ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Christoph Jossi

Management of the Distributed Traffic Control Service

Master's Thesis MA-2005-02
28 April 2005

Tutor: Matthias Bossardt
Co-Tutor: Thomas Dübendorfer
Supervisor: Bernhard Plattner

Abstract

Nowadays, no effective system exists that can counter large DDoS attacks in the Internet. Some proposed systems do not work in the real environment or need to be deployed at each router of any possible attack path in order to work. Some of the systems even react counterproductively when taking countermeasures against the reflected DDoS attack. With the Distributed Traffic Control Service a new system has been suggested. That system allows a network user to deploy services concerning his traffic throughout the Internet. These services are processed at Traffic Processing Devices (TPDs) and/or at routers that are located at the ISPs.
A network user may place a service request at the Traffic Control Service Provider (TCSP), which takes care of the services' installation. To do so, the service must be broken down into subservices that can finally be installed at the devices of the ISP. Therefore the system requires a flexible and powerful deployment.
This thesis proposes an infrastructure that is able to deploy the services. Service requests are passed to the TCSP using XML. The system then uses service descriptors to map the services to subservices. Those descriptors are also stored in XML. Furthermore, a simple TPD is designed and implemented based on the Click Modular Router technology. Finally, the configuration of the services installed at the ISP and of the simple TPD can be edited using the NetConf protocol that also has been implemented.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

A short overview of security issues posed by today's Internet is provided by the introduction. Of special interest are issues concerning the Distributed Traffic Control Service. That new approach is covered in this chapter too. Section 1.3 outlines the work contributed by this thesis. The final section gives information about the layout of the documentation.

1.1 The Internet Situation Today

With the steady growth of the Internet, one is faced with new problems. One of them, the evident advantage of the Internet is also its biggest handicap: Rapid distribution of information is allowed by the communication over the Internet. Unfortunately exchanging information includes viruses, worms and trojans.

Because the Internet Protocol only provides the required services to communicate, there are no special features available within the network for Internet users. The tracking of packets would be one feature and so would be time stamping to discover delays.

Security is the main problem of today's Internet. One part concerns viruses, worms and trojans. While they present many problems to users, they can be met with various tools present today. The second kind is about attacks, prepared and launched against a defined target. Since it does not make much sense to launch an attack against a normal user, it sure is more interesting, profitable or satisfying to damage a company or organization.

For some companies, the Internet as it is, represents an important market to sell their products. Whereas this market is an additional market for some firms, for others it is the only one. Therefore some companies rely heavily on the Internet and that is also why their servers must run 24/7. An attack against their servers could cost the company a lot of money or even ruin them. That is way they represent an optimal target for blackmailing, opening the market to a third party, selling a system which should prevent the companies from financial loss.

Suffering a DDoS attack nowadays is not only a problem of server resources, but also of downlink bandwidth. The malicious traffic cannot be blocked until it reaches the company's network, where it is redirected through the firewall. Thus one problem is to block the malicious traffic before it reaches the company's downlink. As will be shown in Chapter 2, some systems are trying to do exactly that, but fail when it comes to the reflected DDoS attack or have other drawbacks.

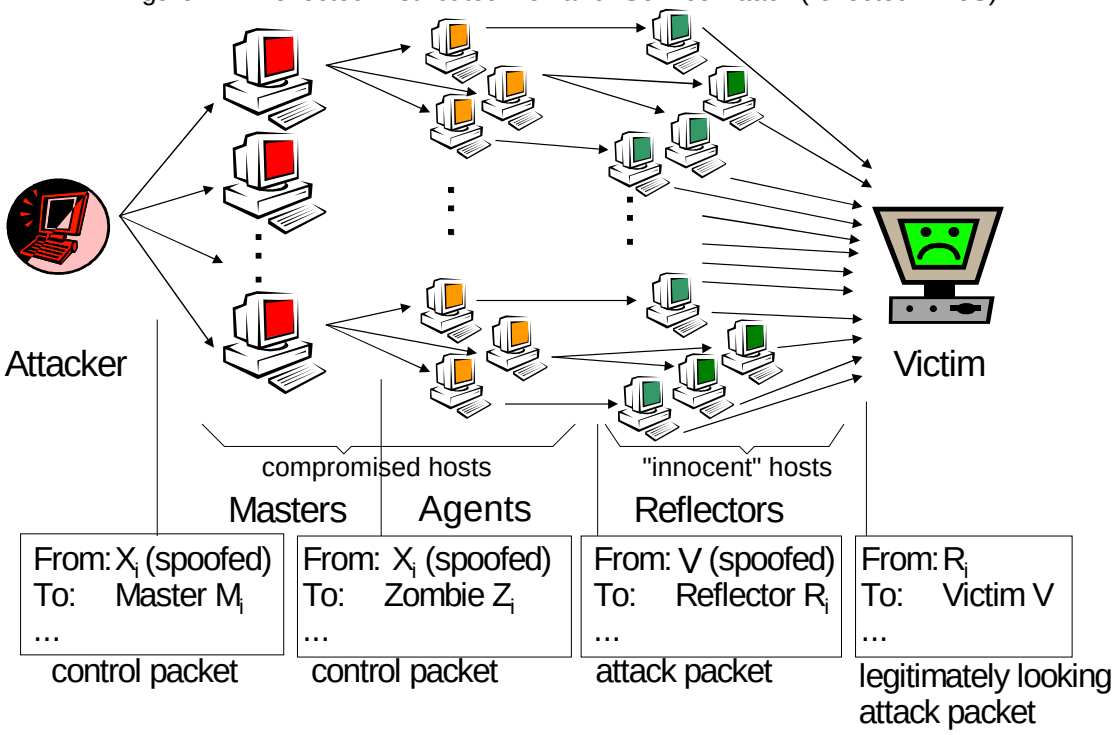
1.2 Distributed Traffic Control Service

To mitigate some of the mentioned attacks, [1] proposes a Distributed Traffic Control Service. The idea is to offer Internet users the possibility to process their IP packets not only inside their network, but also at the ISPs. Motivation for this solution among others is the counteraction of the reflected DDoS attack.

1.2.1 Reflected DDoS, a New Attack

Reflected DDoS is an enhanced DDoS attack. Unlike in a simple DDoS attack, the victim is not flooded directly by the attacker or its agents. It receives the packets via a reflector. Figure 1.1 shows the scenario. A search engine is an example of a reflector. The malicious packet sent from the zombie to the reflector contains a normal request but its source IP address is spoofed and replaced with the victim's IP address. The search engine now sends the response to the victim and that malicious traffic uses precious downlink bandwidth. Reflected DDoS attacks are annoying because the victim filtering the attack packets can no longer use the reflector's service. Ingress fitering at the ISP would immediately stop reflected DDoS attacks, but there is yet no need to do this. Since Ingress filtering does not directly protect the own customers and since it is a costly operation that has to be performed on every IP packet, the ISPs are not willing to do it.

Figure 1.1: Reflected Distributed Denial-of-Service Attack (reflected DDoS)



1.2.2 A New Solution

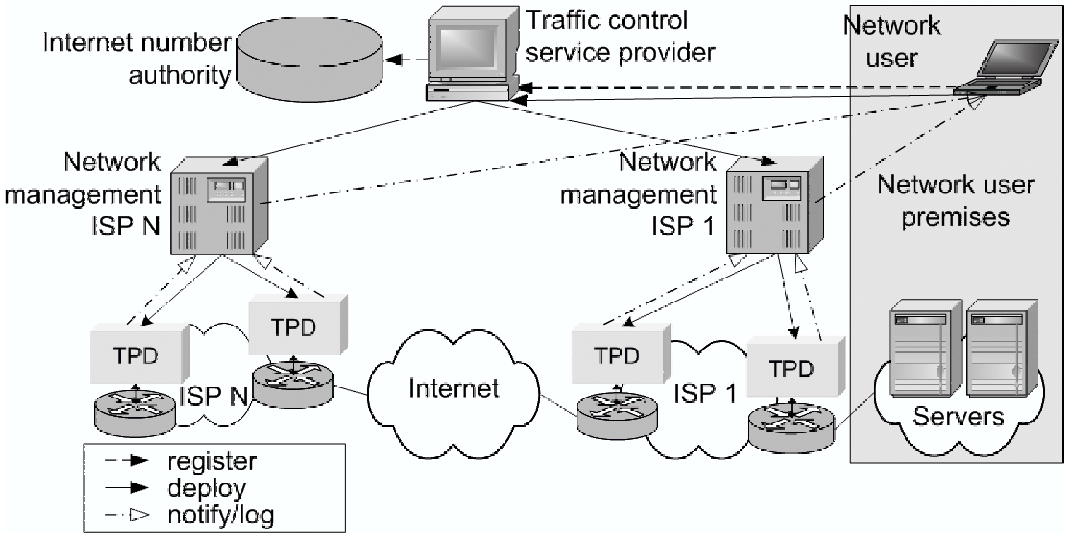
The Distributed Traffic Control Service may also perform ingress filtering. Instead of having the ISP allowing or refusing IP packets to enter the Internet based on whether IP packet comes from the correct network or not, our system leaves it to the owner to declare where IP packets are allowed to enter the Internet and where not. This leaves the responibility to the customer. He can decide what should be done with his traffic.

It has already been suggested that network users should be able to process their packets at the ISPs. Two questions emerge. First, what does the ownership of a packet mean and second, why should the ISP allow a network user to install filter rules at the ISP's network?

The ownership of an IP packet is the core concept of [1]. A normal IP packet is sent from the sender through the medium to the receiver. Both are identified with the source IP address and destination IP address respectively. On its way, the packet belongs to the sender first, then secondly to no one and finally to the receiver. The restriction that only owned packets can be modified is very important. It prevents a network user from bringing harm to any other network user than himself. He can only, positively or negatively, affect his own traffic. Therefore the user is not interested in misusing the system. Protection of the Internet is a main concern of

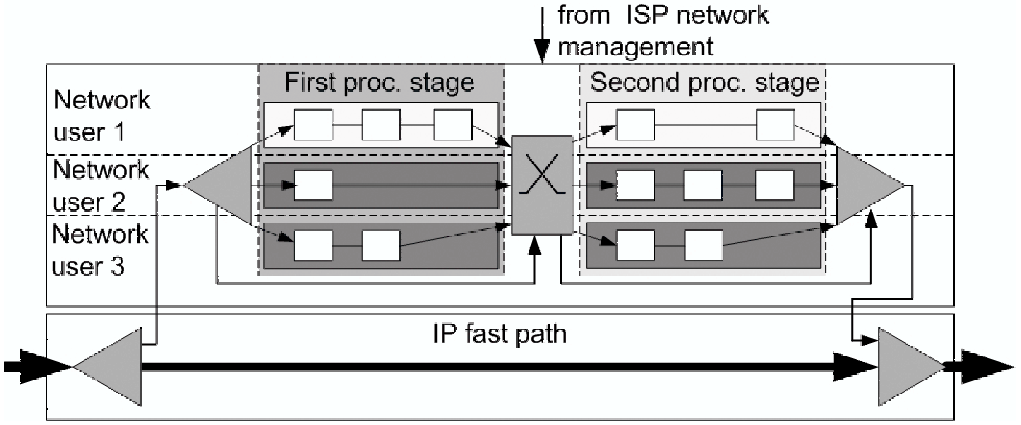
this proposal. The system should not be able to harm the Internet. Of course more than the ownership restriction is necessary to achieve that.
Which brings us to a second point. Why should the ISP allow a network user to install filter rules? One thing that could be offered is money. Of course not every network user could maintain a contract with every ISP, suggesting the use of a third party, that maintains contracts with the network users and the different ISPs. The third party is called Traffic Control Service Provider (TCSP). Network users can then request a service from the TCSP.
Allowing the TCSP to install filter rules needs the ISP to trust the TCSP that he checks whether the network user is truly an owner of the packets that will be filtered. To do so the TCSP contacts the Internet Number Authority. Figure 1.2 gives an overview of the system.

Figure 1.2: Network Model of the Distributed Traffic Control Service



The next step takes a look at how and where packets should be processed. [1] proposed to redirect packets from the fast IP path to a slower path, where they are processed and finally sent back to the router. This slower path can be located locally at the router or at a Traffic Processing Device (TPD). Expensive operation are preferably performed at the TPD. Figure 1.3 shows this redirection mechanism.

Figure 1.3: Processing of Packets



There are other interesting restrictions to the system. In order to prevent the system from harming the Internet, the possible processing operations are limited. The system is not allowed to:

- Change the Time-To-Live field of IP packets
- Change the packet's source or destination IP address

- Enlarge the packet
- Create new IP packets

1.3 Contribution

While [1] defines the general system components, the communication between the TCSP and the ISP is still an open issue. Another unsolved problem is the deployment of services. Furthermore no Traffic Processing Device has yet been defined or implemented.
This work will cover the following points:

- *Defining the service deployment*
- *Implementing the service mapping*
- *Design and implement a simple TPD*

1.4 Layout of the Report

Chapter 2 covers some related work. The system architecture is defined in Chapter 3. Chapter 4 explains the information model including the sophisticated service descriptors. Throughout Chapter 5 the service deployment is specified. The definition of the TPD based on a Click Router can be found in Chapter 6. The work is concluded in Chapter 7 where further work is brought up as well.

Chapter 2

Related Work

This chapter refers to some other projects related with the Distributed Traffic Control Service. It covers an IP packet traceback system, two systems that present a solution to counteract attacks and another system that tries to prevent attacks by acting proactively, that means to shield the server in a way that it is very hard to launch an attack.

2.1 Single-Packet IP Traceback

[5] proposes a system to traceback IP packets. In that system every SPIE-enhanced router maintains a cache of packet digits for recently forwarded traffic. The whole data storage is managed by a Data Generation Agent (DGA) which can be either a software agent, an interface card plug to the switching background bus or a seperate auxiliary box. If some intrusion detection system discovers an offensive packet it can place a request at the SPIE Traceback Manager (STM). The STM contacts several SPIE Collection and Reduction Agents (SCARs), that concentrate the information produced by the DGA, and tries to build the attack graph. When the STM has done its job it returns the attack graph to the client.

2.2 Aggregate-Based Congestion Control: Controlling High Bandwidth Aggregates in the Network

Recent events have illustrated the Internet's vulnerability to both Denial-of-Service-Attacks (DoS) and flash crowds in which one or more links in the network become severely congested. This congestion is neither due to a single flow nor to a general increase in traffic, but to a well-defined subset of the traffic - an aggregate. The aggregate-based congestion control (ACC) proposed in [6] tries to detect and control such high bandwidth aggregates.
If a router is congested, the first problem is to detect the responsible aggregates. This is tricky because the overload could may be chronic due to an underengineered network, or unavoidable, e.g. as a shift in load caused by routing around a fiber cut. An attacker might also cluster the traffic, e.g. Distributed DoS, and/or vary the traffic to escape detection. So this is a sophisticated problem.
Once one or several malicious aggregates have been detected, the system rates these streams, allowing other traffic to pass unaffected. This means that the bad streams have an evidently higher drop rate than the normal traffic.
The proposed system implements another mechanism: pushback. A router that has detected and rate limited some bad aggregates may ask its neighbour router to do the same with this aggregate. Of course the neighbour router can again ask its neighbour to do the same. With this mechanism the aggregate can be limited as close to the source as possible and the routers released from bad aggregates.
The backbone bandwidth is nowadays not the bottleneck of the Internet looking from a company's point of view. More important is the protection of its uplink. Anyhow, this is possible with this system, if it is able to detect the right aggregates. The pushback mechanism is nice, but counterproductive against reflected DDoS attacks. By pushing the filter nearer to the reflector,

its service can no longer be used by all network users. Still, the most difficult part is to detect the bad aggregates and by all means prevent to produce false positives.

2.3 Active Internet Traffic Filtering: Real-time Response to Denial-of-Service Attacks

[7] proposed this system that detects malicious flows and and tries to eliminate them as close to the origin as possible. A victim can ask the next gateway to perform some filter rule to protect the victim's uplink.
In an AITF network each Autonomous Domain (AD) has filtering contracts which each of its end-hosts and each neighbour Autonomous Domain directly connected to it. This is needed to protect the router in the AD from consuming to many CPU cycles to process filter requests. Such a filtering request is initially sent from the victim directed to the victims's gateway. While this gateway installs the filter rule temporarily, it is forwarded to the attacker's gateway to be installed as close as possible to the attacker. The attacker's gateway asks the attacker to stop. If the attacker does not stop, it risks being disconnected. If the victim's gateway still receives malicious traffic after a certain time, the game starts all over again, with the victim's gateway as new victim and the attacker's gateway as new attacker.
One of the main problems of this system is that every AD must trust one another to install non-malicious filter rules. Furthermore it is not sufficient if AD1 trusts AD2 and AD2 trusts AD3, because of the pushback mechanism a filter rule initiated by AD1 can be passed through AD2 to AD3. Note that bad filter rules can severely harm the Internet.

2.4 Mayday

[8] proposes a proactive solution. Instead of trying to detect an attack, the server is protected against attacks. This means more ressources, as some kind of filter has to be running all the time. But let us look at their idea.
A server is protected by a so-called filter ring, some routers that provide the server's Internet connectivity. The ISP, owner of these routers, is willing to perform some filtering at those routers on behalf of its client.
Around the filter ring an overlay network is built. A client can talk to some overlay nodes, the ingress node and from there a request is routed through the overlay network until it reaches an egress node. Such an egress node, also part of the overlay network is able to talk with the server because it got a so-called leightweight authenticator. As a leightweigth authenticator the egress source address could be used. Other solutions are proposed in the cited source.
The inventors describe their service as very flexible. The designer of the system can choose what leightweight authentication he wants to use, how to route in the overlay network and how to authenticate clients at the ingress nodes. The choice of how secure the system should be is of course a trade-off between security and performance.

2.5 Positioning of the Distributed Traffic Control Service

The Single-Packet IP Traceback has been introduced because the Distributed Traffic Control Service can also trace IP packets. Compared to the proposal of [5] our system does not have to log packets of all users. Furthermore we do not need every router to be enhanced with the SPIE engine. If some ISPs do not enter contracts with the TCSP, we are not able to trace IP packets that entered or left through this ISP but we are still able to trace the other IP packets.

The aggregate-based congestion control needs the router to be able to detect bad agggregates. This is as explained quite difficult. Furthermore the different ISPs must trust one another because of the pushback mechanism. The system proposed by [1] needs only each ISP to trust the TCSP.

The AITF network has one major drawback. This system harms the Internet when countering the reflected DDoS attack. Using this system the AD might disconnect the reflector.

If the first three proposed systems should work fine, all routers need to be enhanced with the proposed logic. This is not the case for our system. It works just as fine with only some of the ISPs. Of course the system is more attractive to network users if it is deployed throughout the Internet.

Chapter 3

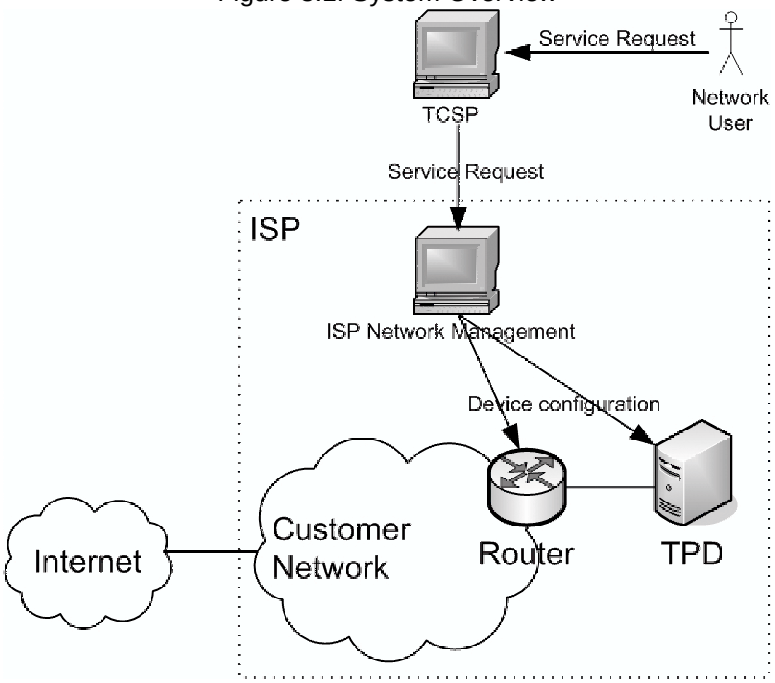
System Architecture

Section 3.1 gives an overview of our system focusing on the ISP side. The purpose and tasks of the different system components are outlined in the following sections.

3.1 Overview

As we focus on the ISP side of our system, Figure 3.1 shows the interesting components only. It consists of the *TCSP* and an *ISP* that can be further divided into the *Network Management* and the devices. Figure 3.1 shows only two devices: A *router* and a *Traffic Processing Device* (TPD). The figure additionally shows a network user that can send a service request to the TCSP.

Figure 3.1: System Overview



When a network user requests the TCSP to install a service, it must be distributed to the ISPs. Since not every ISP offers the same services, the service must be mapped onto subservices that can be performed by the affected ISP. Furthermore the ISP might not be suited for a service. For instance an ingress filter service should only be installed at ISPs maintaining stub networks. This means that some kind of selection must be done.
When the service request arrives at the ISP it must be mapped onto a combination of a router and a TPD. Since different combinations of different routers and TPDs are possible each combination may again offer different services. Additionally devices may not be in the same context.

For instance a router may be located at the ISP network border and be linked to customers and/or other ISPs.

3.2 TCSP Subsystem

This section specifys the different tasks that have to be performed at the TCSP.

3.2.1 Tasks of the TCSP Subsystem

When a network user sends a request to the TCSP, the TCSP processes the request as follows:

- *Authenticate the service request*
  To ensure that the service request has indeed be sent by the owner of the affected IP addresses a strong authentication is needed.
- *Validate the request*
  To validate a request, we need to know what information has to be passed. This is different for every service. There are of course some fields that are always present. This information is provided by service descriptors. For each service one or more service descriptors describe how the servide request must look like. This includes the definition of parameters that can or must be provided.
- *Check if the network user is allowed to deploy that service (with respect to the contract between TCSP and network user)*
  For every network user request, we must test whether the network user is allowed to request this service or not. This will depend on the contract between TCSP and network user and should be accessible through a database.
- *Check if the network user owns the affected IP address range*
  To check if a network user truly is the owner of an IP address range, the Internet Number Authority (INA) must be contacted. How often the INA will be contacted must still be specified.
- *Install the service at the ISPs*
  For every ISP the requested service is mapped onto services offered by the ISP. If a suitable configuration has been found it is sent to the ISP. It is possible that the service should not be installed at ISPs with a specific context. Therefore a selection must be done. The use of several service descriptors allows us to propose different mappings, hoping that at least one might be suited for a specific ISP. Restrictions concerning the ISP's context are also included in the service descriptor that is covered in Chapter 4.
- *Update database*
  After the configuration of the ISPs, the TCSP database must be updated.
- *Inform network user*
  After the TCSP processed the request, the network user is informed if the request succeeded or failed.

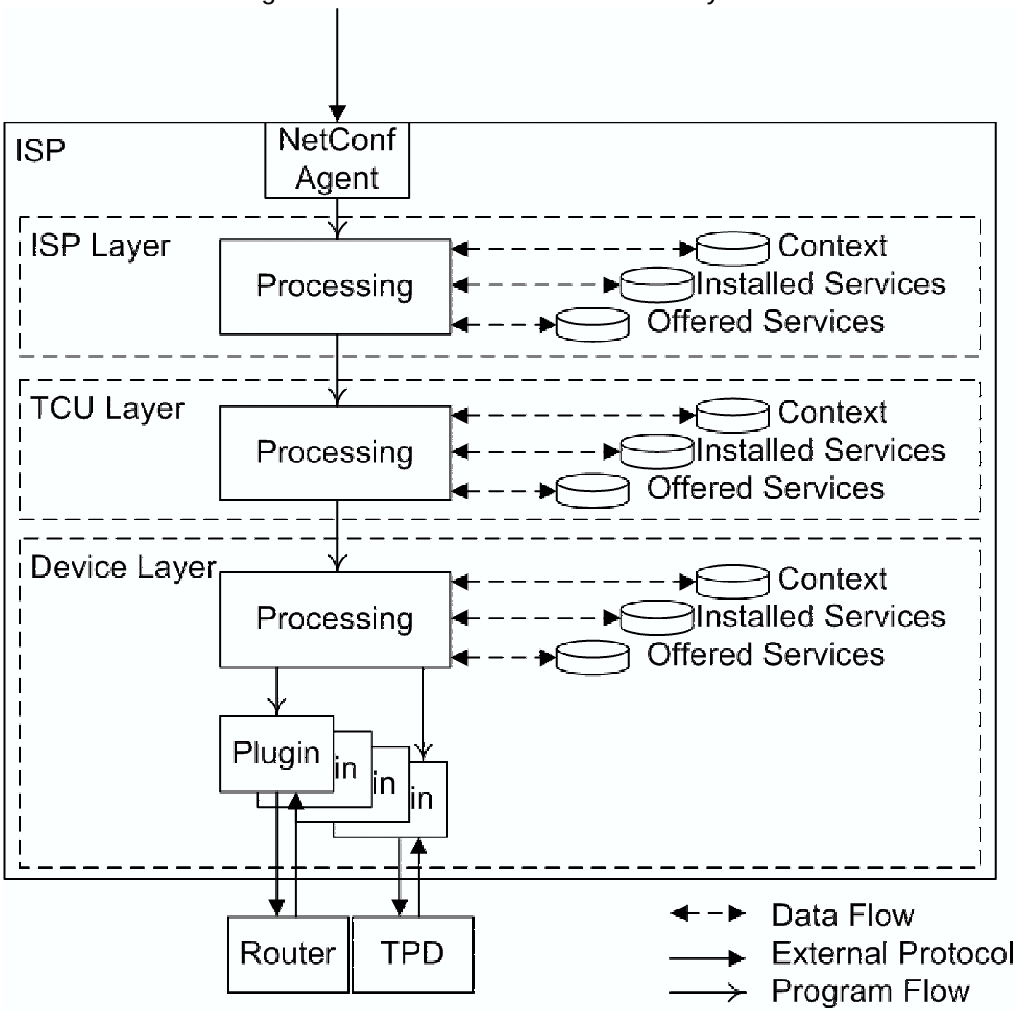
3.3 ISP Subsystem

This section discusses the ISP subsystem showed in Figure 3.2. The protocol used to configure the ISP Subsystem is presented in the first section. Then the architecture is discussed.

3.3.1 ISP Subsystem Configuration over the NetConf Protocol

To configure the ISP Subsystem the NetConf Protocol is used. This is a protocol currently in development that offers some interesting features.
NetConf is a protocol based on XML. It performs a Remote Procedure Call (RPC) that is also encoded in XML. The RPC is then sent over a transport protocol to the Agent. NetConf allows

Figure 3.2: Architecture of the ISP Subsystem



us to specify our own data model adjusted to our needs. Thanks to XML the data model could easily be expanded. NetConf defines several operations that are used for session establishment and tear-down, editing the configuration, etc. The used operations are shortly discussed.

- *edit-config*
  With this function the configuration can be edited. Note that NetConf supports different configurations. Examples are 'start-up', 'running' and 'candidate'. Edit-Config allows the manager to create, edit or delete certain parts of the configuration, not affecting the rest.

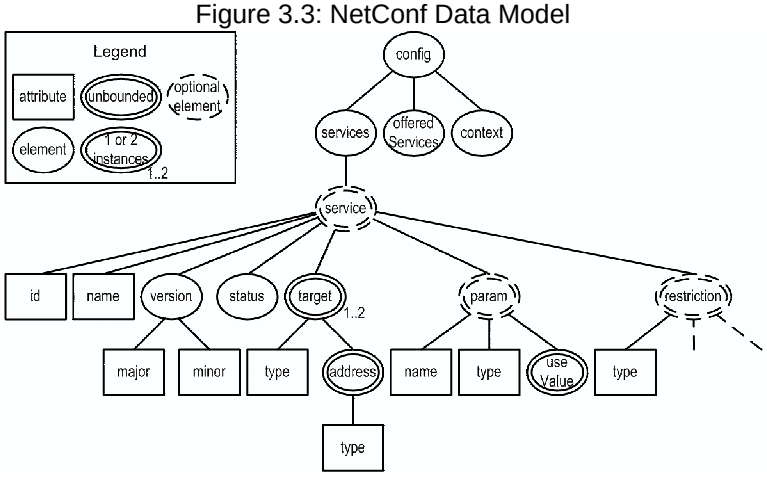
- *get-config*
  This call returns the current configuration. If filters are specified, not the whole configuration will be retrieved, but only the part matching the filter.

- *get-state*
  Retrieve state information with this function. You may again specify filter rules. This function is used to retrieve state data logged by installed services.

- *close-session*
  This function closes an established session with the agent. Previously sent requests are carried out.

With the use of NetConf, the Data Model must be specified. It is shown in Figure 3.3.
Examples for NetConf messages that are used to edit or retrieve the configuration can be found in Section 3.4.

Figure 3.3: NetConf Data Model



3.3.2 The Traffic Control Unit

One thing is very important when performing an ingress filter service for a network user. His uplinks should not be filtered. This means that a service should be mapped onto the interface of a router rather than onto the router itself. Every interface is in a different context as the IP addresses differ. Furthermore we may imagine a TPD that has more than one interface to where packets can be redirected and a router or router interface may send the IP packets to a specific TPD interface. To solve this issue we introduce the Traffic Control Unit (TCU). Table 3.1 resumes the properties of that unit. A TCU connects the interface of a router with the interface of a TPD.

Table 3.1: Properties of the Traffic Control Unit

property	comment
routerIfAddress	the router's interface address
routerId	the associated router
tpdIfAddress	the tpd's interface address
tpdId	the associated tpd
context	other contextual information

Therefore the router interface as well as the router itself and the TPD interface as well as the TPD itself must be specified. Additionally the TCU has an own context. For instance the property link type of a router interface can now be associated with its TCU.

3.3.3 Tasks of the ISP Subsystem

When an ISP receives a service request from a TCSP it is processed as follows:

- *Authenticate the service request*
  To ensure that the service request has indeed be sent by the TCSP specified authentication between the TCSP and the ISP Management is needed.
- *Validate the request*
  The service request can again be validated using a service descriptor.
- *Check if the TCSP is allowed to deploy that service (with respect to the contract between TCSP and ISP)*
  An ISP may allow several TCSPs to install filter rules. Since not every TCSP may have the same rights, they must be checked.
- *Install the service at the devices*
  The requested services must be mapped onto services that are offered by the devices. Every device is in a different context and therefore not every service request will be passed.

Since not every device can be configured using services and over the same protocol, we need to transform the service request into device specific configuration commands.

- *Update database*
  After the configuration of the devices, the ISP database must be updated.
- *Inform network user*
  After the ISP processed the request, the TCSP is informed if the request succeeded or failed.

3.3.4 Architecture of the ISP Subsystem

The ISP Layer actually is what the TCSP sees from the ISP. It offers services to the TCSP and keeps track of all installed services. Additionally the ISP Layer maintains a context database that contains information concerning the ISP. For instance the ISP's network type can be of type 'stub'. Furthermore the ISP Layer contains service descriptors associated with the offered services.
An incoming service request is in a first step mapped onto a TCU using service descriptors. The TCU's context or the TCU's offered services may force the ISP Layer to skip this TCU. If it is not skipped the ISP Layer generates the new service requests that will be passed to the TCU Layer using the service descriptor, the service request and possibly additional information from the context.
To be able to access the ISP Layer via the NetConf Agent, the Processor interface must be implemented that is described in Table 3.2. These are the only public functions of the ISP Layer.

Table 3.2: Processor Interface

function signature
public RpcReply processEditConfigRequest(Rpc rpc)
public RpcReply processGetConfigRequest(Rpc rpc)
public RpcReply processGetStateRequest(Rpc rpc)

The TCU Layer maps the requested service onto the devices with respect to the devices' context and their offered services. It contains context information of the TCUs. For instance the interface address of the TPD can be accessed through the context database. Again service descriptors are used to map the service onto services performed at the router or at the TPD. The created service requests are then passed to the Device Layer.
The TCU Layer provides the public functions presented in Table 3.3. The first function is used to change the configuration while the other ones are used to retrieve information from the TCU Layer.

Table 3.3: TCU Layer/Device Layer Interface

function signature
public void processEditConfig(Services services) throws NcException
public UseValue[ ] getContext(String contextPath)
public boolean offersService(String name, int majorVersionNr, int minorVersionNr)

The Device Layer again has knowledge about the services installed at every device, the context information and offered services per device. An incoming service request is again validated with a simplified service descriptor but no more mapping is done. It is important that the Device Layer maintains the configuration of each device because this is the last time the configuration is available in a general language before it is translated into device specific configuration commands.
The Device Layer is accessed through the same functions as the TCU Layer, that are presented in Table 3.3.

Listing 3.1: NetConf: The Manager's Hello Message

```
<hello>
 <capabilities>
  <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
 </capabilities>
</hello>
```

Listing 3.2: NetConf: The Agent's Hello Message

```
<hello>
 <capabilities>
  <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
 </capabilities>
 <sessionID>1</sessionID>
</hello>
```

The Plugin translates the general configuration commands into device specific ones. Furthermore it assures the communication with the device. Using Plugins allows us to easily add new devices because only the Plugins have to be written.
After a Plugin is loaded it can be accessed through the Plugin Interface that requests a Plugin to provide the function defined in Table 3.4.

Table 3.4: Plugin Interface

function signature
public void processEditConfig(Services services) throws NcException

3.4 NetConf Messages

This section shows the different messages that are used to manage the ISP subsystem. The first two messages that are exchanged are the two hello messages. Listing 3.1 shows the manager's message and Listing 3.2 shows the message sent by the agent. Note that the agent's message contains the sessionID. The passed capability is the base capability of the NetConf Protocol and must be coped by both the manager and agent.
To create, delete or edit a service the attribute 'xc:operation' can be used. This attribute defines the operation for its element. Our data model defines, that only the service element may contain such an attribute. Any operation defined in child elements are ignored. The possible operations are 'create', 'merge' and 'delete'. Use 'merge' to edit some child elements of the service. If no operation is passed inside a service element, the optionally passed default-operation is used or if none is specified, 'merge' is used. Listing 3.3, Listing 3.4 and Listing 3.5 show the associated messages.
To retrieve the service configuration use the message presented in Listing 3.6. Listing 3.7 shows an example of a returned configuration.
Another message you may use is the one presented in Listing 3.8. It will return the whole state data of the ISP.
When the manager effected all operations on the ISP configuration he may close the session. Listing 3.9 shows his message. In any case of a rpc a rpc-reply is returned. You have already seen one that has been returned using the 'get-config' call. For other calls a simple okay message (Listing 3.10) is passed to report that everything went alright. However there might occur some problems while trying to edit or get the configuration. In this case the agent returns an error message (Listing 3.11) containing hopefully enough information to tell the manager about what went wrong.

Listing 3.3: NetConf: Creation of a Service

```
<rpc message-id="1">
 <edit-config>
  <target><running/></target>
  <default-operation>merge</default-operation>
  <config>
   <services>
    <service id="1" name="filter" xc:operation="create">
     ...
    </service>
   </services>
  </config>
 </edit-config>
</rpc>
```

Listing 3.4: NetConf: Modification a Service

```
<rpc message-id="2">
 <edit-config>
  <target><running/></target>
  <default-operation>merge</default-operation>
  <config>
   <services>
    <service id="1" xc:operation="merge">
     ...
    </service>
   </services>
  </config>
 </edit-config>
</rpc>
```

Listing 3.5: NetConf: Deletion of a Service

```
<rpc message-id="5">
 <edit-config>
  <target><running/></target>
  <default-operation>merge</default-operation>
  <config>
   <services>
    <service id="1" xc:operation="delete">
     ...
    </service>
   </services>
  </config>
 </edit-config>
</rpc>
```

Listing 3.6: NetConf: Get the Current Service Configuration

```
<rpc message-id="3">
 <get-config>
  <source><running/></source>
 <get-config>
  <filter>
   <services/>
  </filter>
 </get-config>
</rpc>
```

Listing 3.7: NetConf: Returned Service Configuration

```
<rpc-reply message-id="3">
 <data>
  <services>
   <service id="1" name="filter">
    ...
   </service>
   <service id="2" name="logger">
    ...
   </service>
  </services>
 </data>
</rpc>
```

Listing 3.8: NetConf: Retrieve the State Data

```
<rpc message-id="4">
 <get-state/>
</rpc>
```

Listing 3.9: NetConf: Session Tear-down

```
<rpc message-id="6">
 <close-session>
  <sessionID>1</sessionID>
 </close-sesssion>
</rpc>
```

Listing 3.10: NetConf: Okay Message

```
<rpc-reply message-id="6">
 <ok/>
</rpc>
```

Listing 3.11: NetConf: Error Message

```
<rpc-reply message-id="5">
 <rpc-error>
  ...
 </rpc-error>
</rpc>
```

Chapter 4

Information Model

Chapter 3 outlined the system architecture. The service descriptors have been introduced that play an important role during the service deployment. This chapter outlines the information that is exchanged between the previously defined layers. Of special interest is the service descriptor.

4.1 Overview of the Information Model

Figure 4.1 shows the information model of our system. Every Layer receives a service request that contains information about which service should be installed and how it should be parametrisized. All requests except the one sent to the TCSP may further contain deployment information used to decide where to install the service and where not.
To map the requested service onto offered services of the lower layer, the service descriptors are used. They contain information on how to map the requested service onto lower layer services. This includes the mapping of the parameters. Since additional parameters may need to be specified the ISP context provides some information about the ISP.
To deploy the services, context information from the lower layer is needed.

4.2 Service Descriptor

This section describes the service descriptor. A first approach discusses the general service descriptor. The minor changes for each layer are outlined in Subsection 4.2.2.

4.2.1 General Service Descriptor

Chapter 3 already outlined that the service requests are sent over NetConf, which means that they are available in XML format. Since the service request and the service descriptor are merged to form the new service request for the lower layer, it is nice to have the service descriptor as well available in XML. [4] also proposes service descriptors in XML format.
The following demands should be met by the service descriptor:

- Define the incoming service request
- Map the service on one or more subservices
- Add deployment information

Figure 4.1: Information Model

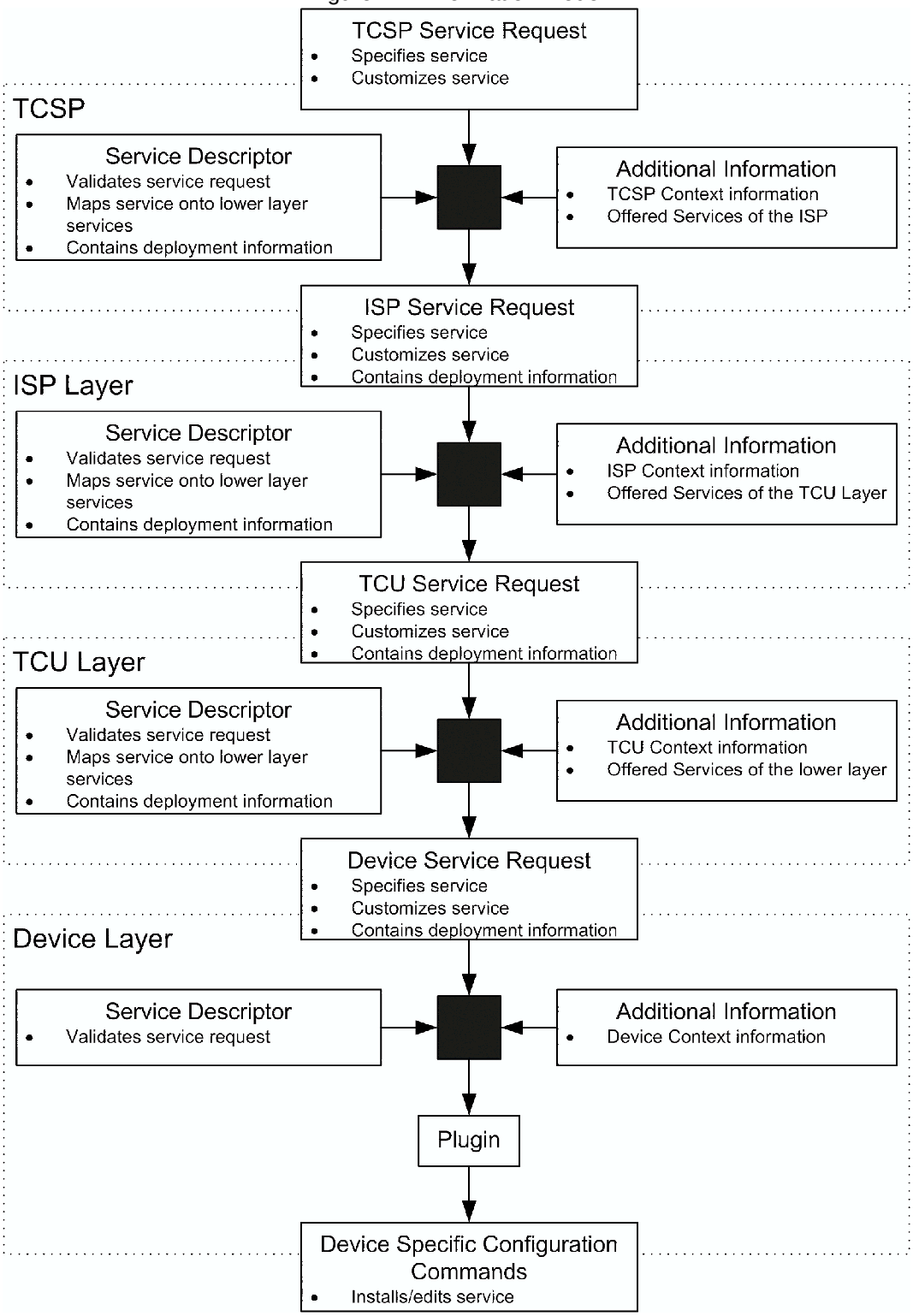


Figure 4.2: XML Schema of the General Service Descriptor

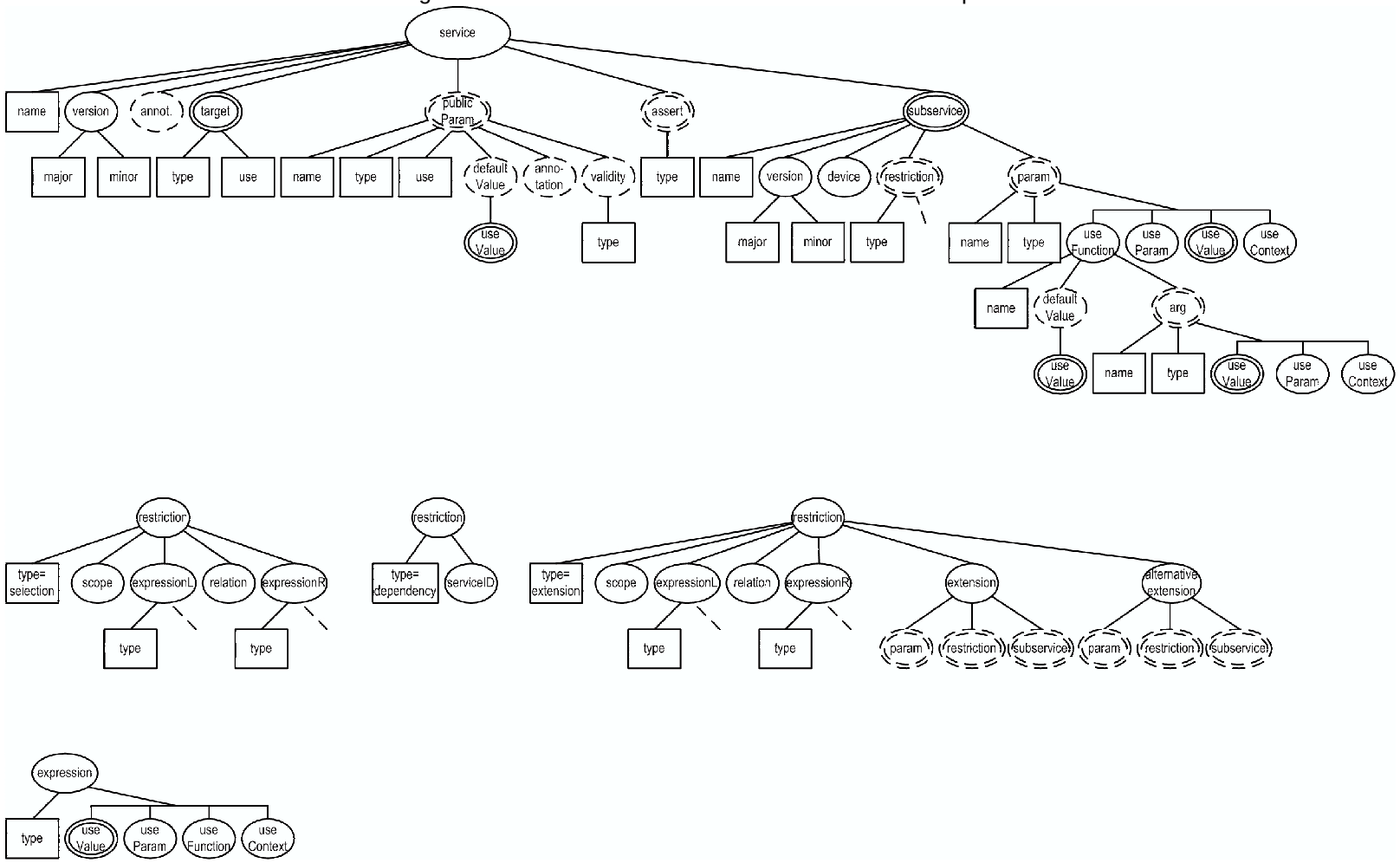


Figure 4.2 shows the complex XML schema of the general service descriptor. The elements are shortly discussed in the following list.

- *Name Attribute And Versioning*
  The attribute name and the version element with its major and minor number are used to identify the service. The name describes the service's base functionality and the numbers give us its current version.

- *Annotation*
  Figure 4.2 defines two annotations. One belongs to the service and the other to the public parameter. Both are used to describe the service and the parameter respectively. This is an optional element, but very useful to communicate with the user.

- *Target*
  The target element specifies has two attributes that specify what targets can or must be passed. The type specifies whether it is a source or destination target. The attribute 'use' can take two values, either 'optional' or 'required'. A required target has to be present in the service request.

- *Public Parameter*
  The publicParam element is used to define the parameters that a service request may contain. All these parameters can be set by the user. A parameter is identified by its name. If the 'use' attribute's value is 'required', the network user has to specify the value. Of course a parameter can also be of use 'optional'. In any case, parameters with the same name have to be of the same type. This is an additional control.

  Imagine a service filtering packets. The filter rule can be specified through the parameter 'filterRule'. If no value is passed the parameter will not be present and this leads to the blocking of all packes.

  In some cases it is more accurate to use a default value instead of nothing. The publicParam provides an element defaultValue where useValue elements can be filled in. An example would be a service blocking http requests where the user can specify a port number that should pass the filter. If no port number is specified, the default value 80 could be used.

- *Subservice*
  With this element the mapping from the incoming service to the lower layer service is done. The lower layer service is specified with its name and version. There are two more elements in this subservice: Parameters and restrictions.

- *Param*
  The parameter element is used to set the value of a parameter of the outgoing service request. The values of a public parameter can be used refering to it with the useParam element or values can directly be filled in using useValue elements. To enlarge the possibilities we provide the elements useContext and useFunction.

  UseContext refers to some context accessed by the specified path. Such elements are replaced with useValue elements during the processing.

  With useFunction a fuction can be called and the return value will be stored in useValue elements. To make the useFunction element powerful, it is important to be able to pass arguments. An argument can be a set of constant values, information from the context or values from parameters. If the function call fails for whatever reason a default value can be specified. If this is not the case, an error will occur.

- *Restrictions*
  The restriction elements are used to select which lower layer instance should or can perform the request and to add some auxiliary parameters, subservices or restrictions. We define three different restrictions to do so:

  - *selection* - used to select the lower layers that should perform the service based on a condition

  - *dependency* - also used to do a selection but it is done according to the passed serviceID
  - *extension* - if the passed condition is met, the extensions are added to the service

- *Dependency Restriction*
  The dependency restriction assumes that each service has a serviceID. Furthermore there are some services that only have to be installed at places where another service (that can be referenced by its serviceID) is already running. For instance a trigger service, that monitors the traffic and eventually triggers another service has only to be installed where the triggered service is installed too.

- *Selection Restriction*
  A selection restriction applies always to a certain scope. The selection restriction is always processed a layer above the one refered with scope. For instance if scope is 'isp', than the selection restriction will be processed at the tcsp. This implies, that the TCSP has some knowledge about the ISPs. Inside every selection restriction there is a condition and if this condition is met the lower layer instance is selected to perform the service. The condition can be put together from the three elements expressionL, relation and expressionR. The condition is met if the following expression is true:

    ```
    expressionL (relation) expressionR
    ```

  Note that all expressionL and expressionR contain a set of values. Therefore we compare sets. The relation can take the following values:

  - is - the two sets are equal
  - isIn - the first set is entirely contained in the second one
  - isNot - the two sets are not equal
  - isNotIn - no element of the first set appears in the second set

Listing 4.1 shows an example of a service descriptor.

Listing 4.1: Service Descriptor Example

```
<service name="ingressFilter">
 <annotation>This service performs ingress filtering.</annotation>
 <version major"1" minor="3"/>
 <target type="source" use="required"/>
 <publicParam name="uplinkRouterIf" type="ipAddress" use="required">
  <annotation>Pass the the IP-addresses of your router uplink
           interfaces here.</annotation>
 </publicParam>
 <subservice name="ispFilter">
  <version major="1" minor="0"/>
  <restriction type="selectionRestrictionType">
   <scope>isp</scope>
   <expressionL type="string">
    <useValue>stub</useValue>
   </expressionL>
   <relation>isIn</relation>
   <expressionR type="string">
    <useContext>networkType</useContext>
   </expressionR>
  </restriction>
  <restriction type="selectionRestrictionType">
   <scope>tcu</scope>
   <expressionL type="string">
    <useValue>accessPoint</useValue>
   </expressionL>
```

```
  <relation>is</relation>
  <expressionR type="string">
   <useContext>linkType</useContext>
  </expressionR>
 </restriction>
 <restriction type="selectionRestrictionType">
  <scope>tcu</scope>
  <expressionL type="ipAddress">
   <useContext>routerUplinkIf</useContext>
  </expressionL>
  <relation>isNotIn</relation>
  <expressionR type="ipAddress">
   <useParam>routerUplinkIf</useParam>
  </expressionR>
 </restriction>
</subservice>
</service>
```

4.2.2 Adjustments to the General Service Descriptor

Since not every layer does exactly the same, the service descriptors must slightly be adjusted. The general service descriptor is valid for the TCSP and the ISP Layer.
The TCU Layer introduces a new child element for each subservice: device. The element defines if the service will be installed on a router or a TPD.
As shown in Figure 4.1 the Device Layer does no further mapping. Therefore it does not need any subservice elements.

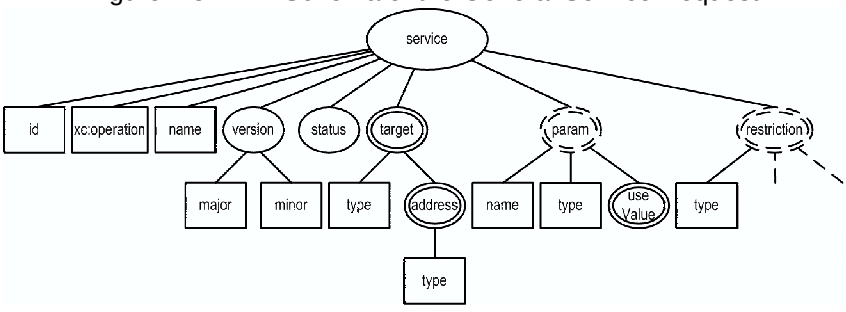
4.3 Service Requests

The first section explains the general service request and the second section deals with the specialities of each layer.

4.3.1 General Service Request

The service requests contain deployment information and the service specification. The service can be defined through the service's name and the two version numbers and further be customized using the different parameters. Figure 4.3 shows the general service request.

Figure 4.3: XML Schema of the General Service Request



The descriptor again consists of different elements that are closely related to the service descriptor. The following list explains their use:

- *Name Attribute And Versioning*
  The attribute name and the version element with its major and minor number are used to identify a service. The name describes the service's base functionality and the numbers give us its current version.

Listing 4.2: Service Request Example

```
<service name="ingressFilter">
 <version major"1" minor="3"/>
 <status>active</status>
 <target type="source">
  <address type="subnet">129.132.12.0/24</address>
  <address type="subnet">129.132.13.0/24</address>
 </target>
 <param name="uplinkRouterIf" type="ipAddress">
  <useValue>114.23.67.14</useValue>
  <useValue>114.23.67.96</useValue>
 </param>
</service>
```

- *Status*
  The status element is used to indicate the status of this service. This could also be done with an attribute, but the element leaves us the possiblity to enlarge its functionality. Right now, the status can only be set on 'active' or 'inactive'.

- *Target*
  The target contains the target address space. These adresses must be owned by the network user. Inside a service request one or two target elements are allowed. One of type 'source' and one of type 'destination'. The element contains one or more address elements. All of them are specified by a type ('single', 'range' or 'subnet') and contain the address (range).

- *Parameter*
  The parameter values are contained inside the param elements. Every parameter is specified by its name and type. Inside a parameter there are one or more useValue elements, which contain the values. This means that we are dealing with sets rather than with single values.

- *Restriction*
  You have already seen those elements when studying the service descriptor. Chapter 5 explains how this deployment information is used.

Listing 4.2 shows an exmaple of a service request that is created by a network user.

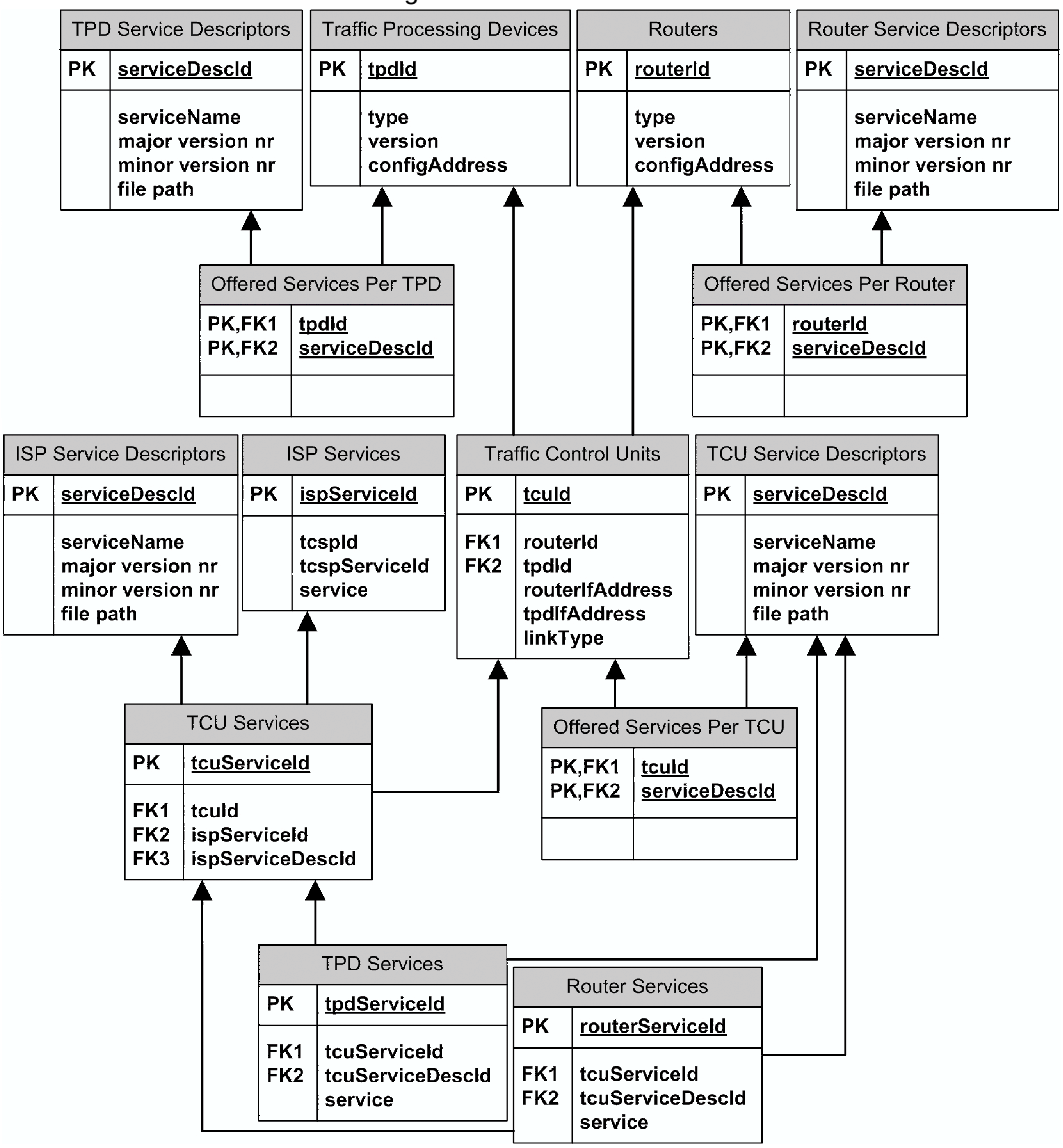
4.3.2 Adjustments to the General Service Request

The network user's service request is not allowed to contain deployment information. Therefore no restriction elements are present in the original service request. This is the only difference. All other service requests look just like presented in Figure 4.3.

4.4 Additional Information

Additional information that has to be provided includes context information from the actual layer and the offered services from the following layer. Chapter 5 explains how the service deployment and hence the processing of this information works.
Next to the context information the installed services must be stored and the service descriptors managed. Figure 4.4 shows a database that can store all this information. How the service itself is stored must still be defined. This is why the service is just referenced as service inside the tables. But the structure must still be serialized.
The proposed database can store the structure of the ISP concerning the Distriubted Traffic Control System. The TCU connects one router interface with one TPD interface. For every defined unit (the whole ISP, the TCUs and the different devices) the offered services must be de-

Figure 4.4: ISP Database



fined and the location of the associated service descriptor specified. Furthermore the currently installed services must be stored.

The database does not yet contain tables to store the context information and the data model of the context information has not been defined yet. The idea is to be able to access the different context information just like information stored in a Management Information Base. This means that the context can be accessed using a predefined path.

Chapter 5

Service Deployment

This chapter discusses how a service is deployed. The information model has already been introduced in Chapter 4. At every layer information from the service request, the service descriptor and from the context is put together to form a new request destinated for the lower layer. Furthermore the service must only be installed at lower layers that are suited. Because every layer does the same processing, we focus on just one layer.
In Section 5.1 the mapping is covered. This includes the creation of the service request. Section 5.2 discusses the selection of the lower layers. Finally a use case is presented: The ingress filtering service.

5.1 Service Mapping

5.1.1 Choosing an Appropriate Service Descriptor

Any incoming service request is identified by the service's name and version numbers. Those properties are used to find an adequate service descriptor that matches the service's name, the major version number and equals or has a bigger minor version number. More than one service descriptor may match an incoming service request, either because there are two service descriptors with different version numbers or there are more than one possibility to map this service onto lower layer services.
For every lower layer instance, an appropriate service descriptor must now be chosen. This depends on the offered services of the lower layer. Some service descriptors might map the requested service onto services that cannot be performed by the lower layer instance. If after this tests more than one service descriptor matches the requested service, the choice is effected using a predefined policy. If no service descriptor is left, the lower layer instance is skipped.

5.1.2 Validate the Service Request

The chosen service descriptor can now be used to validate the requested service. This includes checking if the requested targets and the requested parameters are passed. Furthermore the service descriptor defines which targets and parameters may be passed. Not defined parameters and targets must not be passed. For each parameter the service descriptor may specify a validation test. This may be a range or list of values. The service descriptor may further contain assertations. Those can be used to assert that an expression is inside a specific range or similar.

5.1.3 Creating the Subservice Requests

Since a service may be mapped onto several subservices, the output of the mapping process is a set of service requests. This subsection assumes that only one subservice is created, because the creation of more than one service works similar.
A subservice is identified by its name and version numbers. So these properties are already set since they are contained in the service descriptor. The targets are directly passed from the

service request. Any parameters of the subservice are set using the <param> elements inside the <subservice> element. There are four possibilities to set the parameters values:

- *<useValue>*
  A <param> element may contain one or more <useValue> elements that contain constant values.
- *<useParam>*
  With <useParam> the values are fetched from a declared public parameter. If no values are specified for a certain parameter, the service descriptor may define a set of default values.
- *<useContext>*
  This element is used to fill in values fetched from the context database of this layer.
- *<useFunction>*
  <useFunction> can be used to execute a function call that returns the values that should be stored in the service request. Any function arguments that use again <useContext> refer still to this layer's context database. If the function call fails, the possibly defined default values can be used instead.

Finally the restrictions are added. While restriction passed with the service request are appended to every lower layer service request, the ones specified in the service descriptor are only added to the parent subservice. Any <useParam> and <useFunction> elements are replaced by getting the values from the parameter and executing the function respectively. Note that <useContext> elements are not replaced. Section 5.2 makes clear why.

5.2 Service Deployment

According to the offered services of the lower layer a service descriptor has been chosen and the service request destinated for the lower layer instance been created. We now care if the generated services should be requested from a specific lower layer instance. To do this selection, the restriction have been introduced.

5.2.1 Performing the Extension and Selection Restrictions

Any extension restrictions are processed before other restrictions are processed. Such a restriction does only have to be processed if its scope refers to the lower layer. For instance if the processing is currently in the ISP Layer only restrictions with scope 'TCU' are processed. Any extension restriction contains two sets that will be compared using the specified relation. These sets that are contained in <expressionL> and <expressionR> contain either <useValue> elements or one <useContext> element. With constant values no preprocessing has to be done. For a <useContext> element the context is fetched from the lower layer. If finally the expression is true, additional parameters, restrictions or subservices that are contained inside the <extension> element are added to the service request of this specific lower layer instance. If the expression is false, extensions from the <alternativeExtension> element are added.

For the selection restriction the preprocessing is the same as for the extension restriction. If finally the expression is true the service request is sent to this lower layer instance. Otherwise it is skipped.

5.2.2 Performing Dependency Restrictions

An example for a service that uses a dependency restriction is a trigger service since it should only be installed at ISPs/TCUs where the service that should be activated/deactivated is deployed. To perform a dependency restriction, the services currently installed at a lower layer instance have to be accessible. The interface between the different layers does not yet contain a function that offers this functionality.

Listing 5.1: Ingress Filtering Service: Service Request of the Network User

```
<service name="ingressFilter">
 <version major"1" minor="3"/>
 <status>active</status>
 <target type="source">
  <address type="subnet">129.132.12.0/24</address>
  <address type="subnet">129.132.13.0/24</address>
 </target>
 <param name="uplinkRouterIf" type="ipAddress">
  <useValue>114.23.67.14</useValue>
  <useValue>114.23.67.96</useValue>
 </param>
</service>
```

5.3 Deployment of the Ingress Filtering Service, a Use Case

Ingress filtering is a most effective measure against a reflected DDoS attack. However, care must be taken when installing such a service because it is important that the own traffic is not filtered. The following restrictions must be met:

- Install the service at ISPs that maintain a stub network
- Filter only packets that enter a router through an access point, which is where the customers are connected
- Do not filter the own uplinks

The network user sends a request to the TCSP asking him to perform ingress filtering (see Listing 5.1. With the request, the target address space is specified. This defines that IP addresses that will be filtered.The network user further passes its own uplinks inside a parameter.
After some processing the TCSP finds a service descriptor that matches the service request. This service descriptor is presented in Listing 5.2. The chosen service descriptor requires a target of type source and does not allow any destination targets. Futhermore it requires the parameter uplinkRouterIf. The service is mapped onto a lower layer service called ispFilter. (The use of isp inside the service's name does not mean a thing. It is just used to guide you through the different layers.) The subservice introduces three restrictions and one parameter. The first restriction is performed at the TCSP and is used to skip ISPs that do not maintain a stub network. The parameter specifies that only IP packets coming in through the router interface should be processed, not outgoing ones.

Listing 5.2: Ingress Filtering Service: Service Descriptor of the TCSP

```
<service name="ingressFilter">
 <annotation>This service performs ingress filtering.</annotation>
 <version major"1" minor="3"/>
 <target type="source" use="required"/>
 <publicParam name="uplinkRouterIf" type="ipAddress" use="required">
  <annotation>Pass the the IP-addresses of your router uplink
           interfaces here.</annotation>
 </publicParam>
 <subservice name="ispFilter">
  <version major="1" minor="0"/>
  <restriction type="selectionRestrictionType">
   <scope>isp</scope>
   <expressionL type="string">
    <useValue>stub</useValue>
   </expressionL>
   <relation>isIn</relation>
   <expressionR type="string">
```

```
   <useContext>networkType</useContext>
  </expressionR>
 </restriction>
 <restriction type="selectionRestrictionType">
  <scope>tcu</scope>
  <expressionL type="string">
   <useValue>accessPoint</useValue>
  </expressionL>
  <relation>is</relation>
  <expressionR type="string">
   <useContext>linkType</useContext>
  </expressionR>
 </restriction>
 <restriction type="selectionRestrictionType">
  <scope>tcu</scope>
  <expressionL type="IPAddress">
   <useContext>routerIfAddress</useContext>
  </expressionL>
  <relation>isNotIn</relation>
  <expressionR type="ipAddress">
   <useParam>routerUplinkIf</useParam>
  </expressionR>
 </restriction>
 <param name="trafficType" type="string">
  <useValue>incoming</useValue>
 </param>
</subservice>
</service>
```

After the service mapping and the selection of the ISPs, where by the way the restriction has been removed, the service request looks like presented in Listing 5.3. This request is now sent to the ISPs.
At the ISP the same processing is done again. Listing 5.4 shows an appropriate service descriptor. The two remaining restrictions are both processed at this layer. While the first one ensures that a customer network is connected to this router interface, the second one prohibits the installation of the filter at the own uplinks. The service is mapped onto the tcuFilter service. Any values passed within the parameter filterRule or trafficType is passed on to the lower layer service request.
Listing 5.5 shows the service request that is sent to the TCU Layer. Since no filterRule has been specified, the parameter has been dropped and does not appear in the service request.
Listing 5.6 shows the service descriptor of the TCU Layer. At this layer, the service is mapped onto two services, a redirect service that as specified by the <device> element runs on the router and a tpdFilter that runs on the TPD. The filter rule will be passed on to the TPD, the traffic type parameter to the redirect service. To redirect the IP packets to the right TPD, the service fetches the information from the context.
Instead of one service request we are now dealing with two service requests, one sent to the router and one sent to the TPD. Listing 5.7 and Listing 5.8 show these two requests.
At the Device Layer the service request is again validated but no more mapping is done. Therefore no subservice elements are specified within the descriptor. Listing 5.9 and Listing 5.10 show the service descriptor of the router service and of the TPD service respectively.

Listing 5.3: Ingress Filtering Service: Service Request of the TCSP

```
<service name="ispFilter">
 <version major"1" minor="0"/>
 <status>active</status>
 <target type="source">
  <address type="subnet">129.132.12.0/24</address>
  <address type="subnet">129.132.13.0/24</address>
 </target>
 <param name="trafficType" type="string">
  <useValue>incoming</useValue>
 </param>
 <restriction type="selectionRestrictionType">
  <scope>tcu</scope>
  <expressionL type="string">
   <useValue>accessPoint</useValue>
  </expressionL>
  <relation>is</relation>
  <expressionR type="string">
   <useContext>linkType</useContext>
  </expressionR>
 </restriction>
 <restriction type="selectionRestrictionType">
  <scope>tcu</scope>
  <expressionL type="ipAddress">
   <useContext>routerUplinkIf</useContext>
  </expressionL>
  <relation>isNotIn</relation>
  <expressionR type="ipAddress">
   <useValue>114.23.67.14</useValue>
   <useValue>114.23.67.96</useValue>
  </expressionR>
 </restriction>
</service>
```

Listing 5.4: Ingress Filtering Service: Service Descriptor of the ISP Layer

```
<service name="ispFilter">
 <annotation>This service performs filtering.</annotation>
 <version major"1" minor="2"/>
 <target type="source" use="required"/>
 <target type="dest" use="optional"/>
 <publicParam name="filterRule" type="string" use="optional">
  <annotation>Specify your filter rule. If none is passed all
                                    packets are blocked.</annotation>
 </publicParam>
 <publicParam name="trafficType" type="string" use="optional"/>
 <subservice name="tcuFilter">
  <version major="1" minor="0"/>
  <param name="filterRule" type="string">
   <useParam>filterRule</useParam>
  </param>
  <param name="trafficType" type="string">
   <useParam>trafficType</useParam>
  </param>
 </subservice>
</service>
```

Listing 5.5: Ingress Filtering Service: Service Request of the ISP Layer

```
<service name="tcuFilter">
 <version major"1" minor="0"/>
 <status>active</status>
 <target type="source">
  <address type="subnet">129.132.12.0/24</address>
  <address type="subnet">129.132.13.0/24</address>
 </target>
 <param name="trafficType" type="string">
  <useValue>incoming</useValue>
 </param>
</service>
```

Listing 5.6: Ingress Filtering Service: Service Descriptor of the TCU Layer

```
<service name="tcuFilter">
 <annotation>This service performs filtering.</annotation>
 <version major"1" minor="0"/>
 <target type="source" use="required"/>
 <target type="dest"’ use="optional"/>
 <publicParam name="filterRule" type="string" use="optional">
  <annotation>Specify your filter rule. If none is passed all
                                    packets are blocked.</annotation>
 </publicParam>
 <publicParam name="trafficType" type="string" use="optional"/>
 <subservice name="routerRedirect">
  <version major="1" minor="0"/>
  <param name="trafficType" type="string">
   <useValue>incoming</uuseValue>
  </param>
  <param name="destination" type="IPAddress">
   <useContext>tpdIfAddress</useContext>
  </param>
  <device>router</device>
 </subservice>
 <subservice name="tpdFilter">
  <version major="1" minor="0"/>
  <param name="filterRule" type="string">
   <useParam>filterRule</useParam>
  </param>
  <device>tpd</device>
 </subservice>
</service>
```

Listing 5.7: Ingress Filtering Service: Service Request of the TCU Layer for the Router

```
<service name="routerRedirect">
 <version major"1" minor="0"/>
 <status>active</status>
 <target type="source">
  <address type="subnet">129.132.12.0/24</address>
  <address type="subnet">129.132.13.0/24</address>
 </target>
 <param name="trafficType" type="string">
  <useValue>incoming</useValue>
 </param>
 <param name="destination" type="IPAddress">
  <useValue>114.23.67.38</useValue>
 </param>
</service>
```

Listing 5.8: Ingress Filtering Service: Service Request of the TCU Layer for the TPD

```
<service name="tpdFilter">
 <version major"1" minor="0"/>
 <status>active</status>
 <target type="source">
  <address type="subnet">129.132.12.0/24</address>
  <address type="subnet">129.132.13.0/24</address>
 </target>
</service>
```

Listing 5.9: Router Service Descriptor of the Device Layer

```
<service name="routerRedirect">
 <annotation>This service does the redirecting.</annotation>
 <version major"1" minor="0"/>
 <target type="source" use="required"/>
 <target type="dest" use="optional"/>
 <publicParam name="trafficType" type="string" use="optional"/>
 <publicParam name="destination" type="IPAddress" use="required">
  <annotation>The packets are redirected to this address.</annotation>
 </publicParam>
</service>
```

Listing 5.10: TPD Service Descriptor of the Device Layer

```
<service name="tpdFilter">
 <annotation>This service filters the packets.</annotation>
 <version major"1" minor="0"/>
 <target type="source" use="required"/>
 <target type="dest" use="optional"/>
 <publicParam name="filterRule" type="string" use="optional">
  <annotation>Specify your filter rule. If none is passed all
                                    packets are blocked.</annotation>
 </publicParam>
</service>
```

Chapter 6

The Click Router as Traffic Processing Device

The Traffic Processing Device can be realized with different technologies. Compared to an implementation done with a FPGA, the Click Router implementation is easier to realize and it is therefore more suited for a prototype. Of course it is slower than other solutions. This chapter gives a short introduction to the Click Router, outlines the data model and the protocol used to access the TPD and finally shows the TPD's architecture.

6.1 The Click Modular Router

The Click Modular Router [2] runs in the kernel space of a Unix system. The router is put together from small elements, that perform simple operations.

Figure 6.1: Simple Click Configuration

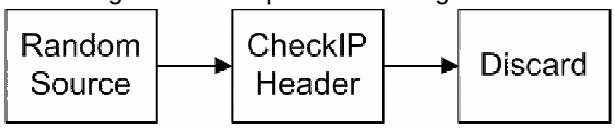


Figure 6.1 shows a simple configuration of a Click Router. Packets are generated at the RandomSource, then checked in the CheckIPHeader element and finally discarded. Most elements allow more than one input or output. With all the currently available elements complex services can be generated.

To load a click configuration the shell command 'click-install <config-file>' can be used. The configuration is written in a defined language which is simple to understand. Listing 6.1 shows the same configuration as Figure 6.1 using the click configuration language. The first part defines the three elements whereas the second part connects them.

Listing 6.1: Click Configuration Language

```
randomSource::RandomSource();
checkIPHeader::CheckIPHeader();
discard::Discard();

randomSource[0]->[0]checkIPHeader;
checkIPHeader[0]->[0]discard;
```

6.2 Protocol and Data Model

The TPD can be configured using the NetConf Protocol. As the processing at the ISP reaches its lowest layer, the device layer, the Plugin is loaded. They are used to transform the data model used at the ISP into the data model of the device and second, to assure the communication between ISP and device.
Using the NetConf Protocol allows us to use the same data model as the ISP and hence reduces the data model eliminates the data model translation. So the Plugin will become very simple. The communication is not very difficult to achieve too, as a NetConf client is already available.
However, the data model needs some minor modification. As we would like the TPD to be able to contain more than one interface, every request must specify to which interface it belongs. A 'interface' attribute inside the service element of the service request will do the job.

6.3 Architecture of the Click TPD

This section defines how the click elements have to be put together to form a dynamic solution that is able to process IP packets. The following three rules apply:

- Redirect non IP packets to the host. Let him deal with this.
- Redirect IP packets destinated for this host to the host.
- Send all other packets through the processing.

After the processing has been done, the data is sent back to where it came from, this means to send it out through the same interface that it arrived and destinated to the router it originated from. Inversing the two addresses of the ethernet packet will do the job. Figure 6.2 shows the click configuration that achieves this. Keep in mind that such a configuration is needed for each interface!

Figure 6.2: Click Configuration of the TPD

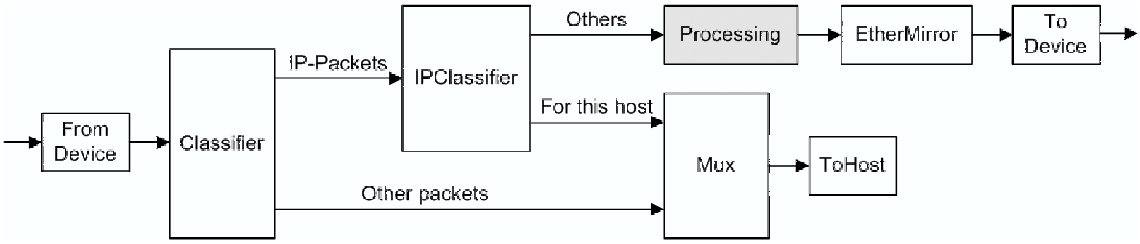
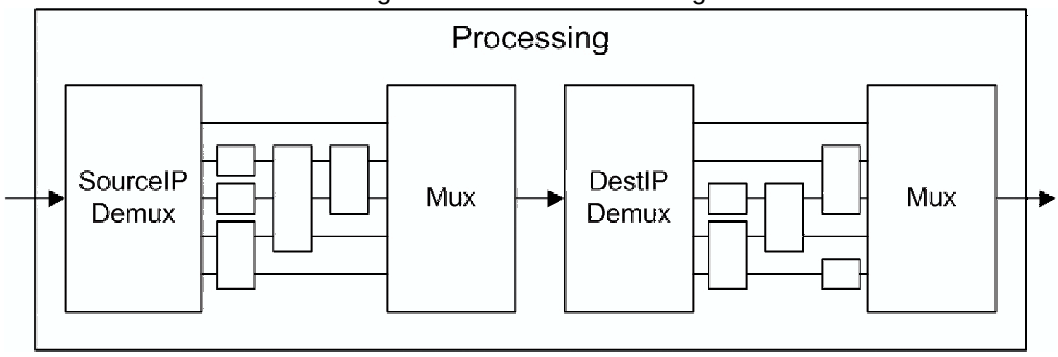


Figure 1.3 showed how packets can be processed. Because the IP packets belong to the sender before they belong to the receiver, we have to process all operations on the source IP addresses before we process the operations on the destination IP addresses. Figure 6.3 shows how the packets are processed. Whereas the little squares mean different services that are processed on the packets.
The two processing stages (source and destination) work exactly similar, so a look at the first stage will suffice. We define processing chains that are identified by an IP address range. Such a processing chain connects the demux with the mux element passing several services on its way. As its name says, the demux is responsible for the demultiplexing of the packets. Every incoming packet is therefore sent out through the assigned outport or if none is specified through port 0, which is always directly connected with the mux element. On its way from the demux to the mux, the packet may pass several service elements.
A service element may consist of more than one click element. The service element is an additional abstraction in order to simplify the connections between the different service elements. A click element is allowed to have zero in- or outputs, while our service element is not. A service element is defined by the following two rules:

- It has many inports as outports

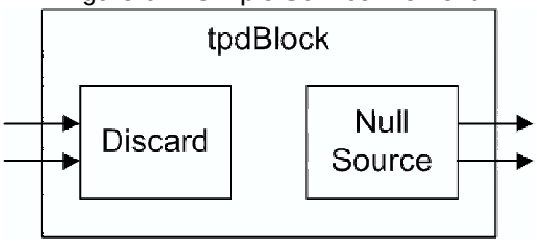
Figure 6.3: Packet Processing



- Packets arriving at inport n are either sent to outport n or to none at all

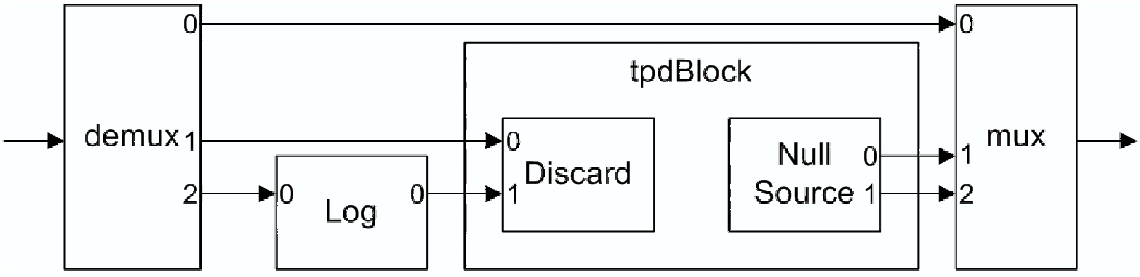
Let us first take a look at the inner working of such a service element before we want to see how they are connected one another. Figure 6.4 shows a simple service element that consists of two click elements. The Discard element discards any packet arriving, no matter what inport. Of course it does not make sense if such an element would have any outports. To satisfy the definition of a service element, another click element that provides some outports, the NullSource element, has to be inserted. Incoming connections of this service element are merged with the Discard element whereas outgoing connections are merged with the NullSource element. Of course a service element may consist of more than two click elements and be wired in a complex way.

Figure 6.4: Simple Service Element



A processing chain passes several service elements and every packet passing a certain processing chain is of the same address space. If a packet is sent through outport 3 of the demux, it may pass a service element. We do not care on which inport the packet will enter the service element but since we know that it will be sent out through the same port number we are able to connect the different service elements one another and to the demux and mux. Figure 6.5 gives a simple example.

Figure 6.5: Simple Processing Example



Listing 6.2: Example of a Click Configuration

```
eth0_fromDevice::FromDevice(eth0);
eth0_toDevice::ToDevice(eth0);
eth0_classifier::Classifier(12/0800, -);
eth0_ipClassifier::IPClassifier(dst host /192.14.14.1, -);
eth0_toHost::ToHost(eth0);
eth0_mux::Mux();
eth0_etherMirror::EtherMirror();
eth0_sourceDemux::sourceDemux(129.30.24.16/16 );
eth0_sourceMux::sourceMux();
eth0_destDemux::destDemux();
eth0_destMux::destMux();
eth0_discard_1::Discard();
eth0_nullSource_1::NullSource();

eth0_fromDevice[0]->[0]eth0_classifier;
eth0_classifier[0]->[0]eth0_ipClassifier;
eth0_classifier[1]->[0]eth0_mux;
eth0_ipClassifier[0]->[1]eth0_mux;
eth0_mux[0]->[0]eth0_toHost;
eth0_ipClassifier[1]->[0]eth0_sourceDemux;
eth0_sourceDemux[0]->[0]eth0_sourceMux;
eth0_sourceDemux[1]->[0]eth0_discard_1;
eth0_nullSource_1[0]->[1]eth0_sourceMux;
eth0_sourceMux[0]->[0]eth0_destDemux;
eth0_destDemux[0]->[0]eth0_destMux;
eth0_destMux[0]->[0]eth0_etherMirror;
eth0_etherMirror[0]->[0]eth0_toDevice;
```

6.4 Service Request Processing

As at the ISP a service request is received through the NetConf Agent and passed to the processor, in this case the ClickProcessor. If a service is requested, the class that represents this service is loaded. This class is what we former defined as service element. After the class has been loaded and a new instance has been generated, we initialize the object with the service specific values, these are the serviceID and the parameters.
If a service should be deleted, we just delete the affected service elements. If it should be edited, the parameters can be set to new values.
In any case, after all modifications have been done, the service elements are asked to pass their click configuration element definitions and in a second step to pass their click configuration connections. Once this information is collected, the new click configuration can be generated and finally loaded. Listing 6.2 shows a sample click configuration.

Chapter 7

Conclusion and Future Work

7.1 Summary

7.1.1 The System

The system designed by [1] introduces the different components of the Traffic Control Service. The thesis defines the ISP part of the system, including the protocols used by the TCSP and ISP to communicate with the ISP and the TPD respectively. With the Traffic Control Unit another subcomponent has been introduced that proves itself very useful. The chosen design meets the deployment requirements of any possible service.
The NetConf Protocol proves itself as being very flexible allowing to the define our own data model and if necessary to extend it. Because it is still in development, things may change. Since no useful implementation of NetConf exists a simple NetConf Deamon and a NetConf client have been programmed in Java. They use another simple implementation of SOAP. Since SOAP itself runs over HTTP, the communication is not secure. The design has been chosen to allow the implementation of TLS/SSL though. The NetConf infrastructure contains 3068 lines of code (comment included).
With the implementation of the processing at the ISP, the TCSP may now send service requests to the ISP, waiting for it to process the message. In any case a message will be returned telling the TCSP about the outcome of the request. The processing runs stable and logs any errors.

7.1.2 XML Schemas

Along with this documentation two XML schemas are provided. One describes the service descriptors. With this schema, new service descriptors may now be written offering new services. The descriptor design has been proven sufficient by this work.
The second XML schema defines the data model associated with the ISP and the TPD. Because the NetConf Protocol is used, the service requests contain the data model as well. It primarily defines how the service requests look like.
To parse XML documents assigned to the two schemas, a data structure is implemented, that marshals and unmarshals data from XML to Java and back to XML respectively. These classes could not be generated automatically because the used programs to do so, failed when it came to extended types defined by the XML schema. These Java classes contain 3129 lines of code (comment included).

7.1.3 ISP Processing

The implemented processing accepts service requests and deploys them according to the available service descriptors. The current configuration is always accessible through a dummy database which is implemented by a Java class.
The lowest layer at the ISP includes the Plugins. Currently one Plugin is provided used to access the TPD. The Plugin management has therefore not posed any problems. The processing has been programmed in Java and contains 2062 lines of code (comment included).

7.1.4 The Click Router as Traffic Processing Device

With the newly created elements, the click router is able to act as a TPD. It is a powerful device allowing us to simply add new services by writing an appropriate service element class.
The additional click elements are the sourceDemux and the destDemux as well as the Mux. Those three elements have been developed and are also provided with this work.
There is one service element currently available: The tpdBlock service. It is simply put together from the discard element and a nullSource element.
To act as a TPD the three new click elements have been programmed in C++. Furthermore the management logic has been implemented in Java and contains 898 lines of code (comment included).

7.1.5 Service Deployment

The service deployment has also been specified by this thesis. It worked for all tried services, especially for the ingress filtering and trigger examples. The use of restriction allows complex services to be installed. Furthermore information at lower layers can be referenced using the different possibilities to get a value: <useContext>, <useParam> and <useFunction>.
To guarantee an effective deployment, the offered services from the lower layer should be available. Moreover context information is needed from the next layer as well. This is no problem inside the ISP management. The TCSP may gather the information from the ISP by using the <get-config> method from time to time.

7.2 Conclusion

With the use of the NetConf Protocol a flexible configuration protocol has been chosen. The data model can simply be adjusted if required. Minor changes caused by the developing process can easily be integrated.
The service deployment effected by the use of service descriptors proves itself very flexible. We have not yet met any service that does not fit into this structure.
With the specified and implemented service deployment and mapping process and with the design and implementation of the simple TPD a first step is done creating a test environment.

7.3 Future Work

7.3.1 Enhance the Mapping Process at the ISP

The mapping process used at the ISP can be accelerated. Right now for every TCU the service descriptor is loaded. This can be eliminated.

7.3.2 Router Plugin and Test Environment

Right now, no router plugin is available. In a next step a router should be chosen that forms a test environment with the TPD. A new Plugin must then be written for that router. The test environment can then be used to perform some significant tests.

7.3.3 Database Access

The data at the ISP is currently stored in a dummy database. So, the next thing would be to implement the database and program a class that accesses it. This class should provide the interface ’Database’, so that it could be used by the current ISPProcessor. The database stores all current configuration information for every layer as well as the associated context data and offered services. Furthermore it tracks the service descriptors and knows how the different devices are connected one another.

7.3.4 Examine the Performance Loss at a Router

When a service is installed and a router has to redirect packets through a TPD the performance of the router is affected. Imagine a router performing some filtering for a specific address. If a second service is requested also acting on the same address space, that is split and distributed over the router and TPD, the performance loss would be less if the router would just perform the redirect service and the TPD would perform the filtering and the second service. Therefore the performance loss of the router should be examined.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Winter 2004/2005

Master Thesis

für

Christoph Jossi

Supervisor: Matthias Bossardt
Co-Supervisor: Thomas Dübendorfer

Ausgabe: 11.10.2004
Abgabe: 15.4.2005

Management of Distributed Traffic Control Service

---

1 Introduction

Frequency and intensity of Internet attacks are rising with an alarming pace. Several technologies and concepts were proposed for fighting distributed denial of service (DDoS) attacks: traceback, pushback, i3, SOS and Mayday. In the case of DDoS reflector attacks they are either ineffective or even counterproductive. We proposed a novel concept and system that extends the control over network traffic by network users to the Internet using adaptive traffic processing devices [3]. We safely delegate partial network management capabilities from network operators to network users. All network packets with a source or destination address owned by a network user can now also be controlled within the Internet instead of only at the network user's Internet uplink. By limiting the traffic control features and by restricting the realm of control to the "owner" of the traffic, we can rule out misuse of this system. Applications of our system are manifold: prevention of source address spoofing, DDoS attack mitigation, distributed firewall-like filtering, new ways of collecting traffic statistics, traceback, distributed network debugging, support for forensic analyses and many more.

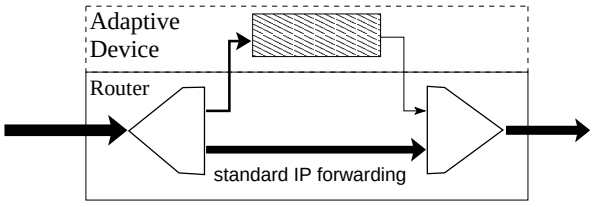


Abbildung 1: Router extension with adaptive device

Our system consists of remotely programmable network traffic processing devices as shown in Figure 1. The owner of a network address or range gets access to the management of some or all of these devices after having registered for the distributed traffic control service. Traffic entering a router is redirected to a nearby adaptive device only if it carries an IP address as source or destination, which the adaptive device was setup for. Most traffic will use the direct path through the router.

When the adaptive device processes a network packet, it first executes traffic control on behalf of the owner of the source IP address. Subsequently, it executes traffic control on behalf of the owner of the IP destination address. This is analogous to the high-level communication process of first sending an Internet packet by the source (and hence under its control) and then receiving it by the destination (and consequently under the recipient's control). This control hand-over is performed at each activated adaptive device on the network path of an IP packet.

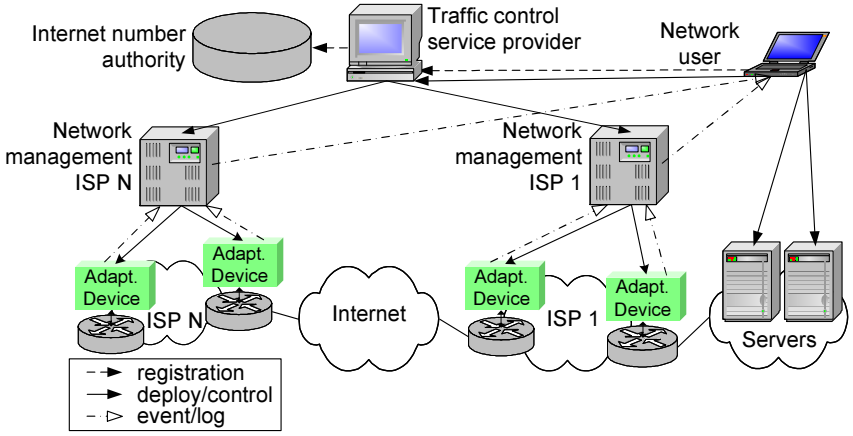
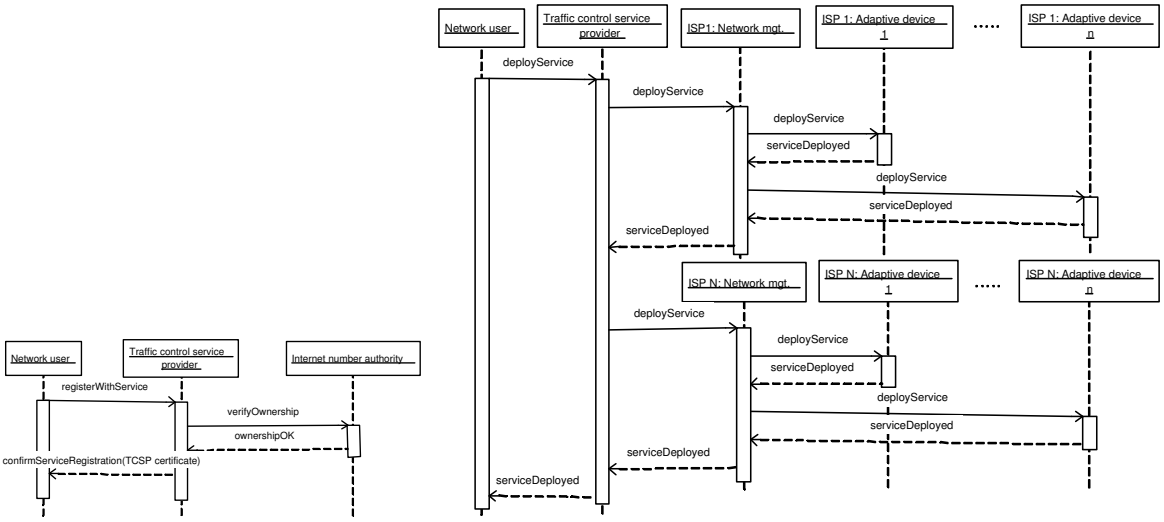


Abbildung 2: Network model

Our network model shown in Figure 2 distinguishes four different roles: *Internet number authority, Traffic control service provider (TCSP), Internet service provider (ISP)*[1], and *Network user*.

The TCSP manages the new traffic control (TC) service. It sets up contracts with many ISPs that subsequently attach adaptive devices to some or all of their routers and enable their network management system to program and configure these adaptive devices.



(a) Service registration

(b) Service deployment

Abbildung 3: Message sequence diagrams

A network user must first register with the TCSP before using the traffic control service (Figure 3(a)). The TCSP checks the identity of the network user[2] and verifies the claimed ownership of IP addresses, which she wants to control traffic for. Therefore, the TCSP checks with Internet number authorities[3] if the IP addresses are indeed owned by the service requester. If everything is ok, access to the traffic control service is granted. The

[1]This section subsumes both type of organisations, ISPs and BSPs, under the role ISP.

[2]To check the network user's identity the TCSP performs similar actions as a digital certification authority (CA), e.g. offline verification of an official identity card or online verification of a digital certificate issued by a trusted CA.

[3]Ownership of (ranges of) IP addresses is maintained in databases of organisations such as ARIN, RIPE NCC, etc.

binding of a network user to the set of IP addresses owned and the subsequent verification when using the traffic control service (TC service) could be implemented with digital certificates signed by the TCSP.

After successfully registering to the basic TC service, a network user may initiate the deployment of a specific service (e.g. ingress filtering), which is implemented on top of the TC service (Figure 3(b)). The network user requests the TCSP to deploy the specific service in the network. The network user may scope the deployment according to different criteria (e.g. “only on border routers of stub networks”). The TCSP maps the request to service components and instructs network management systems of appropriate ISP’s to deploy and configure the service components. ISPs in turn deploy and configure the components on adequate adaptive devices and configure their routers accordingly. Once the service is deployed, a network user may activate, modify specific parameters or read logs of the service. Therefore it sends corresponding requests to the TCSP, which relays them to the appropriate ISP’s network management systems.

2 Assignment

2.1 Objectives

The following objectives are expected to be met by this thesis.

- A system design that allows to map service requests from the TCSP layer to configurations of router and AD.
- XML service descriptors that allow to specify service requirements in a machine readable form on all involved layers (TCSP, ISP NMS, Device). The student may take service descriptors from [1] as a starting point.
- A protocol specification and implementation between ISP NMS and Router/AD. If possible the protocol should be based on (proposed) standards, e.g. SNMP [2] and NetConf [4].
- A management protocol specification between TCSP and ISP NMS.
- A data model for static and dynamic context information at TCSP, ISP NMS and Device (Router/AD) layer.

2.2 Tasks

- Get familiar with the distributed traffic control service and the architecture of the management system as described in [3].
- Describe the deployment of the ”ingress filtering” service in detail.
- Generalise the ”ingress filtering” scenario to one that allows to set triggers on the router/AD and to take specific actions if the trigger condition is met.
- Get familiar with the concept of service descriptors as described in [1].
- Define the information model of the complete system with respect to service deployment. That is, for each layer describe the information available at that layer and the information required from other layers in order to carry out the mapping process.
- Define the format of a service descriptor that allows to specify deployment requirements of the services from the scenarios above.
- Specify the management messages between ISP NMS and router/AD and the associated data model.
- Specify the management messages between TCSP and ISP NMS and the associated data model.
- Implement the protocol between ISP NMS and router/AD.
- Implement a demonstration of the ”ingress filtering” service deployment using a Click router [5] based AD.
- Document your work in a detailed and comprehensive way. We suggest you to continually update your documentation. New concepts and investigated variants must be described. Decisions for a particular variant must be justified.

3 Deliverables and Organisation

- If possible, student and advisors meet on a weekly basis to discuss progress of work and next steps. If problems/questions arise that can not be solved independently, the student may contact the advisor anytime.
- At the end of the third week, a detailed time schedule of the Master thesis must be given and discussed with the advisor.
- At half time of the master thesis, a short discussion of 15 minutes with the professor and the advisor will take place. The student has to talk about the major aspects of the ongoing work. At this point, the student should already have a preliminary version of the table of contents of the final report. This preliminary version should be brought along to the short discussion.
- At the end of the diploma thesis, a presentation of 15 minutes must be given during the TIK or the communication systems group meeting. It should give an overview as well as the most important details of the work. Furthermore, it should include a small demo of the project.
- The final report may be written in English or German (English preferred). It must contain a summary written, the assignment and the time schedule. Its structure should include an introduction, analysis of related work, and a complete documentation of the developed software. Related work must be correctly referenced. See http://www.tik.ee.ethz.ch/.ury/tips.html for more tips. Three copies of the final report must be delivered to TIK.
- Documentation, presentations and software must be delivered on a CDROM.

Literatur

[1] Matthias Bossardt, Roman Hoog Antink, Andreas Moser, and Bernhard Plattner. Chameleon: Realizing automatic service composition for extensible active routers. In *Proceedings of the Fifth International Working Conference on Active Networks, IWAN 2003*, number 2982 in Lecture Notes in Computer Science, Kyoto, Japan, December 2003. Springer Verlag.

[2] J. Case, M. Fedor, M. Schoffstall, and J. Davin. RFC 1157: Simple network management protocol (snmp), May 1990.

[3] Thomas Dübendorfer, Matthias Bossardt, and Bernhard Plattner. Adaptive distributed traffic control service for DDoS attack mitigation. In *Proceedings of the 1st International Workshop on Systems and Network Security, in conjunction with the 19th International Parallel & Distributed Processing Symposium (IPDPS)*, Denver, USA, April 2005. IEEE Computer Society Press.

[4] R. Enns and Editor. Netconf configuration protocol. Internet-Draft: draft-ietf-netconf-prot-04, October 2004.

[5] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

Zürich, den 11.10.2004

Acknowledgement

I would like to thank my two tutors Matthias Bossardt and Thomas Dübendorfer for their excellent support. Working with Martin May, Daniela Brauckhoff and the two tutors has been a real pleasure. Thanks go to Lukas Ruf as well for helping me through some Latex troubles.

Bibliography

[1] Thomas Dübendorfer, Matthias Bossardt, and Bernhard Plattner. Adaptive distributed traffic control service for DDoS attack mitigation. In *Proceedings of the 1st International Workshop on Systems and Network Security, in conjunction with the 19th International Parallel & Distributed Processing Symposium (IPDPS)*, Denver, USA, April 2005. IEEE Computer Society Press.

[2] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263-297, August 2000. *http://www.pdos.lcs.mit.edu/click/*

[3] R. Enns and Editor. Netconf configuration protocol.
Internet-Draft: draft-ietf-netconf-prot-05, February 2005.
http://www.ietf.org/html.charters/netconf-charter.html

[4] Matthias Bossardt, Roman Hoog Antink, Andreas Moser, and Bernhard Plattner. Chameleon: Realizing automatic service composition for extensible active routers. In *Proceedings of the Fifth International Working Conference on Active Networks, IWAN 2003*, number 2982 in Lecture Notes in Computer Science, Kyoto, Japan, December 2003. Springer Verlag.

[5] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent and W. Timothy Strayer: Single-Packet IP Traceback

[6] Ratul Mahanjan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson and Scott Shanker: Controlling High Bandwidth Aggregates in the Network

[7] Katerina Argyraki, David R. Cheriton: Active Internet Traffic Filtering - Real-time Response to Denial-of-Service-Attacks

[8] David G. Anderson: Mayday, Distributed Filtering for Internet Services