, B. Plattner, “A Scalable, High Performance Active Network Node,” IEEE Network, Vol. 13(1), 1999.

[7] AN Architecture Working Group, “Architectural Framework for Active Networks,” K. Calvert (editor), 1998.

[8] AN Composable Services Working Group, “Composable Services for Active Networks”, E. Zegura (editor). 1998.

[9] T. Kunz, “High-Level Views of Distributed Execution”, Proceedings of the 2nd International Workshop on Automated and Algorithmic Debugging, Saint Malo, France, 1995.

[10] C. Olson, “Parallel Algorithms for Hierarchical Clustering”, Parallel Computing, Vol. 21, 1995.

[11] 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.

[12] M. Brunner, R. Stadler, “Virtual Active Networks -- Safe and Flexible Environments for Customer-managed Services”, Tenth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM’99), Zurich, 1999.

[13] A. Liotta, G. Knight, G. Pavlou, “On the Efficiency and Scalability of Decentralized Monitoring using Mobile Agents”, Tenth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM’99), 1999.

[14] M. Baldi, S. Gai, G. Picco, “Exploiting Code Mobility in Decentralized and Flexible Network Management,” First International Workshop on Mobile Agents, Berlin, 1997.

[15] E. Al-Shaer, “Programmable Agents for Active Distributed Monitoring,” Tenth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM’99), 1999.

[16] 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.