MASTER'S THESIS

THE EFFECTS ON SYSTEM SECURITY AND PROVISIONING PROCESS FLOWS CAUSED BY A DYNAMIC TRIVIAL FILE TRANSFER PROTOCOL SERVICE IN LARGE SERVICE PROVIDER NETWORKS

Executed at

Studiengang

Informationstechnologien und Wirtschaftsinformatik

By: Heimo Pleschiutschnig

Student ID: 1510320016

Graz, December 8th 2016 .....

Signature

STATEMENT BY THE AUTHOR

I hereby declare that this submission is my own work and to the best of my knowledge, it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma at any educational institution, except where due acknowledgement is made in the thesis.

.....

Signature

ACKNOWLEDGEMENT

The author would like to thank the following people for their contributions to this thesis. DI Markus Petelinc, BSc (Campus02, academic advisor), Lorenz Geiswinkler (UPC AT, configuration file templates), Matthias Bleile (Unitymedia, configuration file templates), Bernd Predota (LGI, thoughts on Python code) and Tomasz Kubinski (LGI, CMTS configuration details).

ABSTRACT

The purpose of this thesis was to create a working prototype of a dynamic TFTP service for DOCSIS-compliant cable modem configuration files that can serve as a drop-in replacement for existing TFTP services while considerably improving the service's security and reducing the administrative effort required to create such configuration files. The goal was also to ensure that service does not impact legitimate clients and requires only minimal changes to the provisioning service flows and DHCP services.

The paper first explains the components involved and examines an existing reference network and its inherent protocol-related weaknesses. The third chapter then focuses on creating a service blueprint for the dynamic TFTP service using the requirements derived from the examination, with the goal of remedying the detected weaknesses, as well as any weakness dictated by the reference network and those mandated by the ISO 27000. This chapter also defines the KPI and the scientific hypothesis.

The next chapter deals with the design of the prototype (e.g. which language to use, how the prototype's code should work internally).

Chapters five and six explain the code development, the test environment and the gathering of performance data to show how well the prototype performs when compared with the legacy solution.

After gathering all the data, it was necessary to compare this to the previously defined requirements and KPI, as well as to the security clauses of the ISO 27000, which is done in chapter seven.

The final chapter sums up the results from the previous chapters and presents an outlook for possible new functionality and how the service will most likely enter production service.

The conclusion of this thesis is that it is indeed possible to create a dynamic TFTP service that fulfills all the requirements and reaches the necessary KPI values.

TABLE OF CONTENTS

1 INTRODUCTION

"Every revolutionary idea seems to evoke three stages of reaction. They may be summed up by the phrases: 1- It's completely impossible. 2- It's possible, but it's not worth doing. 3- I said it was a good idea all along."

-Arthur C. Clarke

To the uninitiated what we call the Internet, appears as a monolithic, homogeneous network that provides communication, information and entertainment, any time, any place, provided you're in possession of a networked device and the proper connectivity. One touch of a button and it just works.

In reality, the Internet is a heterogeneous conglomerate of billions of independent network nodes that interact with each other using a multitude of transport technologies, protocols and services. There are several standards and specifications that describe the depth and complexity involved to transfer data from one endpoint to another.

Two of these, the International Telecommunication Union's X.200 recommendation (ITU, 1994), also known as the Open Systems Interconnection 7-layer model and the Department of Defense's 4-layer Internet Architecture Model (Cerf, Cain, 1983) will be referred to throughout this paper. A comparison of the two models can be seen in Figure 1.

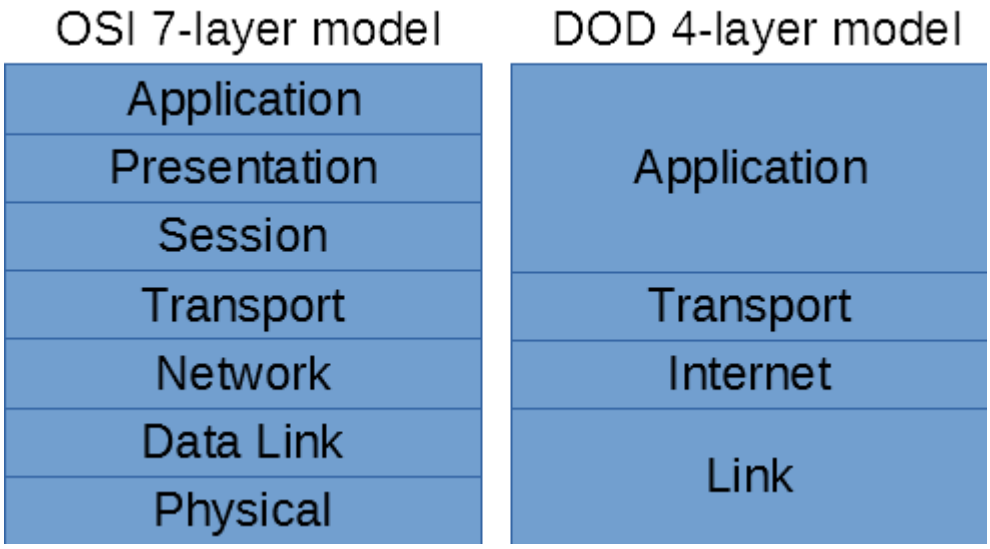

Figure 1 - OSI and DOD model overview

While the predominant Internet Protocol (IP, layer 3 OSI, layer 2 DOD), as specified in RFC 791 for IPv4 (Internet Engineering Task Force, 1981) and RFC 2460 for IPv6 (IETF, 1998) allows

clients to “see” each other in the Internet is used by all clients, the underlying physical connectivity, which is completely transparent to the IP protocol, differs by client.

As stated by Pleschiutschnig in 2015:

“The majority of non-wireless, high-bitrate, consumer internet access today is provided by either digital subscriber line (DSL) or data-over-cable service interface specifications (DOCSIS) transport technology. Similar to previous internet access technologies, like plain old telephone system (POTS) and integrated services data network (ISDN), DOCSIS and DSL require the use of a modulator-demodulator (modem) setup to send-receive data packets over their respective carrier medium.

These setups typically consist of a cable- or DSL-modem on the customer side and a cable modem termination system (CMTS), for DOCSIS or a digital subscriber line access multiplexer (DSLAM), for DSL on the provider side, depending on the type of transport network used as specified by the International Telecommunication Union (ITU, 2000) for DSL and the American National Standards Institute/Society of Cable Telecommunications Engineers (ANSI/SCTE, 2013) for DOCSIS.”

Due to the difference in specifications of DSL and DOCSIS modems and the fact that DSL devices do not rely on the Trivial File Transfer Protocol (TFTP), only DOCSIS modems also referred to as cable modems (CM) are in scope for this paper.

1.1 Background

Every Internet Service Provider (ISP) is interested in having an efficient, well performing and secure network with preferably millions of customers which pay on time. To facilitate that, providing a cable modem and a hybrid fibre cable (HFC) connection to a customer is usually not enough, as the devices also need to be properly provisioned to function and the devices mapped to specific customers so they can be billed.

1.1.1 What is service provisioning?

Simply put, service provisioning is the act of assigning a specific product template to a specific device that belongs to a specific customer. Needless to say, service provisioning plays a large role in the day-to-day business of ISP’s.

The provisioning system logically sits between the customer relationship management (CRM) system, the warehouse management system, the product management and the foundation services servers. A simplified version of the provisioning systems interaction with other components can be seen in Figure 2.

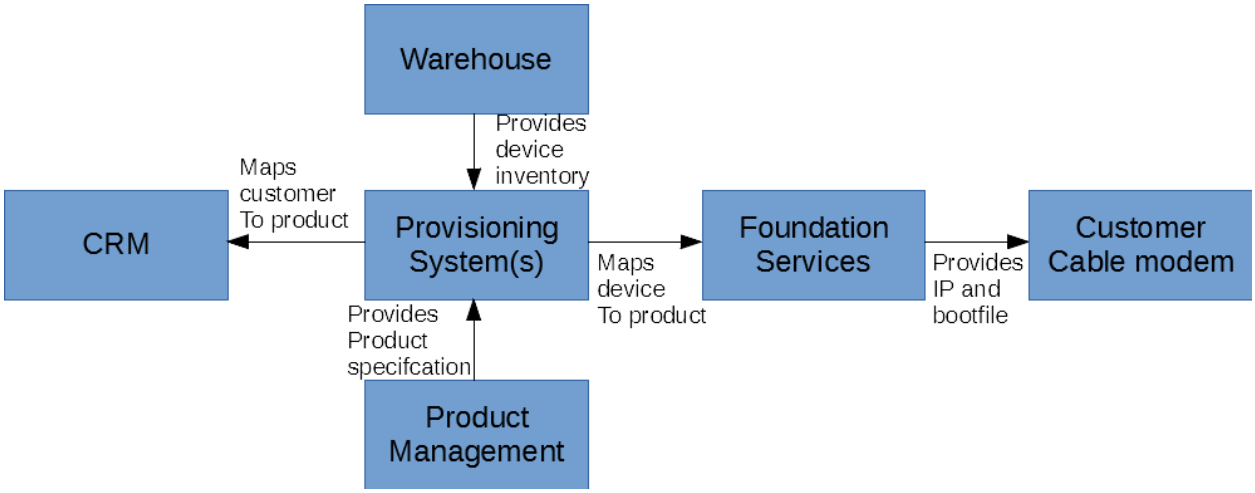

Figure 2 – Provisioning system interaction, simplified

Depending on the type of device one or more provisioning streams can be present. For example a cable modem is provisioned differently than a voice media terminal adapter. However, all devices have in common that they need to be provisioned in order to work properly.

1.1.2 Specifying a cable modem

As stated above, a CM as specified by the DOCSIS cable modem to customer premises equipment interface specification (Cable Television Laboratories Inc., 2014) is a network device that contains at least two physical interfaces, one to the Wide Area Network (WAN) or “uplink” side that allows communication with a CMTS via a HFC network and one to the Local Area Network (LAN) or “customer” side that allows communication with customer premises equipment, typically an Ethernet port. So in essence a CM is a device that “bridges” the gap between LAN and WAN.

Most modern cable modems come equipped with additional service modules called embedded service/application functional entities (eSAFE). These functional entities typically are embedded IP routers, voice gateways or set top boxes, which essentially make the “cable modem” a hybrid device between a cable modem and customer premises equipment. A visual representation of such a hybrid device can be seen in Figure 3.

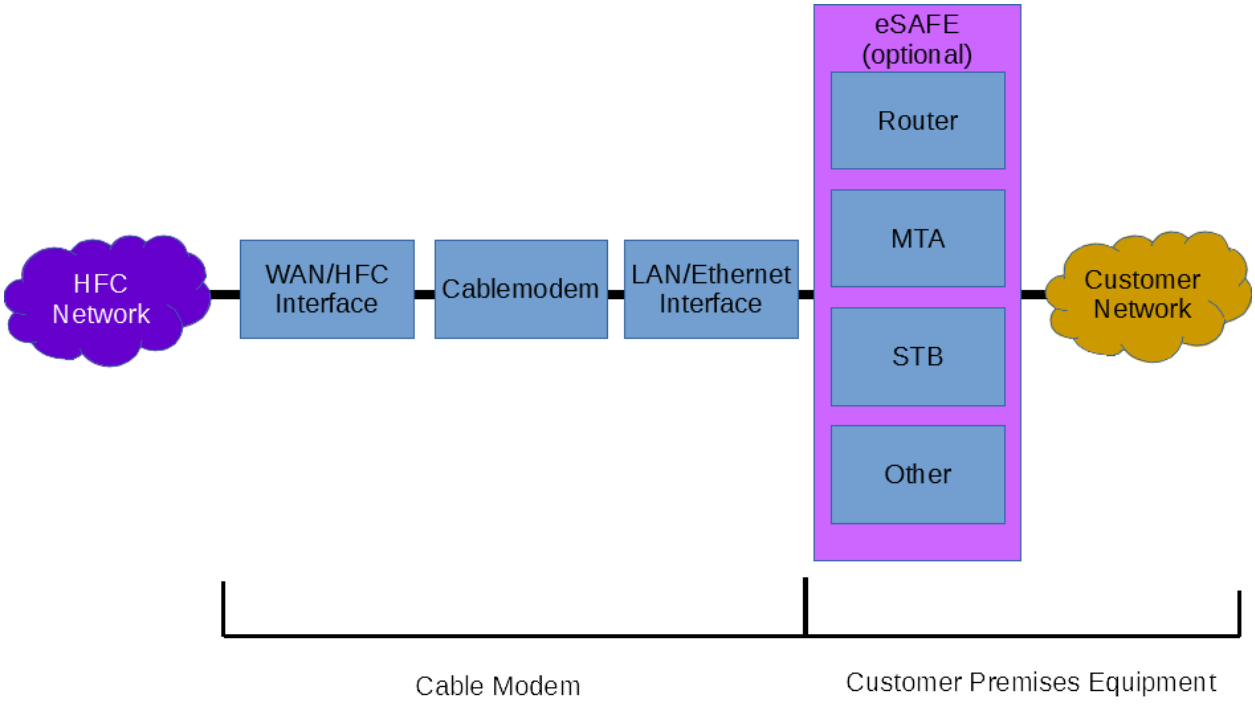

Figure 3 – simplified modern hybrid cable modem schematic

As the majority of the cable modems in the reference network adhere to the above schematic, whenever cable modems or CMs are mentioned in this paper it refers to CM/CPE hybrid devices.

During the device startup process, called booting, the CM initializes both its interfaces and establishes a connection with the CMTS. Once that's done, the embedded dynamic host configuration protocol (DHCP) client requests an IP address and a configuration file from the DHCP server in the provider network.

This configuration file, often also referred to as bootfile, is a binary encoded file that contains relevant configuration parameters for this specific cable modem, like upstream and downstream speeds, capabilities, access lists and firmware images.

This file is downloaded by the modem using the trivial file transfer protocol.

1.1.3 The uses of the trivial file transfer protocol

The TFTP protocol is a very old, but very lightweight lock-step data transfer protocol (OSI layer 5, DOD layer 4) built on top of the user datagram protocol (UDP) and was initially specified in RFC 783 (IETF, 1981) and later updated in RFC 1350 (IETF, 1992), allowing simple transfer of data between two Internet connected nodes.

A TFTP client just needs a TFTP server's IP address, a request type (RRQ for read request and WRQ for write request), a valid filename and a transfer mode to up/down-load something from/to the server. Lock-step means that for every data packet the sender transmits, the receiver needs to send a reply, acknowledging the packet. A visual representation of a client requesting a file from a server using the TFTP protocol can be seen in Figure 4.

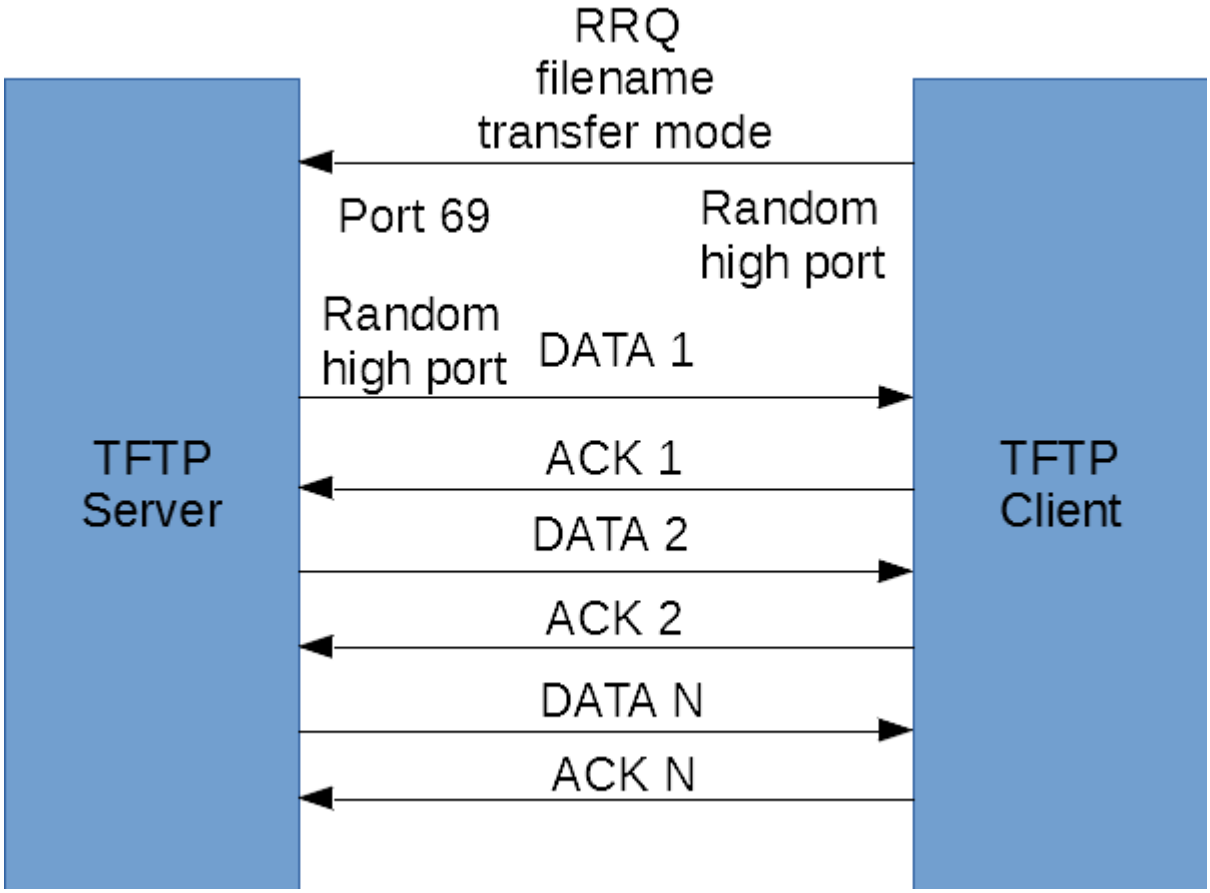

Figure 4 – TFTP protocol file download, no enhancement RFCs

Unlike other data transfer protocols like file transfer protocol (FTP) and secure copy (SCP), which both use TCP as a transport protocol, TFTP uses UDP, meaning integrity checks and retransmissions need to be handled on the application layer, rather than the transport layer. It also supports no authentication or encryption, is slow and generally considered very insecure.

Several enhancement RFC's were published following the years after the initial TFTP RFC was created, to improve the performance, however, none of these have had any effect on the security aspects or lack thereof of the protocol.

According to the DOCSIS, TFTP is also the only acceptable way of a CM receiving its configuration file during startup, making an array of TFTP servers a key component in all cable ISP networks. The same is also true for the Packet Cable specification and the startup process of media terminal adapters (MTA).

1.1.4 The configuration file

Configuration files for cable modems are essentially text files that contain specific operating parameters and settings, which have been binary encoded using a simple cipher string (by default "DOCSIS").

Depending on the environment and use case, these binaries are typically created long before they are actually used and uploaded to a TFTP server which allows the cable modems to download these.

Common ways to categorize configuration file use cases are:

- Creation method (manually, semi-automatic, automatic)
- Purpose (individual, class-based, generic)
- Creation time (preloaded, timed, “on-the-fly”)

Creation method and purpose is directly dependent on the capability of the provisioning system, while creation time is more of a blended parameter. By default, TFTP expects the file the client requests to be present on the file system at the time of the request.

1.1.5 Security aspects of cable modems

ISO 27001 and other security centric standards and recommendation consider it good practice to limit unnecessary exposure of internal systems to outside factors unless absolutely necessary.

To facilitate this, cable modems, which are considered an internal system, even though they reside on the customer premises are not put on the public internet, but are handed private IP addresses for management. These management IPs typically come from the 10.0.0.0/8 block, specified in RFC 6890 (IETF, 2013), which is not routed beyond the ISPs backbone network. In addition to the routing limitation, IP access lists on service and core router devices also prohibit the connection from customer public ranges to the block the cable modem management IP resides in.

Backend systems like the TFTP servers are typically protected in the same way, limiting the chances of unauthorized access, as TFTP does not support any other authentication or authorization. A simplified example implementation of cable modem and TFTP security measures can be seen in Figure 5.

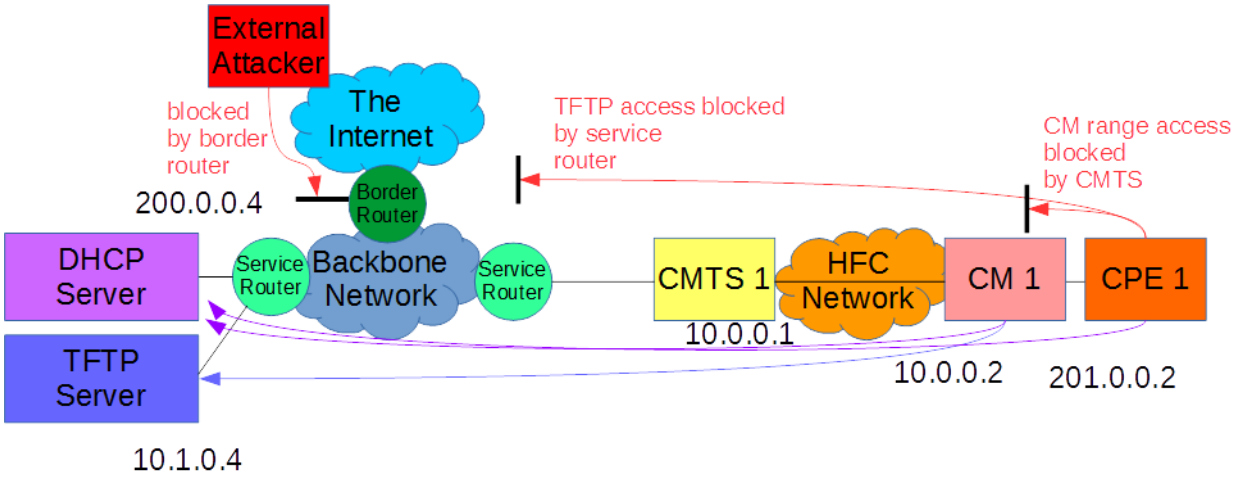

Figure 5 – common TFTP/CM security measures

1.1.6 Reference network

For real-life reference, the network of the Liberty Global affiliate company UPC Austria GmbH (UPC AT) was used in this paper. This ISP's operates both DSL and cable based networks and has a total of 484.800 internet access and 502.500 digital video customers (Liberty Global, 2015). The large size (>100.000 clients) of the ISP and the fact that about 84% of the clients are DOCSIS cable based, make this an ideal environment for the introduction of a dynamic TFTP service.

Like most cable ISP's UPC AT operates multiple local hub sites that house one or more CMTS, depending on the number of homes passed, per geographical region. These hubs are connected to the ISP's backbone using high-speed fibre optic cabling, while the CMTS connect into the local HFC network grid. While the HFC network is owned by UPC AT, in some cases the uplink cabling is rented, depending on the remoteness of the location.

In addition to the hub sites UPC AT also operates several datacenters, which house the servers and equipment necessary to run the provisioning and foundation services. Datacenters, like hub sites, connect to the ISP's backbone using high-speed fibre optics. A simplified overview of the reference networks structure can be seen in Figure 6.

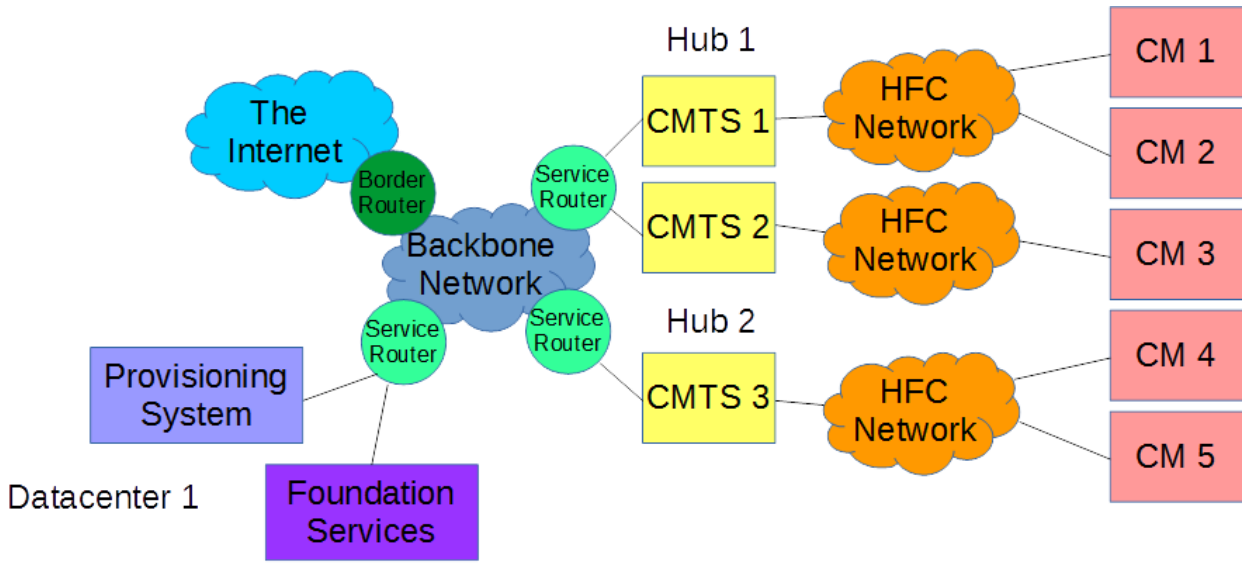

Figure 6 – reference network structure, simplified

The provisioning system, cSpire from Amdocs, used by UPC AT is a closed source, carrier-grade solution which allows only limited modifications that are typically very costly to implement. Customer devices are provisioned in cSpire on a client-class basis. Using the categories introduced in 1.1.4 this means, files are client-class based, manually created, preloaded and downloaded using the TFTP protocol.

The DHCP servers run Cisco Prime Network Registrar version 7 (Cisco Systems, 2016), which is also a closed source, carrier-grade solution, but allows used defined modification by providing an extensive API.

The current TFTP server solution in use is a slightly modified version of the open-source hpa-tftp server which can be freely modified or replaced with a different solution with moderate effort.

TFTP and cable modem security measures are implemented in a way similar to the description in 1.1.5.

1.2 Method of procedure

Given the framework conditions imposed by the DOCSIS standard and the fixed parameters of the reference network will make the development and deployment of a dynamic TFTP server solution a tricky, but not impossible endeavor.

Large modifications to the underlying provisioning systems are considered invasive and costly and are thus, out of scope of this paper.

1.2.1 Analysis of the reference network, the DOCSIS standard and the TFTP protocol

To prove that the introduction of a new service has had any effect, first a reference network needs to be specified to measure the results. This includes picking apart the currently used methods and processes, to find out how they work, how well they perform and which weaknesses they have.

Just because cable operators are bound by the DOCSIS specification and RFCs they require, doesn't mean there aren't any procedures within these limits that can dramatically improve performance or security aspects of the solution. A detailed analysis of the standards and protocols will show just how much is possible.

1.2.2 Define requirements and their key performance indicators

System requirements must be measureable in order to provide any meaningful data for the evaluation stage. All modifiable parameters gathered in the analysis phase need to be fitted with a key performance indicator that allows the new solution to be measured against the legacy setup.

1.2.3 Develop new theoretical model based on extracted factors

Once requirements and KPI are defined, a theoretical blueprint needs to be built, that will serve as a template for the prototype built in a later stage, and this template is then matched against the ISO/IEC 27001 guideline in respect to system security.

1.2.4 Design and develop a prototype

Once the blueprint is finished and sufficiently secured, design and implementation of a new prototype service can begin. All the requirements gathered in the previous stages will be implemented and the prototype's performance measured against the defined KPI.

1.2.5 Deploy the prototype in a live environment

With the prototype finished and performing above minimum requirements, it will be introduced into a live environment to evaluate its performance under real-life conditions.

1.2.6 Evaluate the results

As a final stage, the prototype's performance will be measured against the performance of the legacy solution as well as its robustness versus previously identified security threats.

1.3 Thesis focus

This thesis will focus on both the theoretical and practical implications of the introduction of a dynamic trivial file transfer protocol service in a large service provider network. This will help answer the question, what the effects on system security and provisioning process flows caused by a dynamic trivial file transfer protocol service in large service provider networks, are.

1.4 Objectives

The objectives of this paper are:

- Analysis of the reference network and its TFTP and provisioning related processes
- Analysis and evaluation of weaknesses in currently used methods
- Detailed examination of the RFCs and standards specifying DOCSIS, Packet Cable and TFTP protocols
- Define requirements and their key performance indicators
- Deduction of possible functionality limited by framework conditions
- Develop new theoretical model based on extracted factors
- Evaluate security and operational implications of the new model
- Create a service blueprint based on the results of the evaluation
- Map the blueprint to the limitations of the reference network
- Design a prototype

- Build and implement the prototype
- Deploy the prototype in a live environment
- Compare KPI of prototype with KPI of legacy solutions
- Evaluate the results

The most challenging task will certainly be the development of a functional prototype, given all the limitations imposed by the reference network and the DOCSIS standards, while the performance numbers should ideally be where they are now with the legacy solution. Interacting with what can be considered the heart of a cable provider is a challenge by itself as even a small mistake can have a huge impact on performance and customer satisfaction.

The evaluation of the results could also prove tricky, as there currently are no tools available to properly monitor some of the more obvious performance parameters.

2 REFERENCE NETWORK ANALYSIS

Due to the complexity of the reference network the analysis is split into three segments, these are:

- Infrastructure analysis, from the physical hardware to the IP routing layer (OSI layers 1-3, DOD layer 1+2)
- Provisioning flow analysis, will look at how a cable modem is registered in the backend systems
- Service flow analysis, explains DHCP and TFTP traffic flow detail (OSI layer 4-7, DOD layer 3+4)

2.1 Infrastructure analysis

Due to the size of the network, a full analysis would go beyond the limits of this paper, but since the infrastructure from the hub site to the customer premises looks very similar through the entire network, some elements were omitted (e.g. management networks) others simplified (e.g. server connectivity) because they lack relevance to the topic and only a single customer connection will be used as reference for the analysis. The network's infrastructure is split into three levels, which reflect the first three OSI layers, these are:

- Physical layer, the actual cable and/or Wi-Fi connections that transfer data bits
- Data link layer, provides data transfer and error correction between adjacent nodes and deals with data frames
- Network layer, deals with packets and packet forwarding from one edge of the network to the other

2.1.1 Physical layer

The physical layer typically consists of one or more customer devices connected to the embedded router (eRouter) part of the cable modem either via a copper Ethernet cable or a Wi-Fi signal. The cable modem itself is connected to the nearest fibre node using a coaxial cable, while the eRouter is internally wired to the cable modem. One or more wall plugs typically ease the deployment of cable modems but are completely transparent to the data transfer.

Fibre nodes are, as defined by the American National Standards Institute and the Society of Cable Telecommunications Engineers (ANSI/SCTE), a point of interface between a fibre trunk and the coaxial distribution (ANSI/SCTE, 2013). So it is basically a device with a coaxial

interface on one side and a fibre optic interface on the other that links it to the CMTS, sometimes also doubling as a signal amplifier. Fibre nodes are typically located in street cabinets between the hub site and the customer homes. The amount of fibre nodes connected to a single CMTS largely depends on the CMTS model and amount of line cards inserted.

The fibre optics from the fibre nodes terminate at a CMTS that resides in a hub site. The CMTS itself also has a fibre optic uplink to a set of service routers in the same hub site, typically using multiple 10 gigabit fibre optic Ethernet connections.

There are typically two service routers in each hub site, to provide redundancy in case of a catastrophic failure of one of the devices. By definition a service or edge router is a layer 3 device that aggregates traffic from the network edge and forwards it to the core. In the reference network, the service routers are connected to two sets of core routers, which reside in two different locations using direct dark fibre connections.

All hub sites directly or indirectly connect to the core routers, where all traffic is either forwarded to the border routers that connect to the internet, or to another set of service routers, behind which the internal services reside.

Similar to the service routers in the hub sites, the datacenter service routers, aggregate traffic from all the internal system and are linked to a pair of switches that connect not just the servers but also to a set of firewalls.

While the firewalls provide a certain level of endpoint protection, the servers themselves provide the actual services to the cable modems. Server and switch connections are typically achieved via 10G fibre optic Ethernet. Different to network level redundancy, server level redundancy isn’t achieved by placing two servers in the same location, but rather by placing two servers providing the same service in two different datacenters.

A drawing representing the components and connections on the physical layer can be seen in Figure 7.

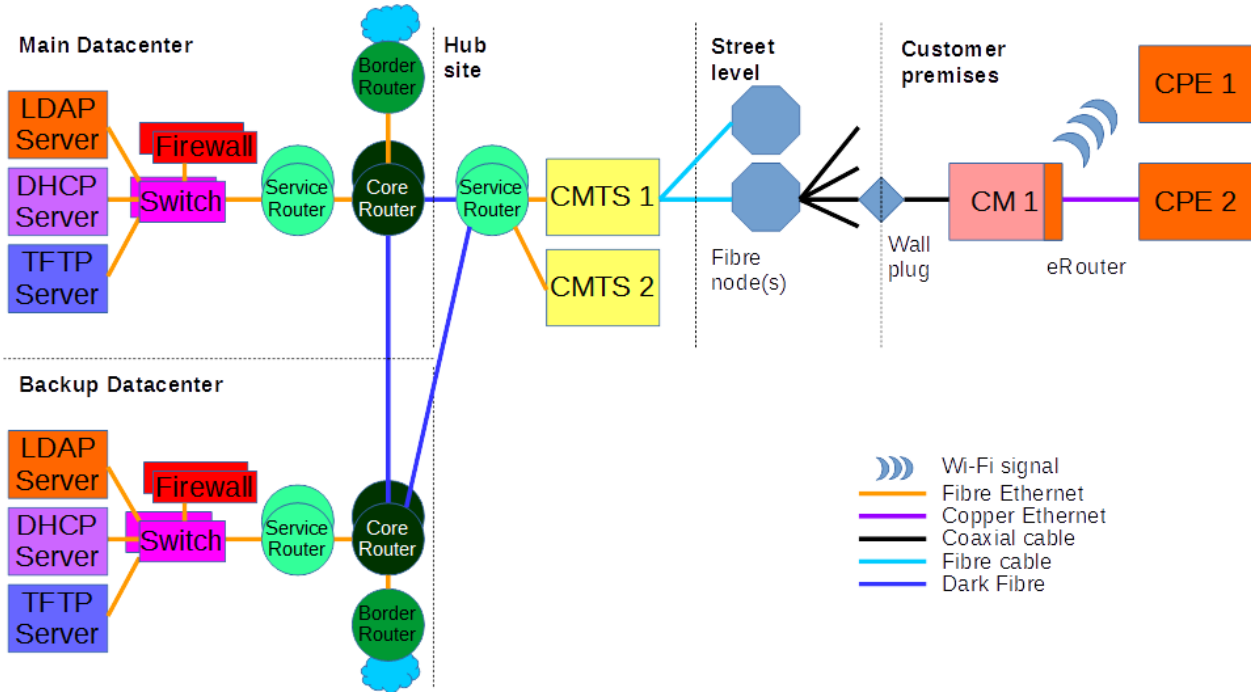

Figure 7 – Reference network infrastructure, physical layer

This drawing closely resembles the DOCSIS reference network as presented in the DOCSIS 3.0 specification (ANSI/SCTE, 2013), with modifications to represent the actual reference network implementation.

2.1.2 Data link layer

The data link layer (L2) encapsulates the data bits sent over the individual physical links into several independent broadcast or collision domains (CD) between the adjacent devices, where each device has its own unique media access control address in EUI-48 format, as specified by the IEEE 802.3 standard (IEEE, 2016). In the DOD architecture the OSI data link layer is part of the link layer. Device separation on the data link layer is used to increase security of the entire network by limiting the “visibility” range of each device to the minimum necessary to function.

Using virtual bridged local area networks, as described in the Institute of Electrical and Electronics Engineers (IEEE) standard 802.1Q (IEEE, 2003) it is even possible to separate collision domains on the same physical medium. This method is heavily used in the reference network on the datacenter level.

A view of the reference network from a data link layer perspective, as well as the collision domain separation can be seen in Figure 8. For picture clarity, the backup datacenter site was removed. However its setup mimics that of the primary datacenter setup, which is present in the drawing.

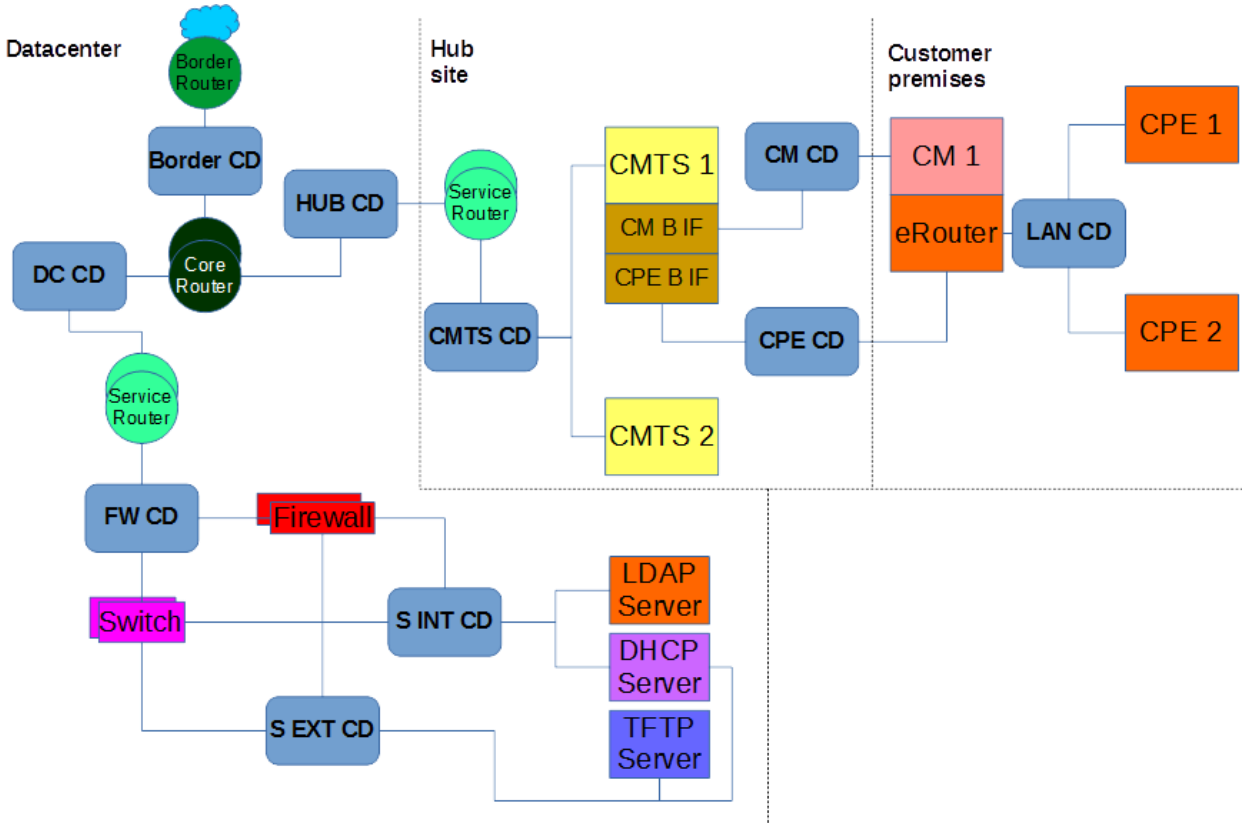

Figure 8 – Reference network infrastructure – data link layer

It is important to note that the physical medium is largely irrelevant to L2 connectivity, as seen on the LAN CD, where actually two physical media (copper Ethernet and Wi-Fi) share the same CD. On the other hand at the datacenter side, server internal and server external CD's share the same physical medium (fibre Ethernet), achieving logical separation by using VLANs.

CMTS, have multiple so called bundle interfaces that logically allow it to communicate with cable modems and eRouters on different channels, so that eRouters cannot "see" cable modems of other customers and vice versa.

2.1.3 Network layer

The network layer allows the individual collision domains to connect to each other using the IP protocol and packet forwarding.

In a clean network architecture, like the one represented by the reference network, only one IP subnet is assigned per collision domain, improving security and limiting crosstalk between clients.

Packets with destinations other than the local subnet are forwarded to the default gateway address, a router or virtual router redundancy protocol address (VRRP), typically assigned the first valid IP of the subnet.

While most of the datacenter equipment and all customer devices are assigned a single static gateway address, the core and backbone networks rely on dynamic routing protocols like Open Shortest Path First (OSPF), Border Gateway Protocol (BGP) and Intermediate System to

Intermediate System protocol (IS-IS) that can provide multiple routes to the same target using different links. As the backbone routing is not relevant to the topic, it has been omitted.

While CMTS, servers and firewalls are assigned static IP addresses, cable modems, their eRouters and customer devices are assigned IP addresses and gateway information using the Dynamic Host Configuration Protocol (DHCP), as specified in RFC 2131 (Dorms, 1997).

A schematic of the reference network in L3 view using placeholder IP addresses can be seen in Figure 9.

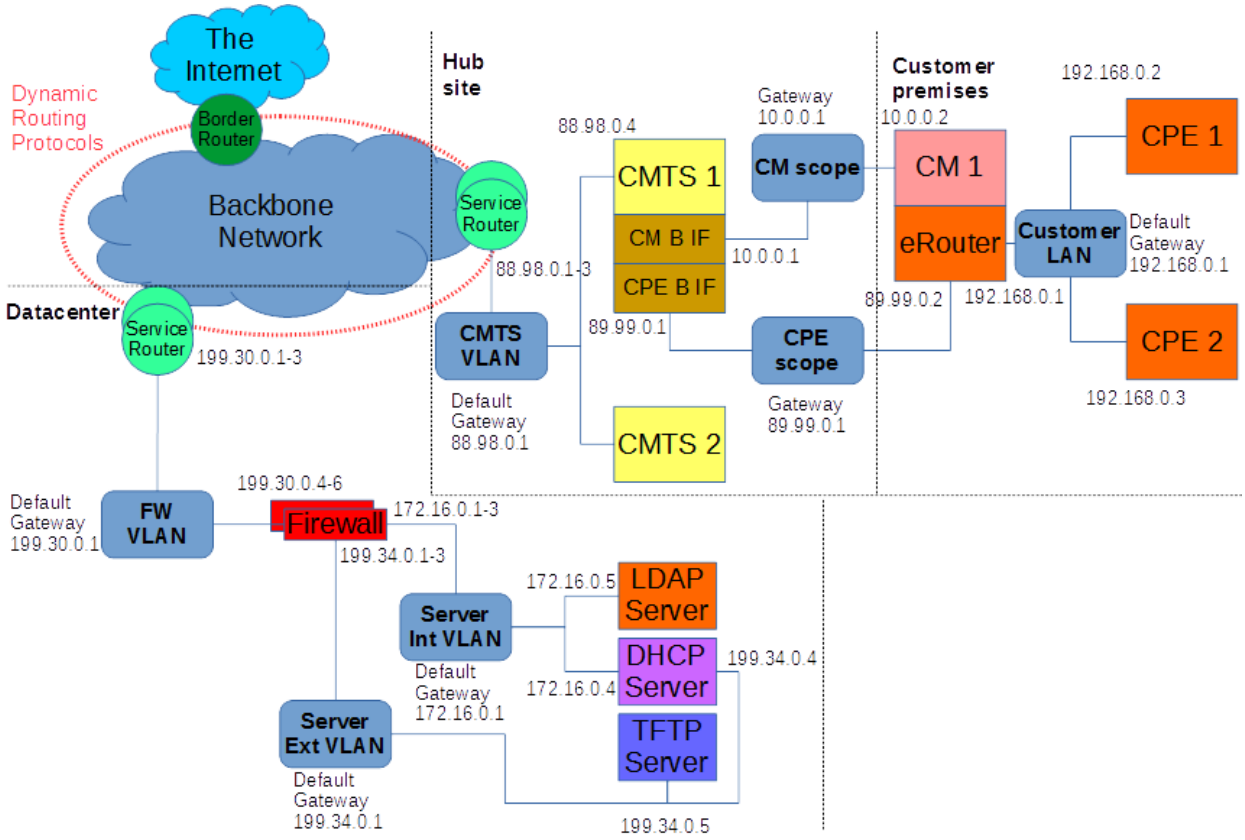

Figure 9 – Reference network infrastructure, network layer

Strict routing restrictions in the backbone network prevent the access of private IP space from public IP space (e.g. eRouter 89.99.0.2 cannot reach CM1 10.0.0.2, nor LDAP Server 172.16.0.5). Systems that do need to interact with public and private space IP addresses are either given an interface in a private and public subnet each, or special routing privileges in the backbone (e.g. CM1 10.0.0.2 can reach DHCP Server 199.34.0.4).

2.2 Provisioning flow analysis

As stated in the introduction, service provisioning is the act of assigning a specific product to a cable modem belonging to a specific customer.

The typical flow in the reference network is that a customer buys a product or service either through the webpage or a human agent. The product is chosen from a product list that contains a predefined set of products varying by up-/download speed and capabilities (commonly voice

over IP on/off, Wi-Fi on/off, Digital TV on/off). It' worth noting that the product list contains more products than the customer can choose from, since outdated products may still be assigned to existing customer but are not allowed to be sold to new customers.

The customer's order is then manually processed in the following way:

- Check for existing customer data (new or returning customer)
- Create/update billing ID
- Verify upgrade status (upgrading from existing product or new product)
- Verify line status (send task to field service crew for installation if not ok)
- Send new hardware if required (cable modem type is determined by product selection)
- Provision product (create new database entry or update client-class on existing device)
- Wait for customer to connect the device to start billing cycle

The provisioning database is a redundant Lightweight Directory Access Protocol (LDAP) database, as specified by RFC 4511 (Sermersheim, 2006), using two master servers in a central location, with several local replicas in case the connectivity to the central location gets interrupted.

An example LDAP entry can be seen in listing 1.

```
domains, at, d
 omains, chello
dn:
jrCNRMACAddress=1\,6\,00:ff:00:ff:00:01,ou=cnrInfra,ou=infrastructure,ou=v
 ienna,ou=domains,ou=at,ou=domains,o=chello
jrCMTSRoamingEnabledFlag: false
jrClientClass: fiberpower100_v4-sip
jrFirstBootFlag: A
jrCNRMACAddress: 1,6,00:ff:00:ff:00:01
objectClass: jrCNRDHCPDevice
objectClass: top
```

Listing 1 – sample LDAP client entry

Apart from the search path, which is predefined and can't be changed, the LDAP entry only contains two relevant attributes, jrCNRMACAddress, which is the MAC address of the client, and jrClientClass, which represents the provisioned product. This means that the provisioning system's only interaction with the DHCP service is the mapping of the client MAC to a product name.

For legacy reasons, LDAP entries are only done for cable modems, MTAs and STBs, never for eRouters or other CPEs, as the DHCP server is capable of identifying the cable modem they are attached to by extracting the relay agent MAC address from the DHCP Discover they send.

2.3 Service flow analysis

The service flow describes the relevant steps that the cable modem and its attached eSAFE go through during the startup process. That process can be split into two separate service flows, these are:

- The Dynamic Host Configuration Protocol service flow, assigns an IP address to the devices and connects them to the network
- The Trivial File Transfer Protocol service flow, downloads a configuration file and optional firmware image from the backend systems

The startup process for cable modems was initially defined in the DOCSIS 1.0 Radio Frequency Interface specification (Cable Television Laboratories, 2001) and remains largely unchanged to this day. Figure 10 shows this startup process and also highlights the two relevant steps for DHCP (purple) and TFTP (dark blue).

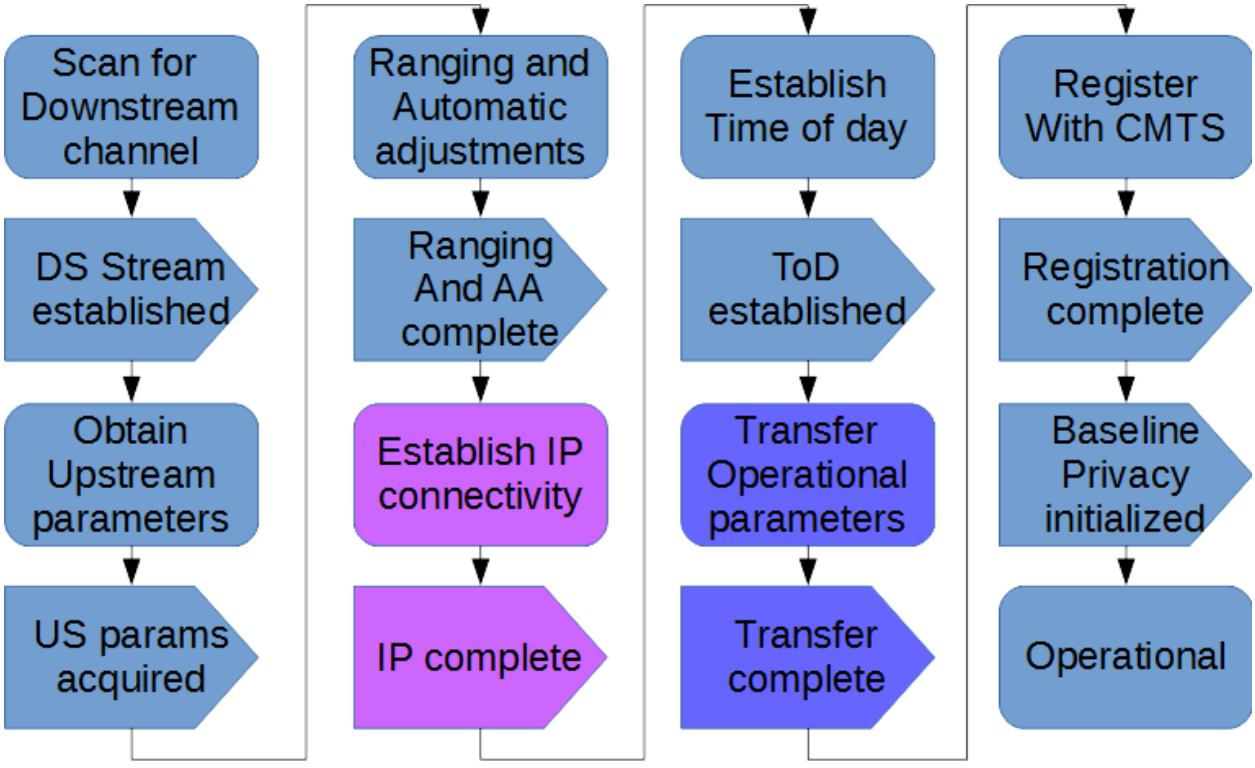

Figure 10 – cable modem startup process

While time of day (ToD) is not directly relevant to DHCP or TFTP processes, it is worth noting that in the reference network the DHCP server also has a ToD server running, which will respond to the ToD requests from the clients.

2.3.1 DHCP service flow

Chapter 2.1.3 explained that cable modems and eSAFE do not have static IP configuration, but instead use DHCP to receive an IP address and other network related parameters.

DHCP can normally only work between devices in the same collision domain (L2), however, the CMTS can capture DHCP broadcast packets from the clients and forward them to DHCP servers, configured as DHCP helper addresses, that don't need to reside on the same CD.

A typical DHCP handshake using the CMTS as DHCP relay can be seen in Figure 11 (Dorms, 1997).

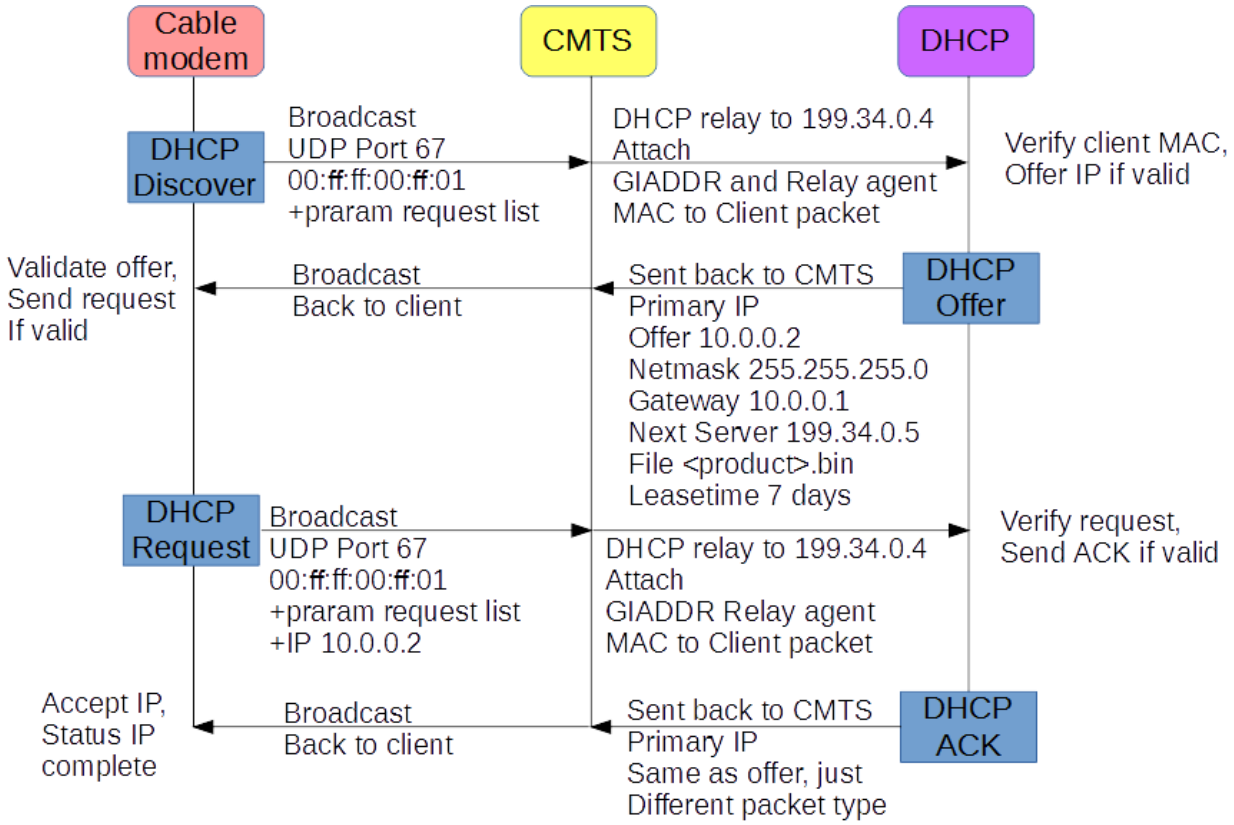

Figure 11 – cable modem DHCP handshake using CMTS as DHCP relay

DHCP handshakes for eSAFE look similar, the only difference is in the IP address range assigned and as stated in 2.2 the fact that the relay agent MAC attached by the CMTS is that of the cable modem.

The Cisco Network Registrar 7 (CNR) DHCP server in use in the reference network configures IP ranges in so called scopes (Cisco Systems, 2011). Each scope has a name reflecting the CMTS it belongs to, the type of scope and the IP subnet it contains. An example would be at-vie-sk11-cr01-cb1-cm-10.0.0.0-22, a cable modem IP range, with subnet 10.0.0.0/22 belonging to CMTS at-vie-sk11-cr1 cable bundle 1. Scopes are also configured with a router address, DHCP option 3, using the first available address in the subnet, 10.0.0.1, which is also assigned to the CMTS. In addition to the IP range, each scope also has a scope selection tag, a text identifier that tells the CNR which client-type is supposed to receive IPs from the scope.

For identification purposes the scopes are also arranged in scope groups, with the network containing the CMTS' loopback interface IP serving as the primary scope. All client scopes belonging to that CMTS are then logically attached to the primary as secondary. This must be

done, since the initial DHCP discover from the client is always sent from the CMTS primary IP, rather than the bundle interface IP.

The CNR is also configured with the products defined in the provisioning system. The product equivalent on the CNR is called a client-class. Client-classes contain a scope selection tag to tell the CNR which scope the client is valid for IP assignment and a policy name. Policies can specify additional Bootstrap Protocol (BOOTP) or DHCP options, as initially specified in RFC 951 (Gilmore, Croft, 1985) for the client-class in question. Two parameters that are always set for cable modem client-classes is BOOTP option 'file' (and DHCP option 67, TFTP file name, which directly reference the cable modem configuration file on the TFTP server. While absolutely necessary BOOTP option 'siaddr' and DHCP option 66, both representing the TFTP server address are not configured in client-class policies but rather as a global parameter in the default server settings.

The client-class name must match the product name in the LDAP database, which is queried by the DHCP server after receiving an offer or a request packet from the client. It's mandatory that all products present in the provisioning system are also present as client-classes on the DHCP server.

A slightly simplified representation of the LDAP communication and the server internal workflow described above can be seen in Figure 12.

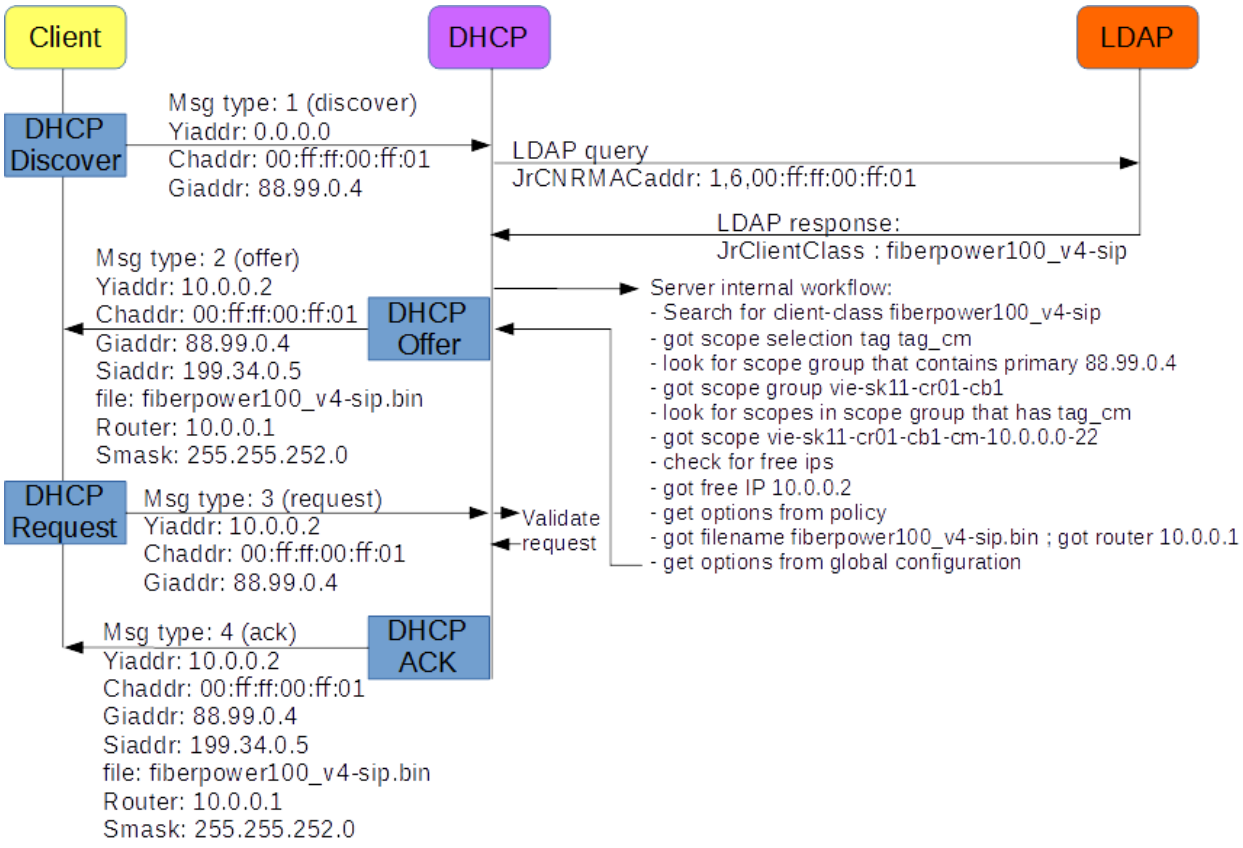

Figure 12 – LDAP communication and DHCP server workflow

2.3.2 TFTP service flow

As shown in the cable modem startup process, the TFTP download starts immediately after the ToD has been established, which typically is about 1-2 seconds after the DHCP handshake is complete.

The cable modem sends a TFTP read request (RRQ) to the TFTP server IP provided by the DHCP server in the siaddr or DHCP option 66 fields for the filename provided in the BOOTP file or DHCP option 67 fields.

As the TFTP protocol supports no authentication or client verification, nothing prevents the client from downloading any file from a server it can connect to, provided it can supply a valid file name.

A standard TFTP transfer was already represented in figure 5, there is however also one transfer method where the CMTS acts as a TFTP relay, typically used with tftp-enforce and dynamic shared secret options. Rather than having the bootfile encoded with the default DOCSIS string, the CMTS and the CM agree on a specific shared secret password that allows the CM to decode the bootfile. In this case the CMTS needs to intercept the configuration file download from the TFTP, then decode the file with the default string and re-encode it with the shared secret for the specific cable modem. A visual representation of this kind of download can be seen in Figure 13.

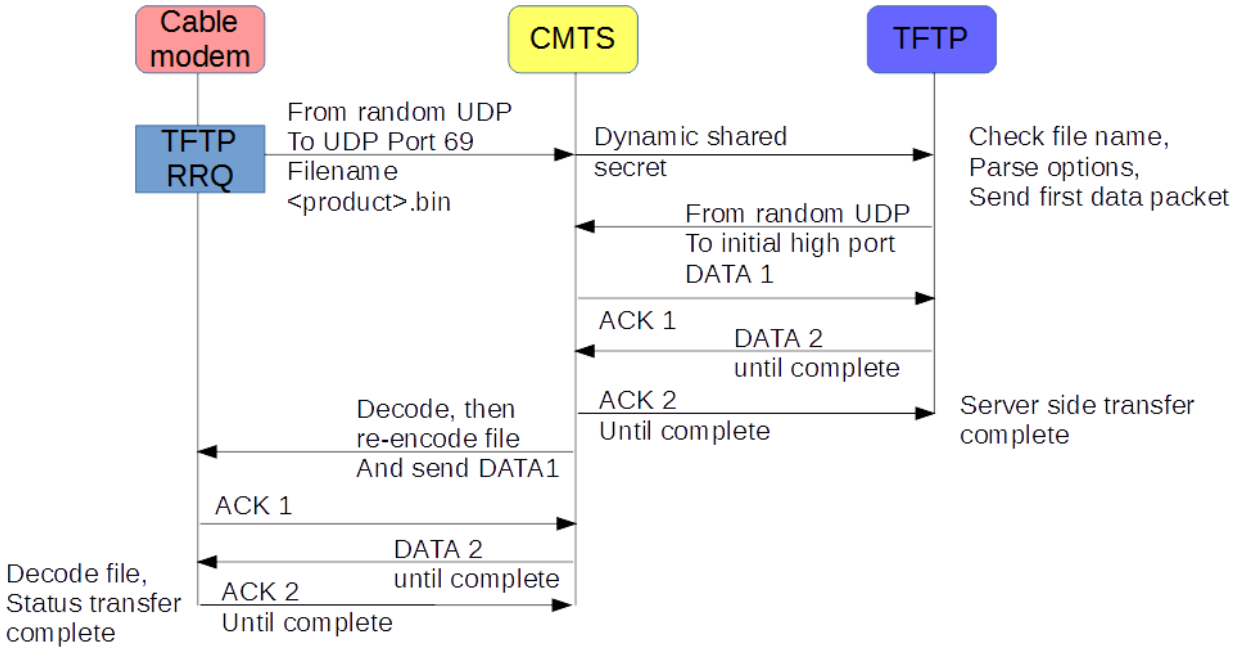

Figure 13 – TFTP configuration file download using shared secret

2.4 Attack vectors and outage scenarios

Being a large production network with several hundred thousand connected clients, the reference network regularly gets attacked or is target of fraudulent behavior, but outages can

also be caused by simple hardware defects or connectivity failures. Unlike the analysis before, attack vectors and outage scenarios will be viewed from a damage category perspective, these categories are:

- Fraud
- Security breach
- Denial of service
- Service degradation and interruption

This sub-chapter will review the historically most common attack types against and outage scenarios of the network, many of which have been encountered before and countermeasures against them have been taken.

As mentioned in the analysis chapter, all systems feature management network access, which is completely separate from the production networks. These are considered secure and unaffected by the type of service running on the system and thus out of scope for this review.

2.4.1 Fraud

The fraud category covers all activities that relate to a misuse of the service or accessing services the attacker is not entitled to, these include:

- Using a service not being paid for
- Using a service beyond the paid capacity
- Steal another customers credentials/service

According to the Communications Fraud Control Association's (CFCA) 2015 Global Fraud loss survey more than $38 Billion are estimated to be lost annually to fraud and fraudulent behavior, with subscription fraud and abuse of network, device or configuration weaknesses among the top 6 applied methods. (CFCA, 2016)

The easiest way to get a service the attacker is not paying for is by means of connecting a stolen, but registered cable modem to the HFC network. This type of fraud can easily be blocked on the HFC network level, since all lines that do not have an active service behind them are physically disconnected, so unless the thief also has a legitimate service on his connection, the cable modem will not be able to contact the CMTS. On a provisioning system level, this fraud can only be resolved by the owner of the stolen modem reporting the theft and by de-registering the modem. On the service level this fraud is nearly undetectable unless the DHCP server also verifies the source of the DHCP request. If it's coming from another but the registered location it should deny service to the customer.

If the attacker is using a non-registered cable modem, even more countermeasures are in place that prevents them from getting service. On the CMTS level, baseline privacy plus (BPI+), specified by (Cable Labs, 2008), prevents any cable modem from registering that does not have

a valid certificate from the ISP. The provisioning system, by the means of the DHCP server will also prevent a non-registered client from getting a working configuration file.

There are however known ways to apply custom firmware images to certain cable modem types that allow them to not just load the necessary certificates, but even allow the cloning of MAC addresses, making it an identical copy of a legitimate cable modem. An attack like this bypasses BPI+, the provisioning system and the DHCP/TFTP restrictions, however, the cloned modem must not be connected to the same CMTS as the original, as it will be prevented from registering twice.

There are currently no automated measures in place in the reference network that prevent MAC cloning, however network scans in regular intervals can detect them and tasks can be issued to field service personnel to manually remove/exchange them.

As this is a very sophisticated attack that requires considerable technical skill and also physical access to the original modem, the chances of having large numbers of cloned modems in the network is relatively low.

Using a service beyond the paid capacity is also rather complicated to achieve. It usually means the cable modem is either running a custom firmware that accepts local parameters rather than those from the configuration file, a modification of the LDAP entry to point to a different product, modifying the filename provided by the DHCP server or exchanging the bootfile on the TFTP. Local downloads are prevented in the reference network as the CMTS intercepts the configuration file download from the TFTP and will prevent the modem from using any values different than the ones in the configuration file. Modifications to LDAP, DHCP or TFTP are considered security breaches and will be covered in 2.4.2 below.

Unlike the MAC cloning example from above, where the theft of the identity doesn't impair the service of the original customer, stealing another customer's service usually has negative impact on the service or financial penalties. Telephony and Video on Demand (VoD) credentials would make prime targets for theft. As VoD credentials are device specific and do not use TFTP, they are out of scope for this review. Session Initiation Protocol (SIP) telephony on the other hand, as specified in RFC 3261 (IETF, 2002) does use TFTP. Typically implemented via embedded media terminal adapters (eMTA), which receive configuration files that contain the registration server and the line credentials, as specified in the Packet Cable 1.5 standard (Cable Labs, 2009), via TFTP.

While the standard allows encryption, it's not mandatory, meaning that anyone able to access the TFTP server and knowing the file naming scheme could possibly download MTA files containing the credentials of other users and use them for personal gain. Fortunately, the file encryption for MTA's is enforced in the reference network and the bootfiles, once written, can only be decrypted by one specific device. So while the files are possible to download they're effectively a jumbled string of bytes to anyone but the legitimate MTA.

2.4.2 Security breach

Security breaches deal with the acquiring of access privileges by an unauthorized user to systems which are not for public use. These privileges can typically be used to execute the following actions:

- Gain entitlements
- Extract information
- Disrupt service

According to the Information Security Breaches Survey 2015 90% of all large organizations in the UK were subject to a security breach in 2015, with the largest breaches resulting in an average cost of €1.77 million to €3.66 million, making security breaches very costly for the company getting breached (Department for Business, Innovation and Skills, 2015).

Breaching the security of a system largely depends on the amount of services running on the machine, the amount of open ports and the security layers around it (e.g. how many hoops an attacker has to jump through before they can do anything on the target system). In general, breaching the security of a system is considerably more complicated than the examples in the fraud category, since it requires in depth knowledge about the software in use, its weaknesses and a detailed topology of the target network.

As explained in the fraud chapter, entitlements can be gained through different means, one of them being the replacement of files on the TFTP server with modified copies. The RFC 1350 standard requires TFTP server to allow both read and write requests, so if the configuration of the service and the file permissions allow it, anyone with access to the cable modem IP range could potentially upload and replace existing files with modified copies. The same can potentially be said about server configuration files, if the TFTP service is configured in a way that the root directory (usually /tftpboot) can be escaped from. In short, great care must be taken when configuring the service itself as a simple misconfiguration can leave the server wide open for attacks.

To counter these threats, the reference network's TFTP servers feature a secure standard configuration that disallows write requests and locks down the TFTP root directory. In addition, the master copy of all bootfiles resides on a central system, which doesn't run a TFTP service and updates the TFTP nodes in regular intervals. So even if a file was changed locally, the central copy would overwrite it.

Gaining entitlements via the DHCP server is possible, but would require an exploitable flaw in the Cisco CNR software, or the ToD service. Both of which have so far not been encountered.

Breaching the provisioning system to gain entitlements is considerably more difficult, since the provisioning system and even its local LDAP copy, can't be accessed from the outside, unlike the TFTP or DHCP nodes. It would require the breaching of an intermediate system that has access to the provisioning system itself and the credentials for the provisioning database to do any modifications.

There are different ways of extracting different kinds of information from the different systems, the simplest one being the extraction of bootfile configuration data. By design, any device with an IP address from a cable modem or MTA range, can connect to the TFTP server and download any file. So given a valid file name and a hacked cable modem, an attacker can download TFTP configuration files and decode them, as they feature no kind of encryption, while still containing some information with security relevance (e.g. SNMP community strings, access lists and management interface default passwords).

It's also possible to estimate the possible amount of devices in the target network, by guessing filenames and just trying to download anything on the TFTP server.

While not a huge threat, both of these practices are currently possible in the reference network, and while the MTA files are encrypted, it's still possible to glean some information from the bootfiles, like amount and manufacturer, which could lead a potential attacker to possible device exploits.

As soon as the security of a server is compromised, either by exploit or misconfiguration, the access gained can also be used to negatively affect the system's response to legitimate requests. This can range from the deletion or rewriting of files on the TFTP server, to shutting down services or reconfiguring network interfaces.

While easy to notice with proper monitoring in place, the potential damage caused is considerable, meaning security breaches that allow the attacker to gain control of the target system must be prevented at all costs. Typically system integrity is maintained by keeping the operating system (OS) and the application up to date and patched with the latest security fixes, as well as having a secure set of configuration parameters that are reviewed in regular intervals.

2.4.3 Denial of service

Unlike security breaches which typically aim to remain undetected for as long as possible, denial of service (DoS) attacks or their more sophisticated cousin distributed DoS (DDoS) are overt and almost immediately noticed.

According to the Neustar April 2016 DDoS Attacks & Protection Report 82% of all attacked companies in Europe the Middle East and Africa were hit by a DDoS attack more than once in 2015, meaning that getting hit by a DDoS attack is not a matter of if or when, but rather, how often (Neustar, 2016).

DoS attacks deal with overloading system components with high volumes packets that appear valid, so they are either slow to respond to or do not respond at all to legitimate traffic. While they are easy to detect, compared to security breaches, they are considerably harder to protect against and their effects more complicated to mitigate.

Due to their necessary exposure, DHCP and TFTP services can be exploited by DoS attacks. Typically a client sends a large volume of DHCP or TFTP requests without completing the handshake, causing a large session backlog on the affected server and potentially taking up all

the buffer space reserved for client queries on the server node, so that legitimate requests can no longer be processed.

On the server level, behavior like this can be countered by limiting the amount of sessions any single client can initiate and mitigated by increasing the available server buffers, sizing the hardware higher than recommended and by lowering the number of connected clients to any single server node (e.g. placing more nodes in the field).

Performance tests show that a DHCP server node can typically handle about 3000 client requests per second in a production environment, which is dramatically lower than the number advertised by Cisco, but still considerably more than required to handle routine workloads (typically less than 50 requests per second, per node). Given the way that DHCP and TFTP sequences are connected, TFTP server performance should not be lower than that of the DHCP server.

In rare cases, such as a region wide power failure that causes a large amount of cable modems to restart at the same time, the effects of a DDoS attack can be mimicked by legitimate traffic, hence why server environments should always be designed around possible maximum spike traffic, rather than averages.

2.4.4 Service degradation and interruption

As explained above degraded service isn't always the result of an attack, it is just as likely that a physical defect is to blame. Physical defects can cover a wide range, but can be categorized in the following way:

- Node level defect
- Datacenter level defect
- Backbone level defect

Node level defects are limited to an individual server node. This can be a crashed service, a damaged hard drive or power supply or simply an unplugged cable. The typical effect is that the service running on the node is unresponsive to client requests. In the reference network, node level defects are mitigated by creating local redundancies, such as hard drives in redundant arrays of independent disks (RAID), dual server power supplies, and multiple network links for each server and backup servers that provide the same service. Additionally services running on the server are monitored and reported to a central service desk in case of an outage.

Datacenter level defects typically affect multiple services housed in a single datacenter. Common defects are damaged routing or switching equipment, network uplink cuts and power outages. To counter these problems, routers and switches are usually set up in redundant pairs that stay operational even if one device shuts down. Similarly, backbone uplinks are created with redundancy, so if the primary path fails, the backup is automatically used without service interruption. Power failures can be handled by having two separate power grids inside the datacenter, where each device is connected to both power grids at the same time. On top of that, each grid also has its own backup generator.

Backbone level defects can cause service disruption in several regions at once. Common forms of this defect are a link loss on the internet uplink and the disconnection of an entire datacenter or hub location. The dual uplinks described in the datacenter level defects can help mitigate a uplink outage, as well as creating geo-redundant datacenters, that both run the same services and automatically switch over clients if one site goes down. Generating locally contained service clusters that remain operational even if the connectivity to the central provisioning platform fails, is another excellent countermeasure deployed in the reference network.

3 DEVELOPING A THEORETICAL MODEL

This chapter focuses on building a theoretical model for the new TFTP service within the framework parameters set by the reference network, the DHCP and TFTP standards, as well as the DOCSIS specification. Several important performance and security factors have already been described in the previous chapter, which will help define the important key performance indicators (KPI) as well as required functionality.

The imposed limitations, the KPI and the functionality requirements will then be used to craft a blueprint for a new service, which will then be analyzed from a provisioning system, operational and security perspective. While provisioning and operational impact will be judged, based on the current TFTP service implementation of the reference network, the security guidelines of the ISO/IEC 27000 standard (also known as and derived from British Standard 7799) will be used to gauge the security implications of the new service.

The basic expectation for the new service is that it can be used as a drop in replacement for the current solution, with minimal to no impact to the provisioning service flow, considerable improvements on the security aspects and a reduction in operational effort, while maintaining responsiveness and performance within specified parameters.

3.1 Scientific theory

The general systems theory assumes, that replacing any part of a system, has effects on the system as a whole (von Bertalanffy, 1968). When viewing the reference network from a systems engineering perspective under the aspect of TFTP related service flows, it becomes obvious that modifying the TFTP service in any way, has effects, not just on directly interacting components, but also indirectly connected ones.

While effects on directly connected components, like administrators (positive result), eligible clients (no change) and possible attackers (negative result) can be dictated by the functionality of the service, effects on indirectly connected components like the DHCP service and the provisioning system are not as obvious, can only be assumed at this point and will have to be gauged solely by comparing KPI results of the legacy and the new service. A representation of the systems theory view of the reference network can be seen in Figure 14.

This allows the forming of the following hypothesis:

- H0: The introduction of a properly designed dynamic TFTP service in a large provider network reduces the operational effort involved in creating DOCSIS configuration files, increases systems security against attackers and is completely transparent to the clients, while having a calculable effect on the DHCP service and the provisioning system.

- H1: The introduction of a properly designed dynamic TFTP service in a large provider network has no effect on operational effort, does not increase systems security and affects client interaction, while having a non-calculable effect on the DHCP service and the provisioning system.

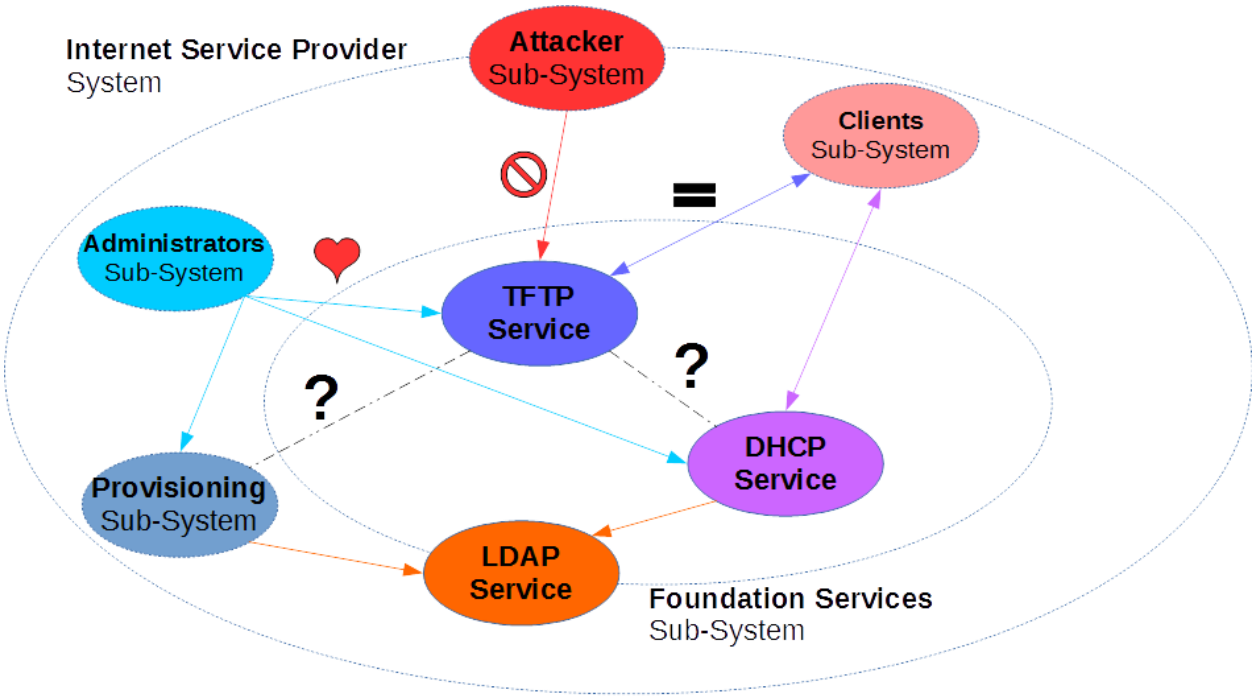

Figure 14 – Reference network in systems theory perspective

3.2 Correlation with ISO 27000

The ISO 27000 family of standards is synonymous for information security and its best practice implementations. While not all systems are ISO 27001 certified, considerable effort has been put into the reference network to establish a near ISO level security baseline. The new service is also required to adhere to the clauses and security controls of the ISO standard. The following 9 of the total 14 clauses, as defined in ISO 27001 (ISO/IEC, 2015), have been deemed relevant to the service:

- Asset Management
- Access control
- Cryptography
- Operations security
- Communications security
- System acquisition, development and maintenance
- Information security incident management
- Information security aspects of business continuity

- Compliance

Relevant aspects of *asset management* cover the responsibility for assets, and the information classification. As responsibility is currently split between the corporate engineering team, which designs solutions and implements new servers, as well as the local operations team, which covers day to day operations, including basic troubleshooting and configuration file creation, no changes are foreseen regarding this security control. Similarly, information classification remains largely unchanged, as the new service deals with pre-existing data and re-created configuration files based on company standard templates.

The access control clause covers the business requirements, the user access management and responsibilities as well as the system and application access. This is perhaps the clause that the new service will have the biggest impact on. While the user access and responsibilities as well as the system access are well defined, business requirements and the implied limit imposed on information access (the configuration files) can be currently considered insecure.

Cryptography, while not part of the service itself, due to restrictions imposed by the TFTP protocol as well as the DOCSIS standard, is still a relevant aspect, when it comes to systems and database access.

Important aspects of *operations security* include operational procedures and responsibilities, backup management, logging and monitoring as well as technical vulnerability management. As mentioned in the asset management clause, operational responsibilities are already well defined. However the operational procedures will require updated reflecting the changes required by the new service. As the new service is distributed and requires no internal stateful data, backups, other than copies of the source code, which are maintained by a version control system are unnecessary. Logging and monitoring as well as technical vulnerability management are key factors that the new service will need to implement.

Network security management and information transfer are the security controls of the *communications security* clause. While information transfer is mostly predefined by the network implementation of the reference network and the TFTP protocol, network security management is a key feature of the new service, when compared to the legacy solution, as it should not only be able to authenticate the client but also confirm it has authorization to access a certain file on the server.

Two security controls stand out when reviewing the *system acquisition, development and maintenance* clause. Security requirements of information systems, define that maintaining information security is relevant for the entire life cycle of the service. This implies that any further development in the code still needs to adhere to the same security guidelines as the initial service. Test data protection and the ability to test and re-test each new implementation of the service mark the second relevant security control of this clause.

Information security incident management implies that the new service should be able to report abnormal client behavior as well as potential breach attempts directed at the service on a regular basis and without user intervention. Mechanisms to detect these should be included in the code base.

Redundancies are the only relevant security control in the *information security aspects of business continuity*, that isn't already covered by any of the above. From a service perspective this means that it's not just required to be redundant, like by implementing multiple services in parallel, but also the ability to use multiple redundant data sources and revert to an emergency fallback mode that, while lowering certain security aspects, still provides the service to legitimate clients, in case none of the data sources are available. The idea being that the service is a key component in in client provisioning and that a temporarily loss of security measures is preferred to a complete service outage.

The last relevant clause is *compliance*. Information security reviews require that the service is implemented in a way that it respects security guidelines put in place on the server it operates on and does not operate beyond parameters specified by those guidelines.

3.3 Expectations

The expectations for the new service can be split into four categories, these are:

- Required capabilities
- Desired capabilities
- Optional capabilities
- Unwanted properties

Parameters are derived from the network's requirements, as explained in the analysis chapter, protocol specifications and defined standards.

3.3.1 Required capabilities

Required capabilities are non-negotiable feature sets that the new model must support in order to be a viable alternative to the legacy service. Several requirements are mandated by the service provider network and the DOCSIS standard and noted as such. The requirements and their sources are:

- RFC 1350 compliant binary TFTP downloads must be supported. Specified by DOCSIS standard (Cable Labs, 2015)
- Must be compatible with existing server and network infrastructure. Mandated by reference network.
- Must be compatible with existing provisioning system. Mandated by reference network.
- Must be compatible with Cisco CNR. Mandated by reference network.
- Must be scalable. Mandated by reference network.
- Must be compatible with existing TFTP directory structure. Mandated by reference network.

- Must provide feedback/logging on a per client basis. Recommended by ISO 27002
- Must not allow modifications of the underlying OS or file system. Mandated by reference network and ISO 27002
- Must work autonomously and generate regular usage reports. Mandated by reference network and recommended by ISO 27002.

3.3.2 Desired capabilities

Desired capabilities are what set the new model apart from the legacy solution. These functions improve the basic concept within the specified confines of the framework parameters. The capabilities are:

- Implement RFC 2347, 2348, 7440 option capabilities for TFTP transfers, as well as the hardware and network address option draft, for increased performance.
- Client source verification and transfer request validation, for increased security.
- Automatic failover, for increased resilience
- Client specific on-the-fly configuration file encoding, to lower operational effort
- Modular construction, to allow for more modifications later on

RFC 2347 defines the capability for additional options for the TFTP protocol. These options allow an increase in bytes transferred per data block (TFTP block size option, RFC 2348) and the modification of the amount of data blocks that can be sent before the client needs to acknowledge it (TFTP window size option, RFC 7440). Both these options can considerably lower the time it takes to transfer data to the client and thus increase the amount of clients that a single TFTP node can handle.

The hardware and network address option for TFTP draft (Zeng, Evans, 2005), allows the packaging of the clients actual IP address in the TFTP read request packet, in case the TFTP relaying has been enabled on the CMTS. It helps to identify the actual client. Due to the fact that this is only an RFC draft and not an actual RFC, not all CMTS vendors support this option. However having this option to identify clients can still be considered beneficial.

Client source verification and transfer request validation are functions that aim to find out if the client trying to download a file is actually entitled to the download and is supposed to be doing the download at the current time. As the DOCSIS specification requires, a client is only supposed to download a file if it just completed its DHCP cycle and ToD request.

A TFTP server node should not be a single point of failure (SPOF), so in case anything happens with the service running on the node, it should either try to self-heal, hand off the IP address to another server node or make sure that clients are automatically redirected to a working node.

Configuration file creation is currently a manual process in the reference network, where client-class based files are created to cater for multiple cable modems for a specific product. While vendor specific parameters do exist, it still requires considerable amounts of testing to update

configuration parameters that then have to work in any device. By creating client specific configuration files on-the-fly, the cross-device testing effort can be saved and files no longer need to be preloaded to the TFTP server nodes.

The software should be built using well know software design patterns as defined by elements of reusable object-oriented software (Gamma, Helm, Johnson, Vlissides, 1995), making the product modular and components exchange and extendable.

3.3.3 Optional capabilities

Optional capabilities further enhance the new solution by adding complementary features. They are considered “nice to have” and may be omitted from implementation to be added at a later stage. Suggested optional capabilities are:

- Administrative graphical user interface (GUI)
- Real-time reporting
- On demand MTA configuration file generation and encryption
- Individual client adjustments
- The ability to handle hypertext transfer protocol (HTTP) traffic in addition to TFTP
- RFC 2349 support

A GUI allows service administrators to comfortably manage the TFTP solution, check server status and update/exchange configuration file templates.

Real-time reporting builds on top of the regular usage reports, by providing them on the spot, rather than at predefined intervals. This can be used to provide monitoring and performance data on an even more detailed level.

MTA configuration file creation is considerably more complicated than creating cable modem bootfiles as the data contained in the configuration file is individual for each client and only present during the initial provisioning of the MTA. On top of that, MTA files in the reference network are not just encoded but also encrypted using Advanced Encryption Standard (AES) and Data Encryption Standard (DES) as specified by the National Institute of Standards and Technology (NIST, 2001), utilizing the public key of the individual client.

Individual client adjustments allow the TFTP service to permanently set certain parameters for a single client. These adjustments are also packed into the configuration file of the client and require an additional data store to save the information in.

While configuration file download *must* be done using the TFTP protocol, firmware downloads can be done using HTTP, which considerably increases data throughput if a client does not support TFTP block size and window size options. As firmware images and configuration files share the same file system, providing HTTP access to certain sub directories using the same server software can be considered a nice bonus.

RFC 2349 specifies the TFTP transfer timeout and file size options, while neither improves data throughput, making the transfer timeout variable can help in making sure that even clients with bad connections can get properly online. It also ensures that the dynamic TFTP service implements and supports all TFTP related RFCs.

3.3.4 Unwanted characteristics

Unwanted characteristics specify parameters that the software should avoid at all costs.

- Licensing costs
- Only useable in the reference network
- Specialized hardware or software

Being the result of scientific research, the solution should not rely on any closed source 3rd party software that requires licensing, even though some of the components in the reference network are. It should also be generic enough so it can work in any kind of ISP environment with only minor adjustments.

The solution should also not have to rely on any specialized hard or software to operate. Existing system capacity should be adequate to accommodate the solution.

3.4 Key performance indicators

KPI help to make non-binary service characteristics quantify- and measurable. KPI are dictated by service requirements and quantified by reference network parameters, the derived KPI are:

- Requests per second
- Number of nodes
- Time to deploy
- Time required to create a configuration file
- Time required to change a configuration file
- Failover convergence time
- Amount of modifications required on the DHCP server
- Number of modifications required on the Provisioning system
- Report quality
- RFC conformity
- Unauthorized client rejection rate
- Data throughput rate

Requests per second define how many concurrent TFTP read requests a single server node can reliably handle. Dictated by scalability requirement, target value equals legacy TFTP server node performance.

Number of nodes defines the amount of hardware nodes the service can run on in parallel. Mandated by scalability requirement, measured in number of nodes, target value should be equal to that of the legacy solution.

Time to deploy, quantifies how long it takes to deploy a new server node within the reference network. Dictated by scalability requirement, target value should be less than twice the time to deploy of a legacy TFTP server node.

Time required to create a configuration file, quantifies how long the system takes to generate a non-existent cable modem configuration file. Mandated by on-the-fly configuration file encoding, target value should be less than 1000 milliseconds as required by TFTP timeout value.

Time required to change a set of configuration files, quantifies how much time an administrator must spend to change the contents of an entire set of configuration files. Mandated by compatibility with existing server and network infrastructure, target value should be half the time it takes to manually modify a configuration file.

Failover convergence time, defines the time it takes for the system to return to a working state in case any component fails. Mandated by automatic failover, target value should be less than 60 seconds, based on DOCSIS recommended DHCP cycle time.

Amount of modifications required on the DHCP server in order to make the DTFTP service work. Mandated by compatibility with Cisco CNR, target value measured in man days (MD), should be lower than 5, is dictated by reference network.

Amount of modifications required on the provisioning system. Mandated by compatibility with existing provisioning system, target value measured in MD, should also be lower than 5, is dictated by reference network.

Report quality indicates how much logging data the service provides. This is mandated by feedback and logging on a per client basis, target value is at least two log messages per client (read request and either a transfer complete or error with appropriate error message), as defined by legacy solution.

RFC conformity defines how accurately the RFC specification is implemented in the service. Mandated by RFC 1350 compliancy, target value is 100%, however, unneeded protocol implementations like write request and non-octet/binary transfers may be omitted due to the fact that they are not used by clients.

Unauthorized client rejection rate, measures the accuracy of the service in regards of being able to identify clients entitled for downloads. Mandated by client source and transfer request validation, target value is 100%, as any non-authorized client should not be able to receive configuration files.

Data throughput rate, measures the number of bytes the service is capable to transfer per second. Dictated by scalability, measured in megabytes per second, target value should match at least half of that of the legacy platform.

3.5 Service blueprint

Based on the requirements defined above a generic blueprint for a dynamic TFTP service can be created, which consists of at least 4 modules that interact with each other, these components are:

- Network socket listener
- Client validator
- Configuration file creator
- Data transporter

The network socket listener (SL) registers a network socket on the host's operating system, that listens to UDP port 69 (and optionally a second, listening to TCP port 80, in case the HTTP service is also implemented). The socket listener's only job is to listen for TFTP packets from cable modems and hand them over to the client validator.

The client validator (CV) examines the initial TFTP packet, parses it for RFC 1350 conformity, extracts the client's identity and verifies it and the file requested against an external data source that holds both client identity and configuration file name information. If acceptable the packet is passed on to the configuration file creator, if not, the request is rejected with an RFC 1350 conforming error packet.

The configuration file creator (CFC), checks the host's file system if the file requested is already present, if not, it queries another external data source, which holds the configuration file data and additional client information, then proceeds to generate the requested file and writes it to the file system. Once the file is present, its handed over to the data transporter module for transmission.

The data transporter (DT) registers another network socket with the OS and starts transfer of the requested file. Depending on options requested by the client, the DT adjusts the packet transfer accordingly and also takes care of transmission errors as specified by RFC 1350.

A graphical representation of the blueprint can be seen in Figure 15.

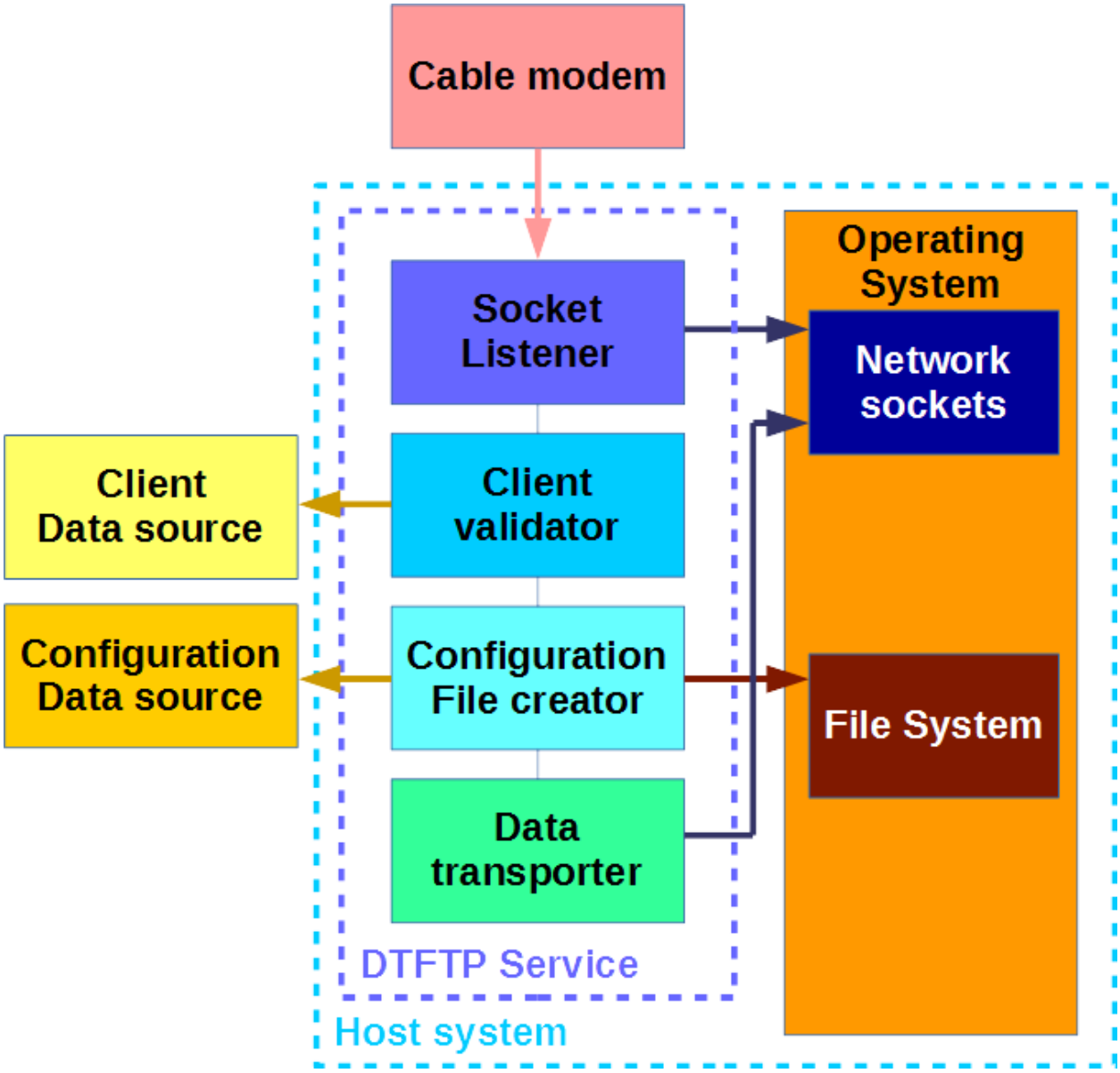

Figure 15 – generic DTFTP service blueprint

4 DESIGNING A PROTOTYPE

Using the blueprint and the requirements defined in the previous chapter, this chapter will focus on the design of the actual service prototype, which will serve as a base for the implementation stage. For easier reference, the new service has been codenamed “Sentinel”, due to its ability to identify clients and protect the data stored on the TFTP server.

Several design choices, defining the structure and capabilities of the Sentinel, in addition to possible future expansions, will be made throughout this chapter, these choices are:

- Implementation language
- System environment
- Component structure and data sources
- Blueprint to component mapping
- Systems interaction
- The Client Database structure

4.1 Choose implementation language

Unlike other choices in this chapter which can theoretically be changed, with moderate effort, after implementation has started, the implementation language is more or less static, since it forms the core and large parts of the structure of the service, making it “the” most important design choice.

According to the IEEE Spectrum there are 48 different programming languages with various specialties in use today, 36 of these are capable of creating enterprise grade software (Diakopoulos, Cass, 2016).

To make the language selection process easier, several requirements will be presented, these are:

- Must run on Red Hat Enterprise Linux or CentOS version 5 or 6. Multi-platform capability is a plus but not a hard factor. Dictated by reference network.
- Must run on x86-64 hardware. Dictated by reference network and specified by Advanced Micro Devices (AMD, 2013).
- Should have a Spectrum rating higher than 55. Languages below that threshold are considered exotic and may create problems with future upgrades and operational support.
- Documentation and compiler must be open source and freely available. Mandated by expectations.

- Must be a veteran language, initially developed more than 5 years ago. Languages that are younger may not be relevant in the future. Mandated by reference network.
- Must be multipurpose language capable of creating enterprise grade software. Dictated by functionality.

Table 1 shows the languages mapped to the requirements.

Language	Rating	Linux	X86-64	Open Source	Year	Multipurpose
C	100	Yes	Yes	Yes	1972	Yes
Java	98,1	Yes	Yes	Yes	1995	Yes
Python	98,0	Yes	Yes	Yes	1991	Yes
C++	95,9	Yes	Yes	Yes	1985	Yes
R	87,9	Yes	Yes	Yes	1993	No
C#	86,7	No	Yes	No	2001	Yes
Ruby	74,5	Yes	Yes	Yes	1995	Yes
Go	71,9	Yes	Yes	Yes	2012	Yes
Swift	70,1	Yes	Yes	Yes	2014	Yes
Matlab	68,5	Yes	Yes	No	1984	No
Perl	58,5	Yes	Yes	Yes	1987	Yes
Visual Basic	56,8	No	Yes	No	1991	Yes

Table 1 - Languages and hard requirements

These hard requirements leave six candidates remaining, C, C++, Java, Python, Ruby and Perl. Since all six are potentially able to create the Sentinel code another selection process needs to take place, this time specific to language functionality. The functional requirements are:

- Lightweight
- Easy to debug and modify
- Multi-threading/processing capable
- Standard DB interfaces
- Level of multi-threading/processing support
- Language experience of the author

Unless otherwise specified, the applicable values are yes or no.

Lightweight specifies the amount of additional packages and compile environments the programming language needs in order to produce working code.

Easy to debug and modify specifies, how complex of a development environment the language needs to allow the efficient writing and modifying of code. C, C++ and Java are compiled languages, which require the raw code to be processed by a compiler before they produce working code. Python, Ruby and Perl are script languages that compile code at runtime and do not need precompiling. Typically compiled languages are faster than script languages, but the latter are more comfortable to write and modify without an IDE.

Multi-threading/processing capability specifies whether or not it is possible to write code in the language that can run multiple threads or processes.

Standard DB interfaces define if the language has a built in or readily available database connector module for MySQL or LDAP databases.

Level of multi-threading/processing support, specifies how many built-in functions and features the language possesses that allow the creation of efficient code that runs multiple processes or threads. Applicable values are 0, for basic support, 1 for third party modules and 2 for comfortable built-in functionality.

Language experience of the author specifies how much prior experience the author has had with coding in a certain language. Applicable values are 0, for no experience, 1 for simple coding experience, and 2 for considerable coding experience.

Table 2 shows the comparison of the 6 candidates and the result.

Language	Lightweight	Easy to modify	Multi T/P cap	Standard DBC	Level of T/P sup	Experience	**Result**
C	no	no	yes	yes	0	1	**3**
Java	no	no	yes	yes	2	1	**5**
Python	yes	yes	yes	yes	2	1	**7**
C++	no	no	yes	yes	2	0	**4**
Ruby	yes	yes	yes	yes	1	0	**5**
Perl	yes	yes	yes	yes	0	2	**6**

Table 2 - Programming languages "soft" requirements

As a result of this comparison, Python, in its current stable version 3.5.2 was chosen as the language of choice to implement the Sentinel service. Python is a very popular, efficient and powerful scripting language that runs on virtually any platform, comes with a multitude if built-in modules and supports multiple coding paradigms (Python Software Foundation, 2015).

Python also comes with a fully featured multithreading and multiprocessing package, as well as the asynchronous input/output (asyncio) package which all allow for multiple parallel execution of code.

4.2 System environment

As most of the reference network's TFTP servers are running on either Red Hat Enterprise Linux 5.11 or CentOS 6.8, two lab server instances are necessary to replicate a most accurate production environment. Both operating systems come with a Python interpreter pre-installed, but typically a 2.x version, rather than the required 3.5.2.

Fortunately, unlike other programming languages, Python has no problem with having multiple different versions of it installed on a system in parallel, so the 2.x version can be retained for legacy code, while the new application will be developed exclusively with the 3.x version.

The only additional package required at this stage is the MySQL Connector/Python module, which can be retrieved from the official MySQL repository that allows Python code to interact with MySQL databases.

The hardware requirements for the development environment mimic those of production systems, which are 4 gigabytes of memory, 4 CPU cores and 30 gigabytes of disk space. The systems also feature the common two network interface structure of the production environment, one for management purposes and one for production (TFTP) traffic.

4.3 Component structure and data sources

This sub-chapter will define how the service and its components should be constructed, how it will communicate with the outside and what its peers are.

4.3.1 Component structure

As Python supports both traditional procedural coding as well as object oriented programming, a fact that will be put to use in the creation of the code. The service blueprint specifies four basic modules of the Sentinel. The actual code however, will have at least two additional components, the “main” process and the service controller, so in total there are:

- Main process
- Service controller
- Socket Listener
- Client Validator
- Configuration file creator
- Data transporter

The *main process* (MP) is the primary executable of the service, it is in charge of spawning the service controller and, given that the Sentinel is a background service, also of detaching the program from the executing shell. This means the main process must be able to implement some form of deamonization, the act of turning a program into a background process. So the actual runtime of the main process is low. Additionally the main process must also implement functionality to terminate or restart a running service.

The *service controller* (SC) is the result of the daemonized main process. It takes care of spawning and controlling the other modules. Given that several thousand parallel transfer tasks may be running at any given time, it must implement some form of multitasking, either via Python’s multithreading, multiprocessing or asyncio package functionality.

Given the specifications of the TFTP protocol, where the initial client request must always be on UDP port 69 on the server, the *socket listener* (SL) will have to be a separate process that does nothing else but listen to the port, collect the data, pass it on quickly and then listen to the port

again. This is due to the fact that no two processes can register the same port at the same time. In case the optional HTTP handling requirement needs to be implemented, another process will have to be spawned that listens exclusively to port 80 TCP instead.

The *client validator* (CV) must have enough information about the client to verify its identity, potentially requiring multiple database lookups to discern the client's authenticity and entitlement to download the requested file. As it is request specific, it may run either as a separate process spawned by the service controller, or be a subroutine of it, which would simplify caching mechanisms. In either case database queries are an inherent requirement of the module.

The *configuration file creator* (CFC), similar to the CV, must have enough information about the requesting client to generate the specific configuration file for it. Depending on the data source, this may also require multiple database lookups. Also like the CV, it may be a separate process spawned by the SC or a subroutine thereof. In addition, it also requires some form of file system handler that allows it to validate the existence of files or write new ones.

Finally the *data transporter* (DT) handles the communication and packet exchange with the client. Given that data transfers take a variable amount of time, the DT must run in a separate process that has no impact on any of the other components. It requires the ability to register network sockets, must have file and client information and depending on the implementation of the CFC it may also require file handler capabilities. Given the TFTP specification, the DT must always send a valid TFTP response to the client on a proper request, even if that response is an RFC 1350 conforming error message, like access violation.

4.3.2 Data sources

As mentioned in the above segment, the CV and CFC components require certain information from and about the client, not all of which can be present in a simple tftp request. Hence the need to specify certain requirements for the request and the data source to validate the request with.

Client validation would typically be done using a combination of the IP address, MAC address, and provisioned client-class name of the client. As the configuration file name typically matches the client-class name with the extension ".bin" client-class name and configuration file name can be considered the same information.

Configuration file creation on the other hand would be reliant on client-class name, device type, MAC address and specific parameters for the client.

Fortunately the reference network offers several pre-existing data sources that may be utilized by the Sentinel to retrieve client specific information. These sources are:

- DHCP servers
- Equalizer database
- Lease database

- Provisioning database
- Serv Assure Advanced

The *DHCP servers* are the primary data source for client IP and MAC information, due to the DOCSIS specification, they also contain device type information, as the client is required to send its device details encapsulated in the vendor options part in the DHCP discover message, as detailed in RFC 3925 (J. Littlefield, 2014). Unfortunately this device specific information is dropped after processing and the reference network also consists of multiple redundant DHCP servers. So the Sentinel would theoretically have to send queries to several servers before actually getting a response and these would have to be done using DHCP leasequery packets, rather than proper database queries and would only get back part of the information needed.

The *Equalizer* is a high performance cable modem firmware management solution, which is in production use throughout Liberty Global's European operating countries. It contains complete client IP, MAC, device type, configuration file and client specific information. So essentially all the parameters specified above. Unfortunately, the Equalizer does not operate in real time, as cable modems may take between 30 and 60 seconds before becoming responsive to SNMP requests which the Equalizer uses. So there is a distinctive delay between the initial client request and updates to the Equalizer's database.

The *lease database* (LDB) is a custom, near real-time IP, MAC and client-class database, that's fed directly by the DHCP server farms, essentially providing the "as is" lease information of all DHCP servers in a single database. It was created specifically to counter the dilemma of having to query multiple DHCP servers before getting a single answer. There are two server nodes for redundancy and both run a single MySQL server instance.

The *provisioning database* is an LDAP directory server that contains the client MAC and client-class information about every client on the network. It's readily available via multiple nodes in the reference network as it is also being used by the DHCP server farm.

Serve Assure Advanced is the reference network's client statistics and monitoring data gathering solution. It contains MAC, IP and device type information. But similar to the Equalizer database, it does not operate in real-time, taking up to 15 minutes to identify new clients.

Given the limitations presented above and the need for real-time data, the LDB appears to be the most useful existing candidate for the Sentinel to interact with. Its possible extension with device type information will be part of the implementation chapter.

If special, individual client specific settings are required to be made persistent and additional database will have to be created, which stores this kind of data. This database could either work in conjunction with the existing LDB solution or be running locally on the Sentinel nodes.

Another problem is the fact that CMTS' with dynamic shared secret and tftp-enforce option enabled obscure the client's TFTP request by exchanging the client's IP with the CMTS' loopback interface IP, meaning that all downloads for that CMTS appear to be coming from the same client, when viewed from the TFTP server perspective. As client source IP information is critical for client validation, this issue needs to be addressed.

There are two ways to work around the obscuring. One of them is to utilize the network address encapsulation in the TFTP request as specified in the hardware and network address options for TFTP RFC draft (Zeng, Evans, 2004). The other would be to modify the DHCP server to encapsulate the actual client IP in the TFTP filename. Both modifications will be examined in detail in the implementation chapter.

4.4 Blueprint mapping

By merging the initial blueprint with the components explained above the system's design can be updated to serve as an implementation template. A graphical representation of the template with the CV and CFC running as part of the DT worker processes can be seen in figure 16.

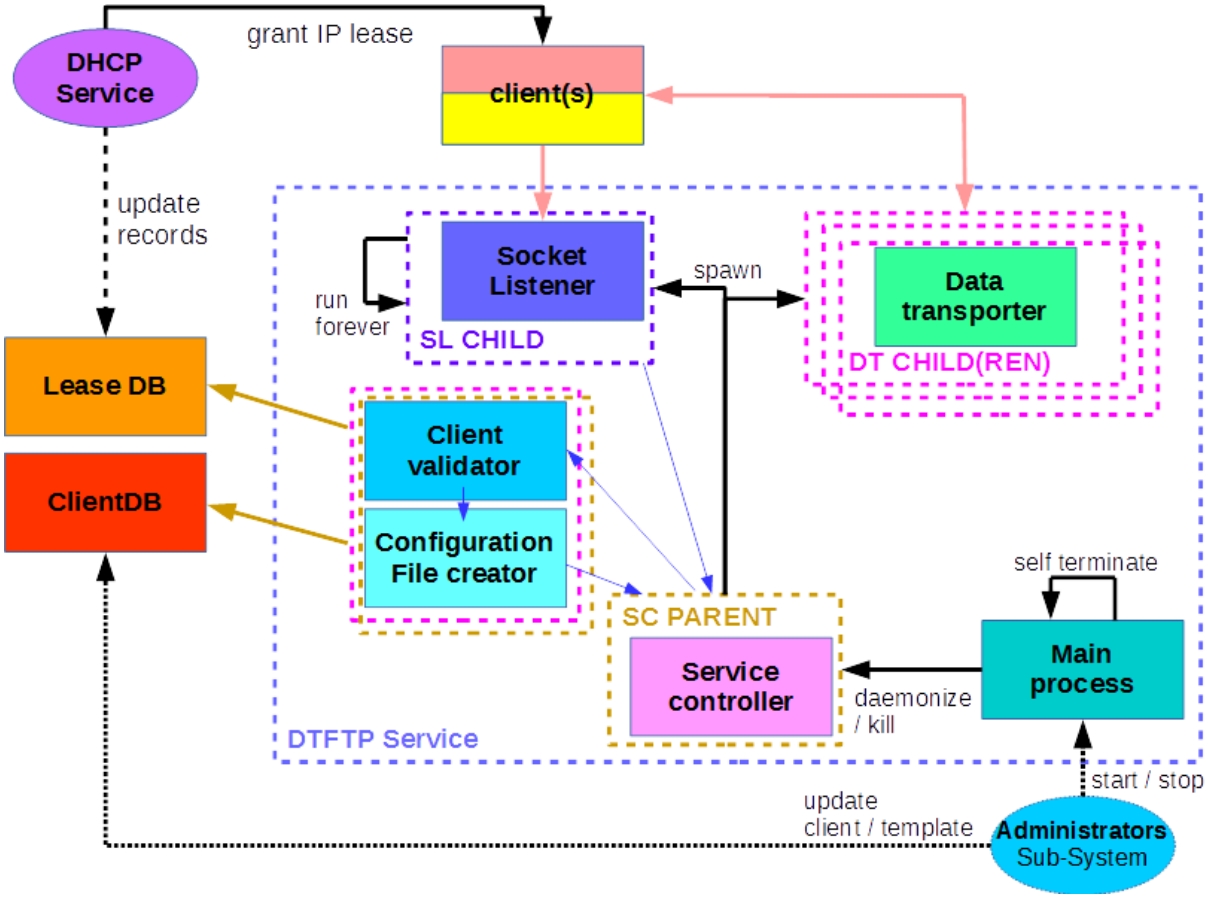

Figure 16 - Service Implementation Template

While perhaps not 100% accurate at this point, this implementation template will serve as a guideline for the prototype construction. Deviations may occur in the way the worker child processes are set up and the form and function of the LocalDB, given that high data throughput and requests per second are a key performance indicator of the service.

4.5 Systems interaction

The service implementation template from above tells a lot about how the service is supposed to be constructed internally, but provides no details about how data is passed inside the service and how it interacts with external components. As having an understanding how the Sentinel interacts with other system components is crucial, the service flow steps will be explained again and deviations from the standard TFTP process highlighted.

The trigger event for the service is a cable modem that boots up. This CM initiates its DHCP cycle which is relayed by the CMTS it's attached to. The DHCP server receives the relayed DHCP discover message and does an LDAP lookup with the client's MAC address. If a valid LDAP response is received, the DHCP continues with the DHCP cycle and grants a lease, otherwise the DHCP packet is dropped and the client starts its DHCP cycle again. Additionally the DHCP server provides the client with a TFTP server address and file name and updates the Lease Database with the new client IP.

This stage of the DHCP process could possibly be modified to include additional information in the TFTP file name, like actual client IP and modem type, as from the client's perspective and the TFTP protocol specifications, filenames can be a string of any kind of ASCII characters with up to 255 bytes length.

As the CMTS is relaying the DHCP packet it can perform certain modifications to the packet, one of which is the exchange of the TFTP server address, so the actual TFTP read request from the client is going through the CMTS rather than the actual TFTP server.

Once the DHCP cycle is complete the client sends a TFTP RRQ message to the TFTP server address that is now the CMTS acting as proxy. The CMTS itself forwards the request to the actual TFTP server, but can attach certain options to the RRQ, like the network address, which represents the actual client IP.

The Sentinel receives the RRQ and does some initial parsing on it, to make sure it conforms to the TFTP specification, if not, the request is dropped and an error message sent to the client.

Then, unlike a regular TFTP server, the Sentinel performs two steps of validation, first the client's IP and the filename, which were included in the RRQ, are extracted and queried against the Lease Database. Should validation of either fail, the packet is dropped and an error message is sent to the client. If successful the Sentinel checks whether or not the requested file is present locally. If the local copy exists, it then starts transferring the file to the client. If it doesn't the Client Database is queried for information about the client and if a valid response is received, the Sentinel creates the file from the template and then transfers it to the client. An invalid Client Database response will also cause the Sentinel to drop the packets and send an error message to the client.

If the transfer is successful, the client parses the file and is online. A flowchart representation of the entire systems interaction can be seen in figure 17.

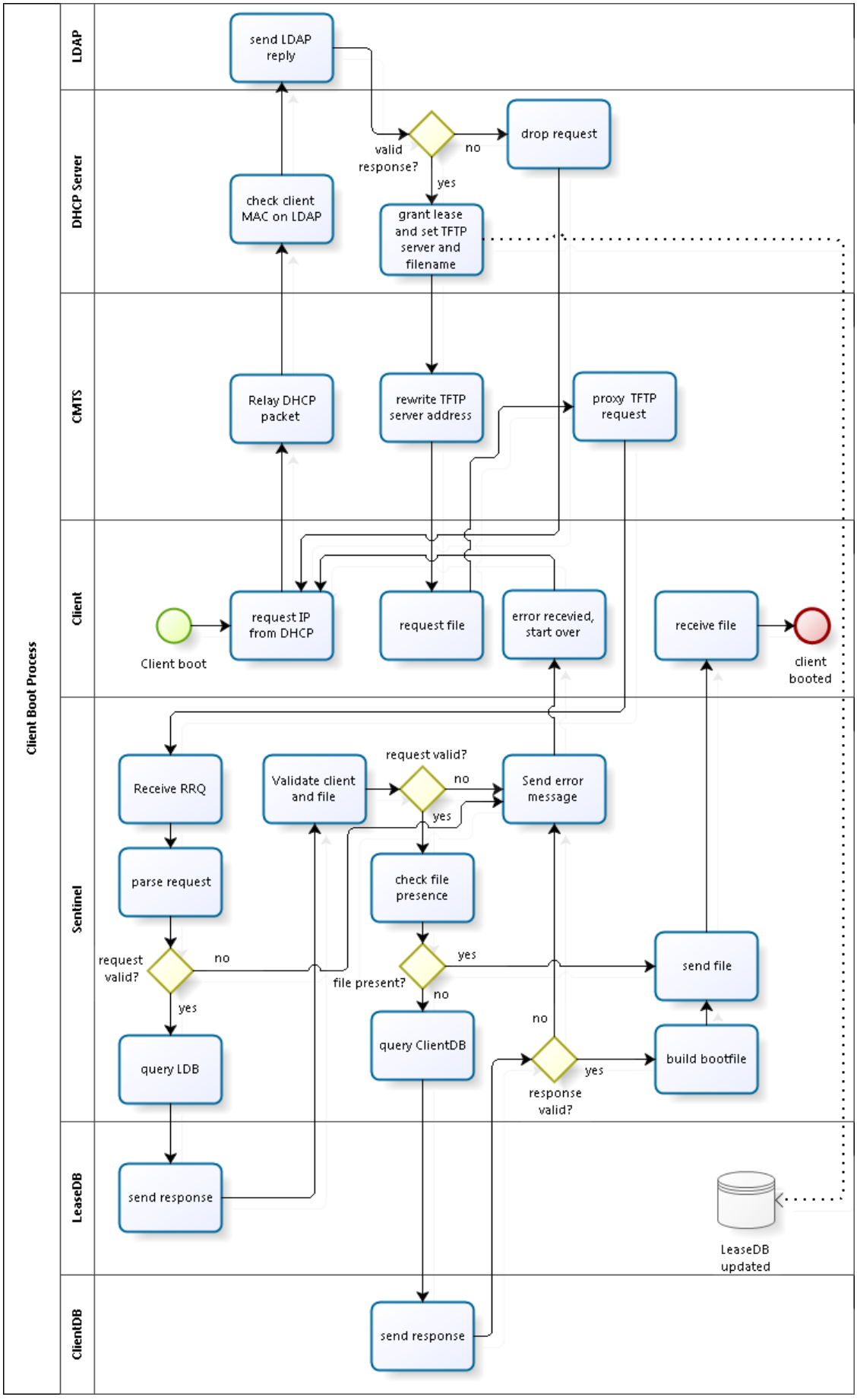

Figure 17 - Systems interaction and service flows

4.6 The Client Database structure

The Client Database (CDB) construct serves two purposes, the first being a standardized storage area for client specific configuration data, while the second function is to be able to store modem configuration templates. The following design template specifies a basic structure that contains all the necessary information for easy and efficient retrieval, but does not dictate the actual form the two components of the client database need to take.

4.6.1 Client storage

The client storage should be a flexible database construct that allows specific client configuration data to be stored. An entry in the client storage is an optional component, as the Sentinel can glean generic cable modem data from the bootfile (e.g. speed, voice capability, Wi-Fi capability and network mode) and modem type (vendor and device specific parameters). It would also serve as a way to override parameters usually fixed by the client-class name.

Client specific configuration data items could be:

- Individual IP/port access lists
- Static IP addresses
- Custom SNMP communities
- VPN configuration
- Client specific credentials

As these configuration items are both optional and very heterogeneous in their format, two separate tables are recommended. One containing the client MAC and an ID (table client). The other table should contain a reference to the client ID, the configuration item name, the configuration item value and an ID (table config). This way an arbitrary number of configuration items can be stored for any given client, without having to redesign the database layout every time a new configuration item needs to be added. The database configuration and tables with sample data be seen in table 3.

Table client	
ID	mac
1	aa:ff:cc:bb:dd:00
2	aa:ff:cc:bb:dd:01
3	aa:ff:cc:bb:dd:02
4	aa:ff:cc:bb:dd:03
5	aa:ff:cc:bb:dd:04
6	aa:ff:cc:bb:dd:05

Table config			
ID	clientID	cname	cvalue
1	1	smtpblock	1
2	1	snmpcommunity	jagf34zrx
3	1	staticip	89.99.12.17
4	2	vpnvlan	1070
5	1	webpass	Lk7pQ3f
6	3	smtpblock	0

Table 3 - Client storage sample data

4.6.2 Modem template storage

The modem template storage contains the actual settings that will be encoded into the cable modem configuration file. This includes generic parameters and specific ones. Depending on the function of the value either the entire content is saved or just parts of it. For example, it would not make much sense to hardcode every individual speed combination. Instead the speed should be calculated by the Sentinel's CFC module and then appended to the value in the template storage to complete it. On the other hand, having the entire set of port block rules for the SMTP protocol in one set does make sense, since setup and values may be completely different for another type of protocol block. The complete entity relationship model for both databases can be seen in figure 18.

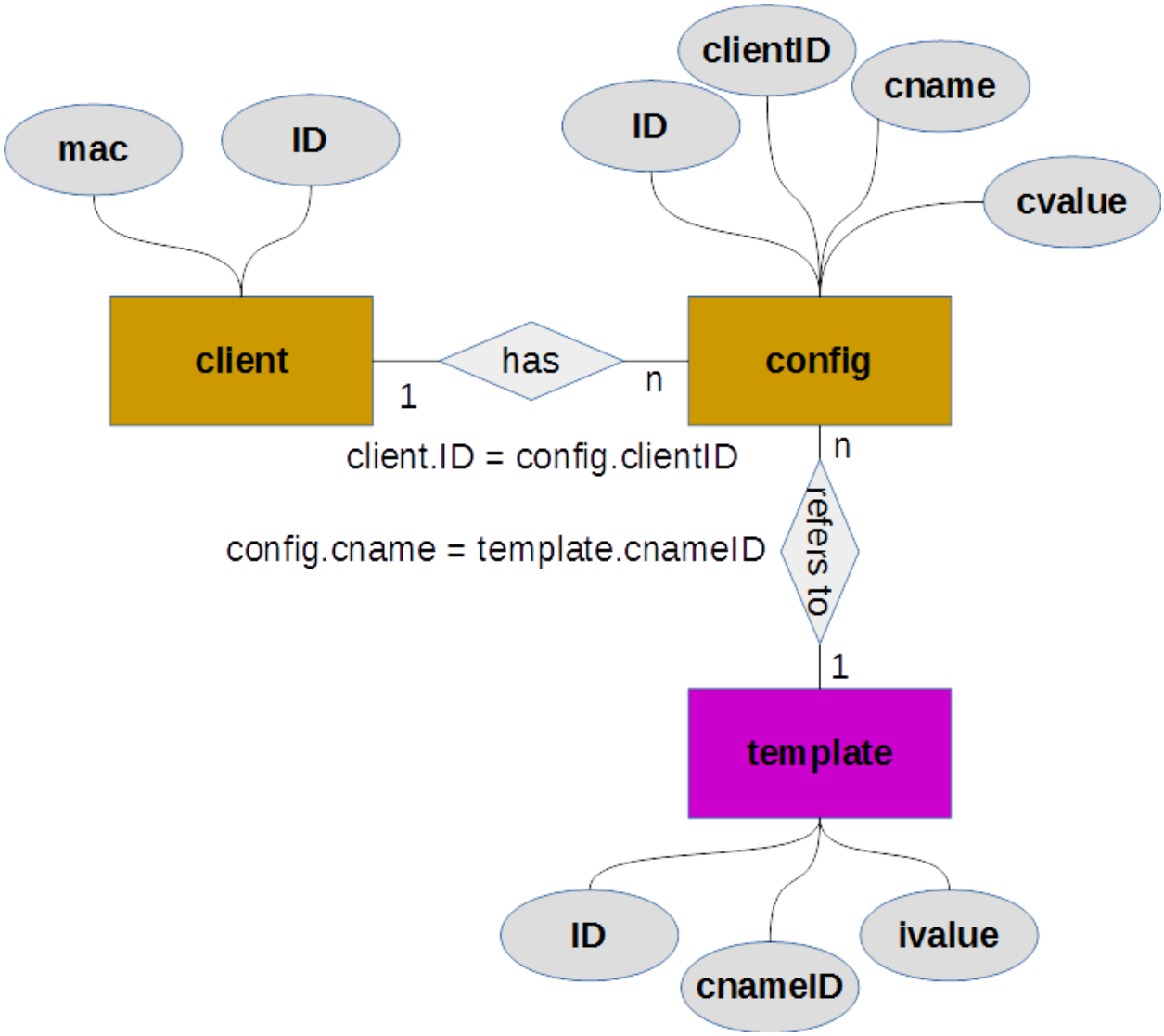

Figure 18 – Client database entity relationship model

5 BUILD AND IMPLEMENT THE PROTOTYPE

With the design complete, work can now finally start on the implementation of the prototype service. Due to the complexity of the project and the fact that development has to start from scratch, the need to split it into several smaller but incremental tasks arises. The following development tasks have been identified and will be executed during this chapter:

- Object structure
- Build the development environment
- Stage 0, baseline TFTP protocol implementation
- Functional verification and test tool development
- Stage 1, TFTP option parsing
- Functional verification and testing
- Stage 2, implementation of client source verification
- Functional verification and test tool expansion
- Stage 3, on-the-fly configuration file encoding
- Final functional verification

Even though TFTP is a very old protocol, there are only a very limited number of test tools available and none of the sophisticated ones are available free of charge. Due to this circumstance, part of the development effort will include the creation of a test suite that is tailored to the Sentinel's unique capabilities and the ability to report proper performance data even outside of a production environment.

5.1 Object structure

Making use of Python's object oriented coding capabilities, the Sentinels code will rely heavily on class based construction. Given the design template from the previous chapter, the following classes have been deemed necessary for implementation:

- Main process
- Service controller
- Socket listener
- Client validator
- Configuration file creator

- Data transporter

As stated in the design phase, the *main process* is implemented in the aptly named *sentinel.py* file and only has two tasks, controlling the operating state of the service controller process and turning the service into a background process by terminating itself after spawning the service controller. This Linux daemon functionality is achieved by implementing the generic Linux daemon base class (LDBC) for Python 3.x (Marechal, Anonymous, 2009) which launches the Sentinels service controller code as a separate thread before terminating the parent thread. Minor modifications to the code were made to allow it to terminate two related processes, rather than just one. The entire main process code can be seen in listing 2.

```
if __name__ == '__main__':

  sentinel_proc = sentinel('/tmp/sentinel.pid')
  if len(sys.argv) == 2:
    if 'start' == sys.argv[1]:
      sentinel_proc.start()
    elif 'stop' == sys.argv[1]:
      sentinel_proc.stop()
    elif 'restart' == sys.argv[1]:
      sentinel_proc.restart()
    else:
      print("Unknown command")
      sys.exit(2)
    sys.exit(0)
  else:
    print("usage: %s start|stop|restart" % sys.argv[0])
    sys.exit(2)
```

Listing 2 – main process code

The LDBC also takes care of proper housekeeping for the daemonized process, like detaching it from the executing shell, providing functions to start, stop and restart the service gracefully, setting proper user IDs and root directory, as well as redirecting the OS standard input/output queues. This proper daemon behavior is also described in Python Enhancement Proposal (PEP) 3143 (Python Software Foundation, 2009).

As no changes are expected to the main class after the initial configuration, the *service controller* code is also added to the sentinel.py file, albeit using the class tag daemon instead, since it is mandatory for code using the LDBC to override the daemon class' "run" subroutine. The SC has multiple tasks, the first of which is to set up a proper set of environment variables, like database connection details, service port and root TFTP directory path. Setting up data constructs to keep track of client sessions and active processes, as well as reporting to the OS via syslog that the service has started also falls to the SC's setup tasks. Finally it's also in charge of instantiating and launching the socket listener class as a separate process, by providing it with a data structure to put TFTP packets in. Additionally it is provided a port number and network interface to which to listen to, before entering into an infinite loop to process TFTP packets.

The socket listener is the only other permanently running process of the Sentinel. It starts off by writing its process ID into a file, so it can also be identified and stopped by the LDBC. Afterwards it binds a network socket as specified by port and network interface handed to it by

the service controller process, before also entering an infinite loop to receive data packets on the bound network socket. As listening to the socket is a blocking operation that suspends all tasks of a thread, the SL is launched in a separate process. So, for performance reasons, all the loop does is listen to the socket and put the received data into the data construct provided by the service controller process, before listening to the socket again. As the TFTP protocol only uses port 69 for the initial request and not the actual data transfer, this separation of tasks has been deemed necessary for optimal system performance.

Detailed CV, CFC and DT descriptions will be provided in the respective sub-chapters below that require their functionality.

5.2 Build development environment

In order to develop and test code a desktop system, a pre-existing LeaseDB installation as well as a set of three container based servers running CentOS 6.8 and Python 3.5.2 were provided, designated viezcnrat97-99. All three servers and the LeaseDB share a common external interface that connects to the lab environment's firewall and allows traffic from dedicated lab CMTS' to pass through. The desktop and the servers also have direct access to a central pre-existing GIT repository for source code download and maintenance. Server 99 runs both a DHCP and a TFTP server similar to a production environment, while 97 and 98 currently have only the basic OS installed and are intended for future use. Figure 19 shows a schematic of the development environment.

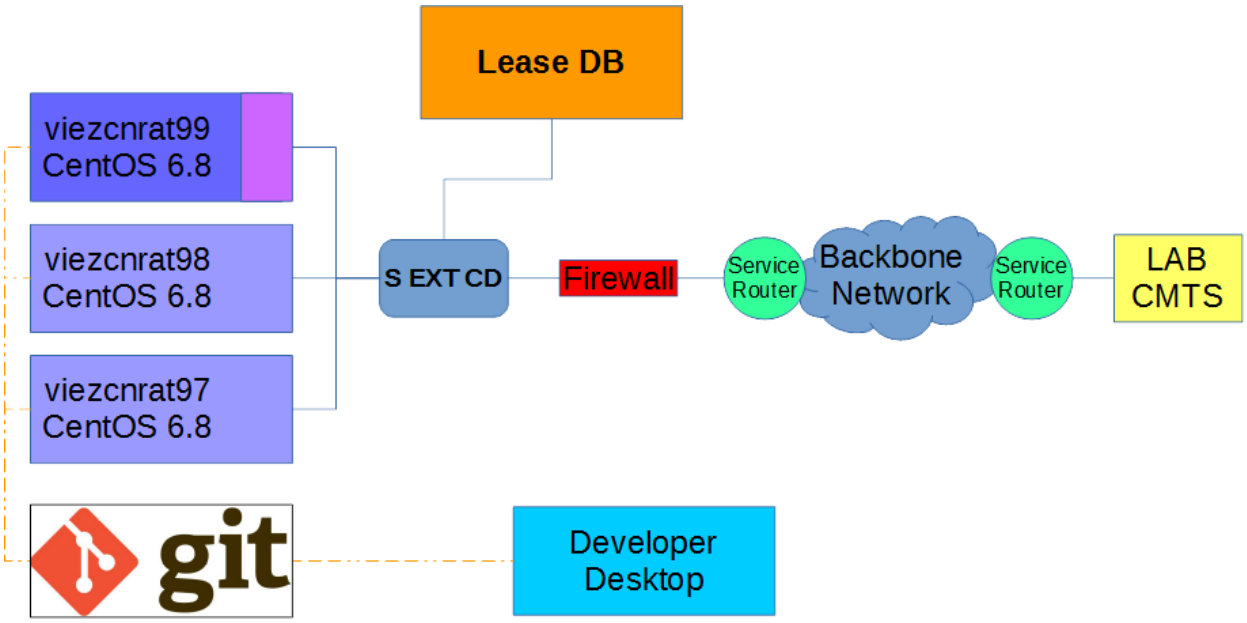

Figure 19 - Development and testing environment

5.3 Stage 0, baseline TFTP functionality

As there is no Python standard TFTP server implementation, unlike server implementations for other popular protocols, the development of the Sentinel has to start essentially from scratch. The Stage 0 development effort should result in a working prototype that provides basic TFTP service on par with the legacy solution.

5.3.1 Code development

After implementing the code as stated in 5.1, the Sentinel has two processes running, the socket listener which listens to port 69 and encapsulates requests into a data structure, and the service controller which runs in background and waits for packets from the socket listener. Passing data reliably between separate processes can be tricky, fortunately Python's multiprocessing library provides so called "queues", which are essentially arrays that can contain any kind of data. In the Sentinel's case, this data is a tuple of values, one containing source address and port of the TFTP request, the other the actual binary data.

The service controller's infinite loop constantly checks the queue for new data packets, once a packet is received it instantiates a new class generically named process_send_data (PSD) and spawns it into a separate new process, along with the data packet from the queue and the tftp root directory variable. In addition to spawning the process it also keeps track of its status by writing both the source address and the process object into a data structure. The first is to prevent the client from causing the spawning of multiple processes, the latter one to make sure that the child processes are properly joined once they terminate.

It is important to note that PSD is actually a separate process, not a thread, since Python's global interpreter lock (GIL), prevents threads from using more than one CPU of the host system. This limitation does not apply for processes created with the multiprocessing module, as described in PEP-371 (Python Software Foundation, 2008).

The PSD encapsulates some of the functionality of the Data Transporter module. Once spawned it will parse the data packet and decides, based on the TFTP specification, whether or not the request is valid. If valid and the transfer options are supported, it checks whether or not the file is present on the file system. Once all prerequisites are met it binds a network socket, reads the first 512 bytes of the file and sends them to the client using the address and port tuple provided by the service controller. As per RFC 1350, the client will acknowledge every data packet sent until all bytes are transferred, so the PSD constantly needs to switch between listening to and sending data on the network socket. Like request packets, ACK packets also need to be parsed and their content examined before sending the next data packet.

Should an error occur, as in the request being invalid or the file not being present, the server must still send a properly formatted TFTP error message to the client.

Once the request packet has been processed the PSD terminates and the service controller joins the process to prevent it from turning into a so called "zombie".

A flowchart representation of the Stage 0 Sentinel can be seen in figure 20.

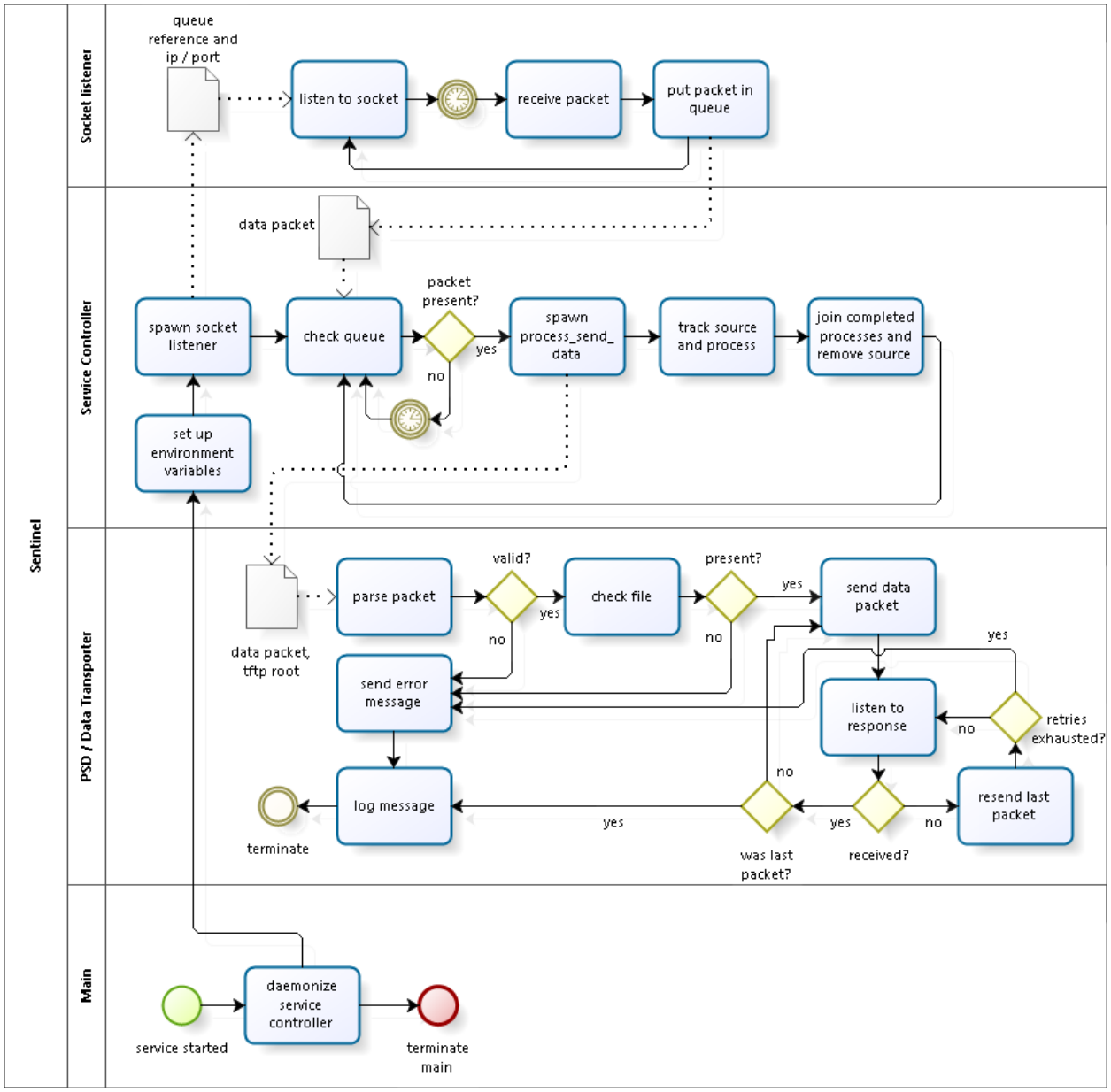

Figure 20 - Sentinel Stage 0 process flow

5.3.2 Functional verification and test tool development

During development it became quite obvious that doing functional verification with just the stock tftp client program that comes with any Red Hat or CentOS installation was not going to be enough. While it does give basic RFC 1350 functionality like changing the transfer type, it does not allow any of the functions described in the extended RFCs, nor does it allow for any kind of performance and stress testing.

With that in mind, development on a tool that would complement the Sentinel was started. Given the inherent stress test functionality it would be able to do the name “Jackhammer” was deemed appropriate for the tool.

As both programs essentially deal with the same kind of data, just from different perspectives, large parts of the Sentinel’s code could be reused to create the Jackhammer.

Modifications included the removal of the packet parsing from the PSD class, which was then encapsulated in a new class called “Packet Parser” (PP) that allows the proper handling of any of the 5 kinds of TFTP packets. The PP is used as a generic way to analyze TFTP packets, perform validity checks on them and write the appropriate parameters into easy to retrieve variables. It’s worth noting that the packet parser does not check if a file is present or not, it just extracts and validates data contained in the packet. The simplified process flow of the Packet Parser can be seen in Figure 21.

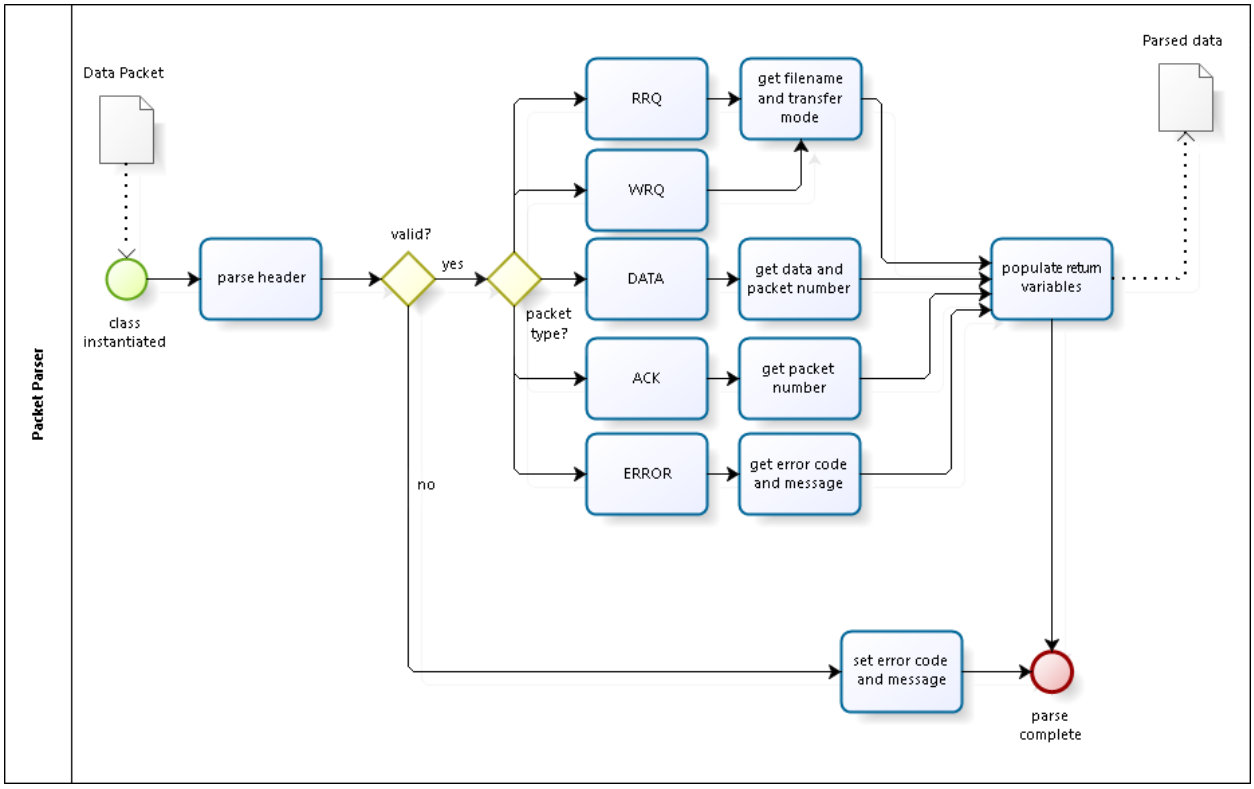

Figure 21 - Packet Parser process flow

Other modifications included the creation of a mirror image of the “Process Send Data” class, aptly named “Process Receive Data” (PRD), as well as the removal of the daemon class from the main process, as Jackhammer doesn’t need to operate as a background process.

The initial version of Jackhammer supports two operating modes, these are:

- RFC conformity checking
- Performance testing

RFC conformity checking sends a series of TFTP packets, covering the entire RFC 1350 capability spectrum, towards a specified server and reports back the result. Given that the Sentinel was coded specifically to support only TFTP read requests in binary mode, it is expected that other tests fail this check.

Performance testing mode runs a fixed amount of tftp downloads in rapid succession against the target TFTP server. This operating mode reuses the Sentinel’s multiprocessing component and reports back the number of requests sent, peak parallel processes and amount of failed transfers. The result of this mode should give an idea of the performance of the target server,

with the option of running multiple Jackhammers from different source servers to one Sentinel target server.

The performance results of the test suite against the legacy TFTP server can be seen in listing 3, verifying the Jackhammer's usefulness as a test tool. Server 99 served as a base for the legacy HPA TFTP server, while Jackhammer was operating from 98.

```
Jackhammer TFTP RFC conformity check!

RFC 1350, RRQ, OCTET: OK
RFC 1350, RRQ, NETASCII: OK
RFC 1350, WRQ, OCTET: OK
RFC 1350, WRQ, NETASCII: OK
RFC 1350, WRQ, MAIL: FAIL, Unknown mode
```

Listing 3 – Jackhammer Stage 0 test suite vs. HPA TFTP server

After running the Jackhammer against the legacy TFTP to get some numbers for comparison, the test suites were run again, this time against a Sentinel implementation, with Sentinel replacing the HPA TFTP on server 99. The results can be seen in listing 4.

```
Jackhammer TFTP RFC conformity check:

RFC 1350, RRQ, OCTET: OK
RFC 1350, RRQ, NETASCII: FAIL, Unsupported transfer mode, use binary/octet!
RFC 1350, WRQ, OCTET: FAIL, Unsupported request type!
RFC 1350, WRQ, NETASCII: FAIL, Unsupported request type!
RFC 1350, WRQ, MAIL: FAIL, Unsupported request type!
```

Listing 4 – Jackhammer Stage 0 test suite vs. Sentinel Stage 0

As Sentinel does not support write requests nor any transfer mode other than octet/binary, the outcome of the test matches the expectations.

5.4 Stage 1, option parsing

The objective of this development stage is the implementation of the extended TFTP RFC's for option processing (RFC 2347), the block size option (RFC 2348), the timeout and transfer size options (RFC 2349) and the window size option (RFC 7440), as well as the RFC draft for inclusion of the network address option. For validation and verification reasons, these modifications have to be applied to both Sentinel and Jackhammer.

5.4.1 Code development

TFTP option parsing requires the introduction of one new packet type and one error message type not present in RFC 1350, the packet being OACK, for option acknowledge and code 8 for the error message type, indicating invalid or unsupported options.

Using the new packet type requires a change to the packet parser class, which allows the extraction of option data from the initial TFTP packet as well as the parsing of a message type 6 (OACK) packets.

A visual representation of the enhanced packet parser class can be seen in figure 22.

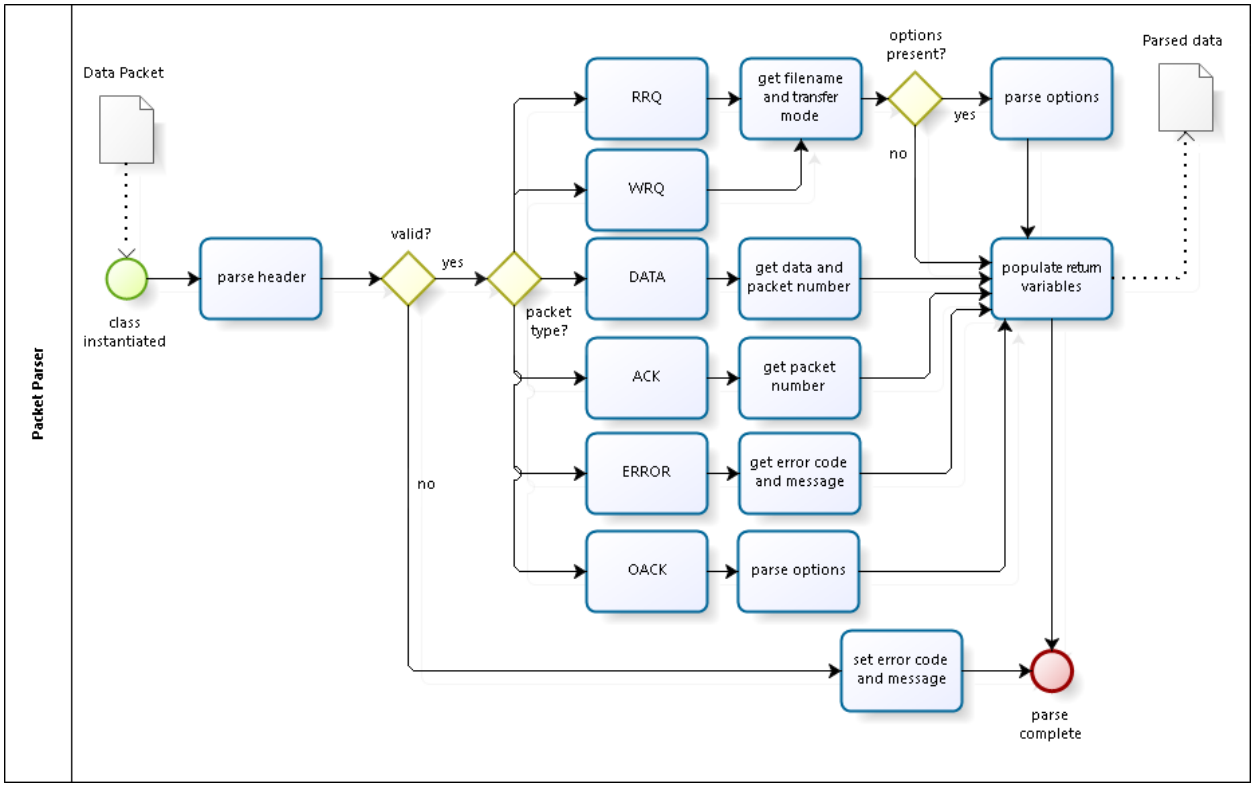

Figure 22 – Enhanced packet parser with option handling

In order to make use of the new option parsing capabilities of the PP, the PSD also had to be rewritten to both understand and reply to TFTP requests including options. The rest of the Sentinel code was however unaffected.

While the older TFTP RFC's are pretty easy and straightforward to implement as they only modify existing TFTP behavior, RFC 7440 and network address encapsulation on the other hand require drastic behavior changes in the PSD and also the PRD.

General *option processing* requires the sender to attach one or more option name and value to the initial read or write request after the closing zero byte packet behind the transfer mode setting. Any non RFC 2347 compliant TFTP server would stop parsing the request after the transfer mode and would just reply with the first data packet as usual. Servers that are RFC 2347 compliant however, parse the options and send an option acknowledge packet back to the requester, containing the same sequence of options that the client sent, in turn, the client acknowledges this packet with packet number 0, rather than the usual packet number 1. Afterwards the default TFTP packet transfer commences.

The *blocksize option* allows the client to request bigger packets than the default 512 bytes to be transferred per data block. Due to the maximum transfer unit (MTU) size limit of Ethernet with 1500 bytes payload and the inability of UDP to support packet fragmentation, the effective maximum is 1456 or 1476 bytes for TFTP packets using IPv6 and IPv4 respectively. As only about 1/3 of the packets have to be transferred, this typically means that the transfer completes faster and the data throughput is higher. To facilitate this option, the PSD and PRD have a

global variable that defines how many bytes from the file are read at a time and it defaults to 512, but can be updated after parsing the initial request.

Using the *timeout option*, an integer value of seconds can be sent to the server, to change the default time the server waits for a packet from the client, before retransmitting the last packet. The PSD defaults to a 1 second timeout, which can be modified through the timeout option to a maximum of 2 seconds. This limit was imposed to prevent potential attackers from exploiting this option by setting it to the maximum of 255 seconds and overload the server with multiple stale sessions.

The *transfer size option* allows the client to inform the server about the file size it intends to transmit, for write requests, or lets the client query the file size on the server before actually downloading it. This option is purely cosmetic and doesn't influence server behavior. However, unlike the timeout option, the value contained in the server response can actually differ than the one in the client request.

Utilization of the *windowsize option* allows the client to change the fundamental TFTP feature of having to acknowledge every received data packet, before the next one is sent. The client can specify a value between 1 and 65535, to define how many data packets can pass before the receiver needs to send an acknowledge packet. As the number of packets required to transfer data is reduced, this option, similar to the blocksize option reduces transfer times and increases data throughput. It is however also affected by diminishing returns (increasing blocksize beyond 32 do not improve throughput) and very sensitive to quality of the network. The PSD needed to be modified to read and hold data equal to the blocksize multiplied by the windowsize, as a retransmit from the client could happen at any of the packets within the window, and the next window had to be moved up to the last ACK received. This was implemented by using a byte array with elements equal to the number of the windowsize, which is dynamically assigned new values as appropriate. The PRD on the other hand had to be modified to send ACK packets only once the packet equaling the value of the windowsize option had been received, or if a timeout happened, the ACK would equal the last packet received.

Compared to the implementation of the windowsize option, adding the *network address support*, was simple. The network address option field allows a relay agent to encapsulate the client IP address in the TFTP request packet, as the source IP of the TFTP packet is changed to the CMTS the cable modem is connected to, if TFTP relaying is enforced. The "netaddr" keyword is reserved for this option and is attached by the CMTS, rather than the client, this also means that the netaddr option must not be present in the option acknowledge packet the server sends back to the client, because the client does not know that its being relayed. The PSD was modified so it would be able to extract the actual client IP from the packet, but would also not put the netaddr option back into the OACK. In order to simulate this option without a CMTS present, the PRD was modified to attach the option itself, encapsulating the client IP in the netaddr field, just like a CMTS usually would, but still flagging it as an invalid response if the server returned the netaddr option in the reply. The modified PSD process can be seen in figure 23.

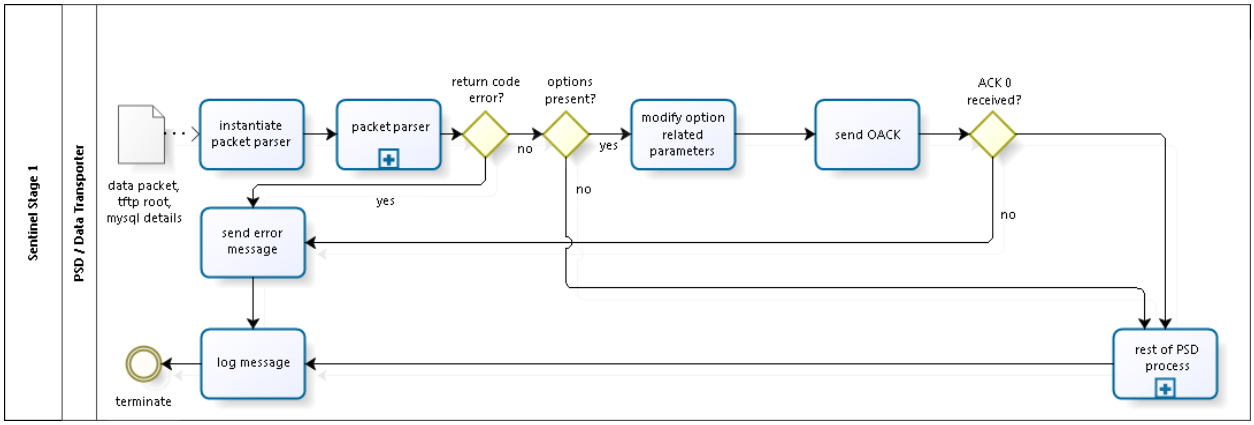

Figure 23 - Stage 1 modified PSD process

5.4.2 Functional verification and testing

After upgrading the Jackhammer program to also support the defined option sets, functional verification was repeated, again using the HPA TFTP as a static reference. The result of the checks versus the Sentinel can be seen in listing 5.

```
Jackhammer TFTP RFC conformity check:

RFC 1350, RRQ, OCTET: OK
RFC 1350, RRQ, NETASCII: FAIL, Unsupported transfer mode, use binary/octet!
RFC 1350, WRQ, OCTET: FAIL, Unsupported request type!
RFC 1350, WRQ, NETASCII: FAIL, Unsupported request type!
RFC 1350, WRQ, MAIL: FAIL, Unsupported request type!
RFC 2347, RRQ, OCTET, tsize: OK
RFC 2347, RRQ, OCTET, blksize: OK
RFC 2347, RRQ, OCTET, timeout: OK
RFC 7440, RRQ, OCTET, windowsize: OK
RFC DRAFT, RRQ, OCTET, netaddr: OK
RFC ALL, RRQ, OCTET, all options: OK
```

Listing 5 – Jackhammer Stage 1 test suite vs. Sentinel Stage 1

As the HPA TFTP only supports a more limited function set, several tests failed as can be seen in listing 6.

```
Jackhammer TFTP RFC conformity check!

RFC 1350, RRQ, OCTET: OK
RFC 1350, RRQ, NETASCII: OK
RFC 1350, WRQ, OCTET: OK
RFC 1350, WRQ, NETASCII: OK
RFC 1350, WRQ, MAIL: FAIL, Unknown mode
RFC 2347, RRQ, OCTET, tsize: OK
RFC 2347, RRQ, OCTET, blksize: OK
RFC 2347, RRQ, OCTET, timeout: OK
RFC 7440, RRQ, OCTET, windowsize: FAIL, timeout!
RFC DRAFT, RRQ, OCTET, netaddr: OK
RFC ALL, RRQ, OCTET, all options: FAIL, timeout!
```

Listing 6 – Jackhammer Stage 1 test suite vs. HPA TFTP

It's important to note that the RFC DRAFT check, reports back OK, even though the server does not understand the option and discards it. This is however not noticeable on the response packet, as the netaddr option must not be sent back to the client.

5.5 Stage 2, client verification

With the entire spectrum of regular TFTP functionality covered in the previous stage, stage 2 can focus on the first piece of advanced functionality, the client verification. The idea behind client verification is to build an enhancement to the TFTP server module that can somehow identify the requester and check whether or not he is qualified to download the requested file from the server. As the TFTP protocol itself does not support client authentication, custom code has to be developed to achieve this functionality, while maintaining RFC 1350 conformity.

5.5.1 Code development

In order to reduce workload on the service controller process, it was decided to put the required modifications exclusively in the individual PSD processes. This required additional information to be passed to the PSD process during initialization, the MySQL server details and credentials.

The stage 1 sentinel also took use of the new packet parser class for packet validation and data extraction, but apart from the extra client validation large parts of the process remained unchanged.

As usual the service controller instantiates a PSD process and passes the MySQL details for the LeaseDB along with the data packet and tftp root directory. The PSD then instantiates a packet parser class, which takes care of packet validation, data extraction, and also checks if the requested file is present on the server. The PSD then only needs to read the error code value from the packet parser object to see if it can immediately send an error message or if it needs to keep processing the request. Should the latter be the case, a MySQL query containing the client's IP address is sent to the LeaseDB, requesting MAC address, client-class and lease start time information.

Once the reply from the DB is received the PSD compares the result of the query with the current time. Should there be no response or if the start time of the client's lease was longer than 60 seconds ago, an access violation error is sent to the client and the session terminated. If the response is valid however and the time difference lower than 60 seconds, the PSD compares the requested file with either the client-class of the requester, if it's a cable modem or the MAC address of the client, if that client happens to be an MTA. If successful, the PSD goes on with the default packet transmission, with the exception that the packet parsing again is handed off to the packet parser class. In case the file name or MAC cannot be validated an access violation is sent back to the requester. The process flow of the modified PSD class can be seen in figure 24.

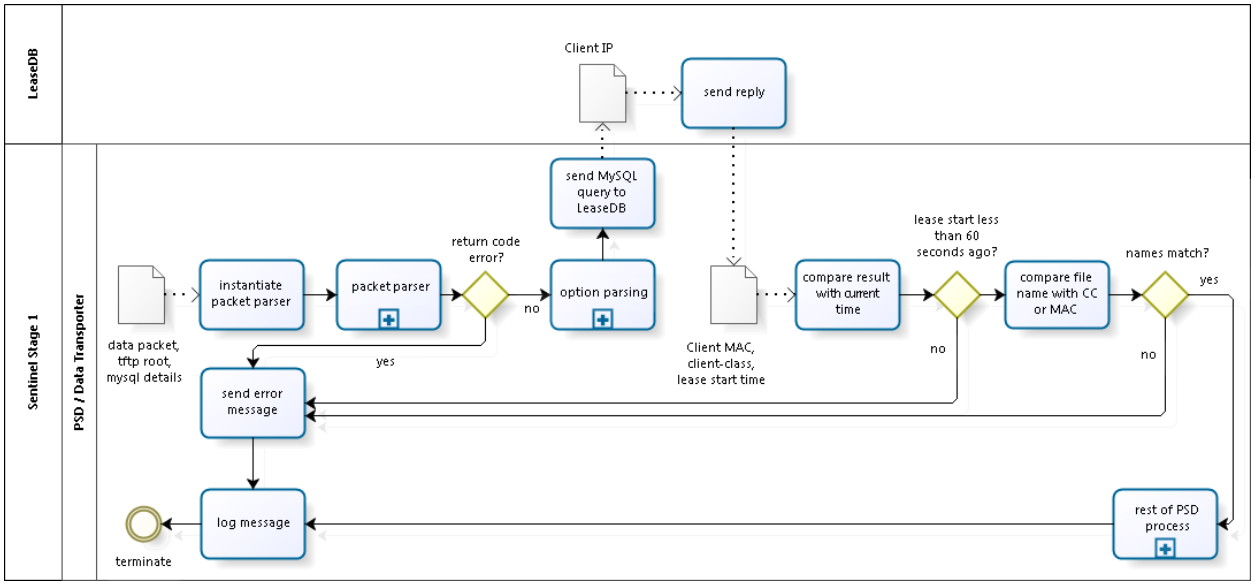

Figure 24 - Sentinel Stage 2, PSD modifications

5.5.2 Functional verification and test tool expansion

Unlike previous verification stages, the Jackhammer program did not require any structural modifications to support the new functionality. To test the implementation, a configuration file download was attempted, once without the client being present in the LeaseDB (expected result, download fails), another with the client being present (expected result, download successful) and the final one with the client being present, but its timestamp outside the valid download window. To facilitate this, the entries were manually modified on the LeaseDB via MySQL statements. The results of the test suite can be seen in listing 7.

```
Download attempt without existing client entry on LeaseDB:

Jackhammer TFTP single download test:

RRQ, OCTET, filename: 6y3fp7_6y4so25_v4.bin,size: 0 bytes
Return code: FAIL, Client verification failed, download denied!
--------------------------------------------------------------------------------

Download attempt with client entry in LeaseDB:
Create sample client entry in LeaseDB.
mysql> insert into leases_v4
values(inet_aton('172.28.245.22'),'00:26:5e:13:89:26','00:ff:ff:00:ff:ff','no
sid','6y3fp7_6y4so25_v4',inet_aton('195.34.134.215'),1479476800,NOW());

Jackhammer TFTP single download test:

Filename: 6y3fp7 6y4so25 v4.bin, size: 8411 bytes
Return code: OK
--------------------------------------------------------------------------------

Download attempt with client entry but expired download window in LeaseDB:
Modify LeaseDB entry so the leasetime is outdated.
mysql> update leases_v4 set ts=(NOW()-1000) where ip=(
inet_aton('172.28.245.22'));

Jackhammer TFTP single download test:
```

```
Filename: 6y3fp7_6y4so25_v4.bin , size: 0 bytes
Return code: FAIL, Valid transfer time window exceeded, download denied!
```

Listing 7 – Jackhammer Stage 2 test suite vs. Sentinel Stage 2

Since the HPA TFTP server does not support the stage 2 functionality, but any TFTP client can mimic cable modem download behavior, the CentOS/RHEL standard tftp client was also used against the stage 2 Sentinel, using the same client entries as above. The results are visible in listing 8.

```
Download attempt without existing client entry on LeaseDB:

tftp> binary
tftp> verbose
Verbose mode on.

tftp> get 6y3fp7_6y4so25_v4.bin
getting from 172.28.245.21:6y3fp7_6y4so25_v4.bin to 6y3fp7_6y4so25_v4.bin
[octet]
Error code 2: Client verification failed, download denied!
---------------------------------------------------------------------------

Download attempt with client entry in LeaseDB:
Create sample client entry in LeaseDB
mysql> insert into leases_v4
values(inet_aton('172.28.245.22'),'00:26:5e:13:89:26','00:ff:ff:00:ff:ff','no
sid','6y3fp7_6y4so25_v4',inet_aton('195.34.134.215'),1479476800,NOW());

tftp> get 6y3fp7_6y4so25_v4.bin
getting from 172.28.245.21:6y3fp7_6y4so25_v4.bin to 6y3fp7_6y4so25_v4.bin
[octet]
Received 8411 bytes in 0.1 seconds [612873 bit/s]
---------------------------------------------------------------------------

Download attempt with client entry but expired download window in LeaseDB:
Modify LeaseDB entry so the leasetime is outdated:
mysql> update leases_v4 set ts=(NOW()-1000) where ip=(
inet_aton('172.28.245.22'));

tftp> get 6y3fp7_6y4so25_v4.bin
getting from 172.28.245.21:6y3fp7_6y4so25_v4.bin to 6y3fp7_6y4so25_v4.bin
[octet]
Error code 2: Valid transfer time window exceeded, download denied!
```

Listing 8 – Standard TFTP client vs. Sentinel Stage 2

5.6 Stage 3, on-the-fly configuration file encoding

Stage 3 development is the final piece of feature code that will be added to the Sentinel. It allows the program to generate working DOCSIS configuration files at request time. This theoretically means that the Sentinel is able to generate files not only for specific groups of clients (client-class based), but also files specific for one particular client. This feature however was flagged as an optional objective, and may not be part of the actual implementation depending on effort involved.

5.6.1 Code development

Creating a DOCSIS configuration file is pretty complicated process as described in the most recent MAC and Upper Layer Protocols Interface Specification (Cable Labs, 2016). Basically any DOCSIS configuration setting is represented by a set of type, length and value parameters (TLV), which fall into two basic categories, mandatory and optional. Under normal circumstances, a tool able to create configuration files is required to be able to interpret and properly encode all parameters from both categories, which makes it a very large and comparatively slow process.

The whole file creation process can be abstracted to a simpler 4 stage process. Stage one, takes all the configuration type, length and value parameters in binary form and adds them to a byte array.

Stage two is the cable modem message integrity check (CMMIC) calculation. This is achieved by calculating a message digest 5 (MD5) checksum, as described in RFC 1321 (Rivest, 1992), over the entire byte array.

The third stage is the CMTS message integrity check (CMTSMIC) calculation, which takes an ordered sequence of certain configuration parameters, defined in DOCSIS, out of the byte array and calculates a MD5 hashed machine authentication code (HMAC), as described in RFC 2104 (Krawczyk, Bellare, Canetti, 1997). This HMAC calculation also requires a shared encryption key, which is the word “DOCSIS” by default.

Once both MICs are calculated, they are also attached as TLV’s to the byte array, as well as the so called “end of data marker” and “file padding” TLV’s, which complete the process and represent the final configuration file that can be interpreted by the client.

This file however, would be static and provides no additional benefit to a pre-encoded file, so the process needs to be refined further.

The reference network already has existing set of binary files, which already provide exactly what is required by clients. These files can be split up into several binary blocks, the majority of which are static and can be copied 1:1 from the source. Some of them however, need to be configurable, like up/down-stream speed and syslocation settings. A full review of the existing configuration file base resulted in the detection of the following non-generic parameters and the required level of change:

- Up/down-stream speed, value
- System location, vlaue
- Voice capability, block
- Wi-Fi capability, block
- Lowband capability, block
- IPv4 / IPv6 capability, block
- Vendor/Model specific parameters, block

Together with the generic parameter blocks, these parameters allow the creation of a custom byte array representing the configuration file for one specific client-class/modem type combination.

To implement this functionality into the Sentinel, a new class named configuration_file_creator was made, that is instantiated by the PSD if the requested filename is not present on the file system. The CFC only needs the original filename to extract the information which value and block combinations are necessary to build the file.

For simpler processing, the binary blocks are stored as parameters inside the configuration_file_creator class and the ClientDB is represented by a dictionary construct inside the CFC class.

Essentially the CFC parses the file name, which contains hints to the speed and system location value as well as the needed binary blocks, performs the MIC calculations from above and then writes the complete file to the local file system, where the PSD can pick it up and transmit it to the client. The stage 3 PSD process flow can be seen in figure 25.

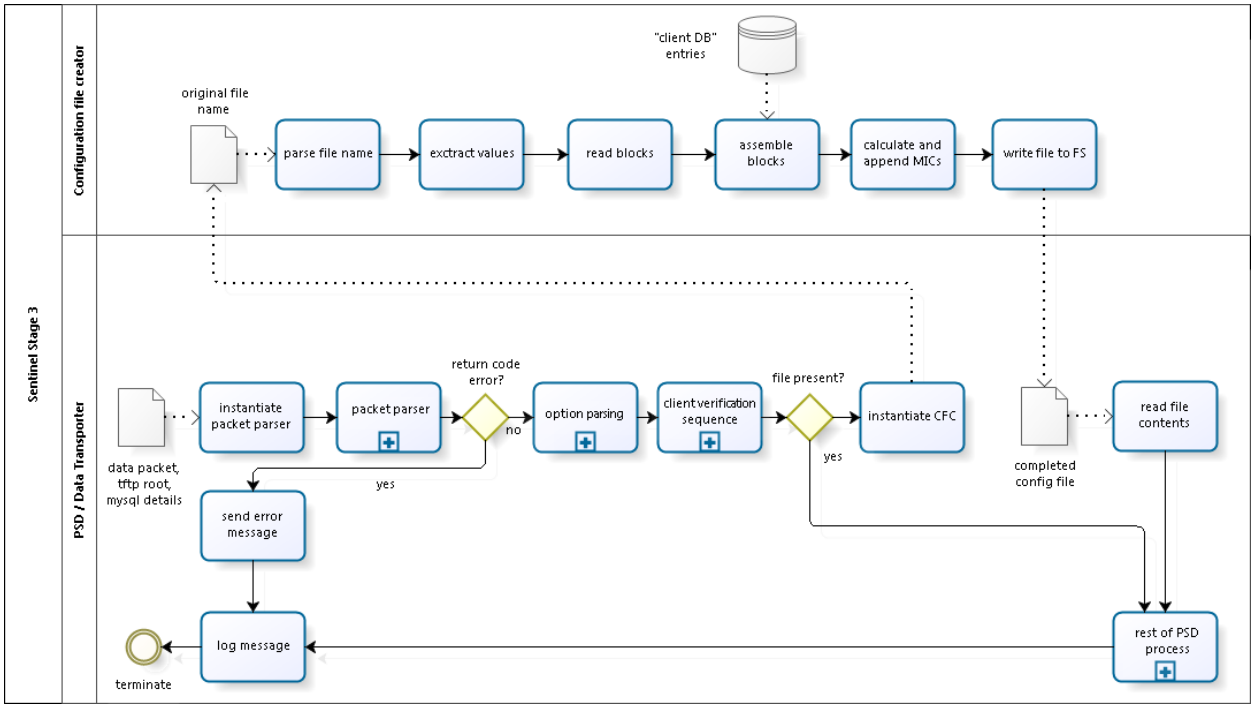

Figure 25 - Sentinel stage 3 process flow

5.6.2 Final functional verification

In order to verify the results of the CFC module, the Jackhammer program is fitted with a new test mode that can compare file contents on the byte level. To facilitate this, a pre-encoded bootfile is required to reside on the same server as the Jackhammer program. Jackhammer will then request a similarly named file from the Sentinel, which is written to disk in the regular fashion. Afterwards both the pre-existing file and the newly generated copy are loaded and compared byte per byte. The output includes the number of bytes in the file and any bytes that do not match the original.

For reference purposes, the same process is repeated using a regular TFTP client, and using the Linux ‘diff’ tool, which also allows comparing the contents of two files. It’s important to note that diff produces no output if the files match. The results of these tests can be seen in table 4, while the actual console output can be found in appendix A.

Filename	Jackhammer	Unix "diff"
6y3fp7_v4.bin	match	match
6y3fp7_v4-sip.bin	match	match
6y3fp7_v6.bin	match	match
6y3fp7_v6-sip.bin	match	match
6y8fp12_v4-sip.bin	match	match
6y8fp12_v4.bin	match	match
6y8fp12_v6-sip.bin	match	match
6y8fp12_v6.bin	match	match

Table 4 - file comparison test results

As expected the file contents match the original data.

6 DEPLOY THE PROTOTYPE

While the tests performed in the previous chapter verify the functional aspects of the Sentinel application, they do not actually test its performance limits or its robustness against malicious traffic and non-standard client behavior. This chapter will compare the Sentinel's performance against that of the legacy solution, review necessary performance enhancements to make it able to reach the defined goals.

As introducing the Sentinel application into a full production environment was not possible during the writing of this paper, a production-like environment was created to simulate expected service impact and validate the findings. To facilitate this, the pre-existing cable modem validation environment was modified to support the Sentinel application.

This environment would then also serve as a template for actual production roll-out after receiving approval. This allowed accurate data to be pulled from the system, without impairing production systems.

6.1 Performance test setup

As a complete live environment is hard to simulate in the lab, two upgraded Jackhammer instances were deployed on server 97 and 98. The Jackhammer code was modified so it is able to initiate multiple TFTP transfer sessions to the Sentinel running on server 99. The test scenarios were set up as follows:

- Average transfer time, TFTP only
- Time impact of client verification
- Time impact of configuration file creation
- Speed and maximum sessions with parallel downloads
- Same as above using two Jackhammers

Average transfer time using only basic TFTP functionality, consists of 1000 consecutive timed downloads of pre-existing configuration files. The result is an average calculated across the 1000 downloads, as well as minimum and maximum transfer times for the whole batch. Using only TFTP makes this test also viable against the legacy TFTP server.

Time impact of client verification repeats the above test suite, but includes the database lookups necessary to validate clients. This test will show if the database lookups require improvements.

Time impact of configuration file creation runs the consecutive tests again, but this time with the added task of newly creating each file before transfer.

All three of the above test suites are identical from the Jackhammer's perspective. The different behaviors are achieved solely by enabling/disabling debugging functions inside Sentinel.

Speed and maximum sessions with parallel downloads, puts Jackhammer into a mode where it aims to simulate multiple cable modems behind the same CMTS (as requests will always come from the same source IP). Several test runs, downloading a predefined file, using 1, 10, 100 and 1000 parallel sessions will be attempted against both the legacy solution and the Sentinel.

In an attempt to simulate multiple CMTS', the above test will also be repeated using a second Jackhammer instance on a different server.

6.2 Gather performance data

To facilitate the performance data gathering, server 99 was running the Sentinel and the legacy HPA TFTP server, but never at the same time. Jackhammer applications were operating from server 98 and 97 in dual server tests. Using this environment and running the test sets explained above the following results were gathered.

6.2.1 Average transfer time, TFTP only

For this test the Sentinel was switched into TFTP only mode, meaning it would not perform client verification or file creation, essentially mimicking the HPA TFTP server behavior. Listing 9 and 10 represent the outputs of the Jackhammer application when verifying TFTP only downloads.

```
Jackhammer TFTP consecutive download test:

Runtime: 12.3153 s for 1000 downloads. Filesize: 8411 bytes
Minimum DL time: 0.0086 s, Maximum DL time 0.1124 s, Average time: 0.0123 s
```

Listing 9 – Jackhammer consecutive download test vs. Sentinel, TFTP only

```
Jackhammer TFTP consecutive download test:

Runtime: 3.3232 s for 1000 downloads. Filesize: 8411 bytes
Minimum DL time: 0.0028 s, Maximum DL time 0.0537 s, Average time: 0.0033 s
```

Listing 10 – Jackhammer consecutive download test vs. HPA TFTP

On average the HPA TFTP proved to be almost 4 times faster than the Sentinel in TFTP only operations.

6.2.2 Time impact of client verification

For this test suite, the Sentinel's client verification subsystem was activated, while the file creation remained off. As the HPA TFTP does not support this function, the test run against it was omitted. The result can be seen in listing 11.

```
Jackhammer TFTP consecutive download test:
```

```
Runtime: 27.4319 s for 1000 downloads. Filesize: 8411 bytes
Minimum DL time: 0.0189 s, Maximum DL time 0.1219 s, Average time: 0.0274 s
```

Listing 11 – Jackhammer consecutive download test vs. Sentinel with client verification

Implementing the MySQL query necessary for client verification considerably increases average transfer time, more than doubling the resulting time.

6.2.3 Time impact of configuration file creation

For this test suite, the Sentinel's client verification was turned off again, and the file creation turned on. As before the test run against the HPA TFTP was omitted as it doesn't support this function. The results of the test can be seen in listing 12.

```
Jackhammer TFTP consecutive download test:

Runtime: 12.6309 s for 1000 downloads. Filesize: 8411 bytes
Minimum DL time: 0.0104 s, Maximum DL time 0.1126 s, Average time: 0.0126 s
```

Listing 12 – Jackhammer consecutive download test vs. Sentinel with file creation

Activating the file creation module only has a minimal impact on average transfer times.

6.2.4 Consecutive download tests in comparison

To complete the consecutive download tests, a final run was done with client verification and file creation turned on. This can be seen in listing 13.

```
Jackhammer TFTP consecutive download test:

Runtime: 28.4864 s for 1000 downloads. Filesize: 8411 bytes
Minimum DL time: 0.0184 s, Maximum DL time 0.0916 s, Average time: 0.0285 s
```

Listing 13 – Jackhammer consecutive download test vs. Sentinel with client verification and file creation

Summing up all the consecutive download tests, results in the graph shown in figure 26. With all security features turned on, the Sentinel is more than 8 times slower than the legacy solution.

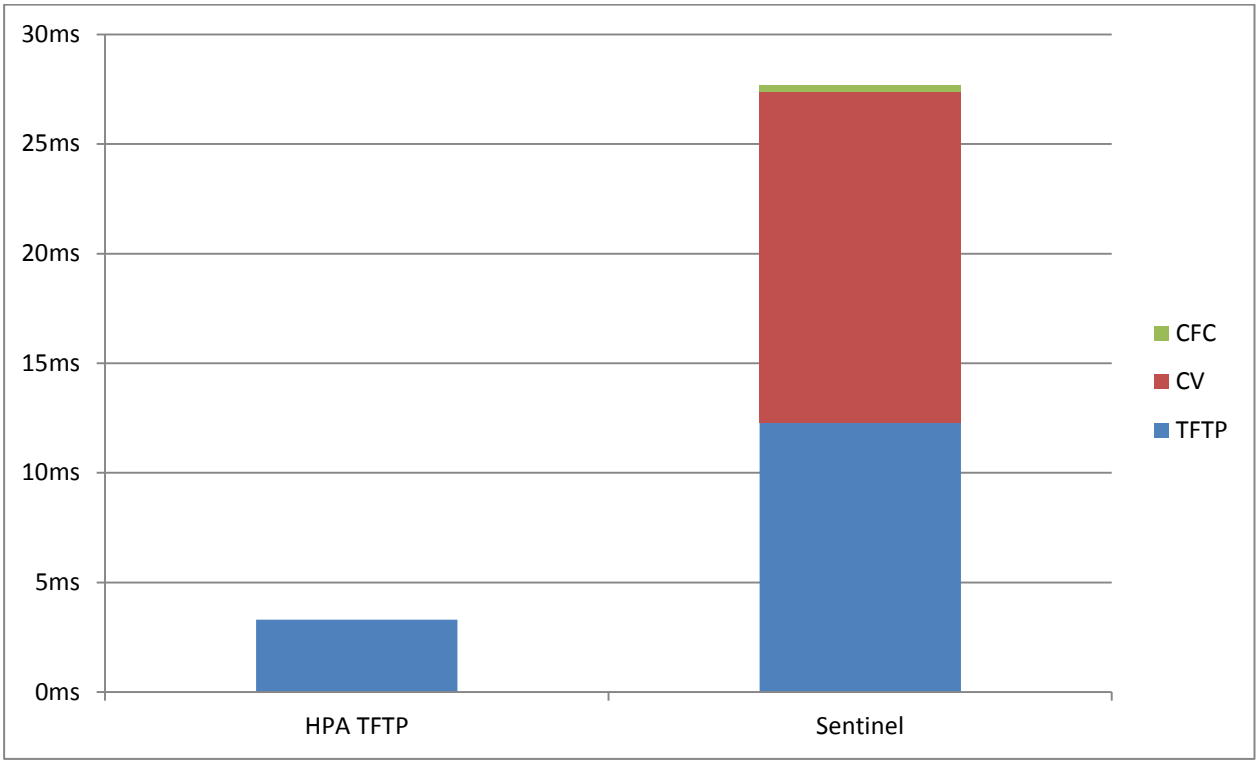

Figure 26 - consecutive download results

6.2.5 Speed and data throughput rate with parallel sessions

Similar to the previous tests, the parallel session tests will perform 100 downloads, but this time with several processes working in parallel, considerably increasing the strain on both the sender and receiver systems. Continuous file creation on the Sentinel has been disabled, as multiple processes interfere with each other when trying to download the same file.

Additionally the test file size was increased to 512 kB as the HPA TFTP server had severe issues operating with more than 50 parallel sessions trying to download the same 8411 bytes file.

The results can be seen in figure 27, while the console output can be found in appendix A. A time value of 0 indicates that the download produced timeouts, resulting in a failed test.

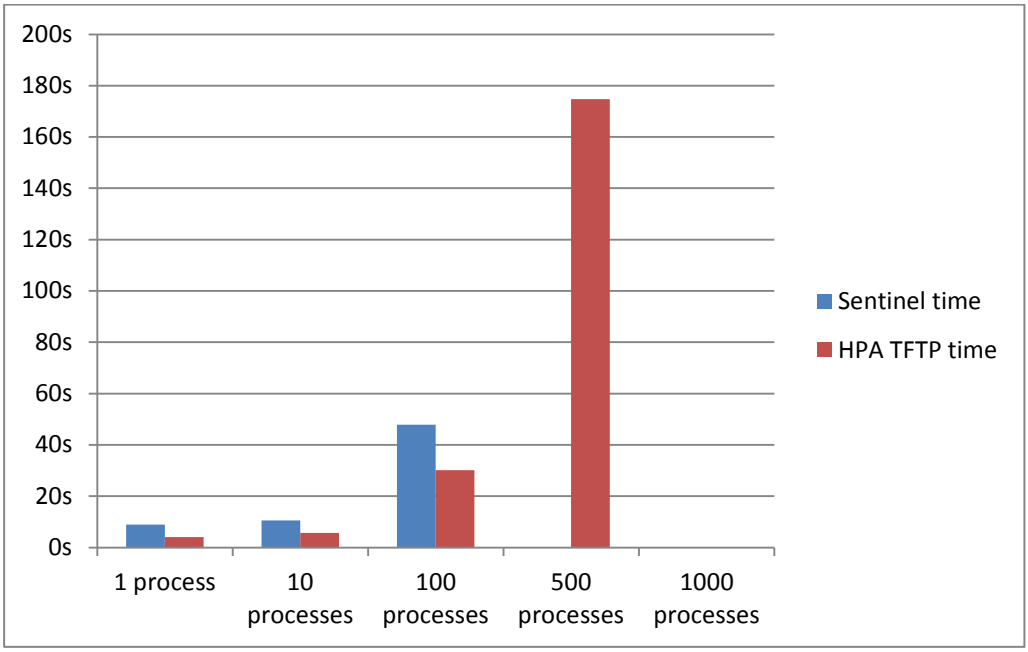

Figure 27 - parallel download runtime for 100 downloads each

Neither the HPA TFTP nor the Sentinel were able to reliably operate with 1000 concurrent sessions, so a new limit with 500 sessions was introduced that the HPA TFTP was able to reach, while the Sentinel failed again due to CPU constraints on the server.

6.2.6 Same as above using two Jackhammers

Using two Jackhammer applications against a single Sentinel or HPA TFTP instance provided no additional insights, as both applications were limited by server CPU power as soon as more than 500 parallel sessions were attempted.

6.3 Performance enhancements

While the Sentinel performs as expected when handling single requests, its ability to handle multiple client requests, seems to be severely limited by host system CPU performance. Closer investigation shows that it's mostly related to the comparatively slow PSD process spawning and terminated process joining. Hence the following modifications were applied to the code:

- PSD processes can handle more than one file transfer before terminating
- Spawned PSD processes do not terminate immediately after completing a transfer.
- Service controller does not check status of all processes in the active queue
- Service controller takes turns between joining processes and spawning new ones

In order to have *PSD processes handle more than one file transfer* an additional queue argument is passed to the PSD during spawning, which allows it to access data provided by the socket listener once the current transfer is complete, without going through the service controller. As the service controller still needs to know about any completed transfers and open

sessions a second queue is added that is only written to by the PSD processes and only read from by the service controller, which contains source address and status details.

With above modification the PSD process will still terminate if the transfer is complete and the listener queue is empty. To improve efficiency an additional timer loop is added that keeps the process alive for another 60 seconds while it waits for data from the listener queue. This behavior closely mimic's that of the legacy TFTP server when spawned from the extended internet daemon (xinetd).

In the initial implementation of the Sentinel, *the service controller always checks all processes in its service queue*, whether or not they have terminated. If there are only few processes in the queue, this is done quickly, but as the queue fills up, it takes more and more time, reducing its ability to spawn new processes. To counter this problem, another queue object is introduced, where a PSD ready for joining writes its own process object. So the service controller knows exactly when to join a process and doesn't need to check on every process all the time.

Taking turns between spawning processes, basically means, that the service controller only always joins one process before checking the listener queue for content. This results in a lower time between receiving the initial packet from the client and sending a response.

The Sentinel's process flow with performance enhancements can be seen in figure 28.

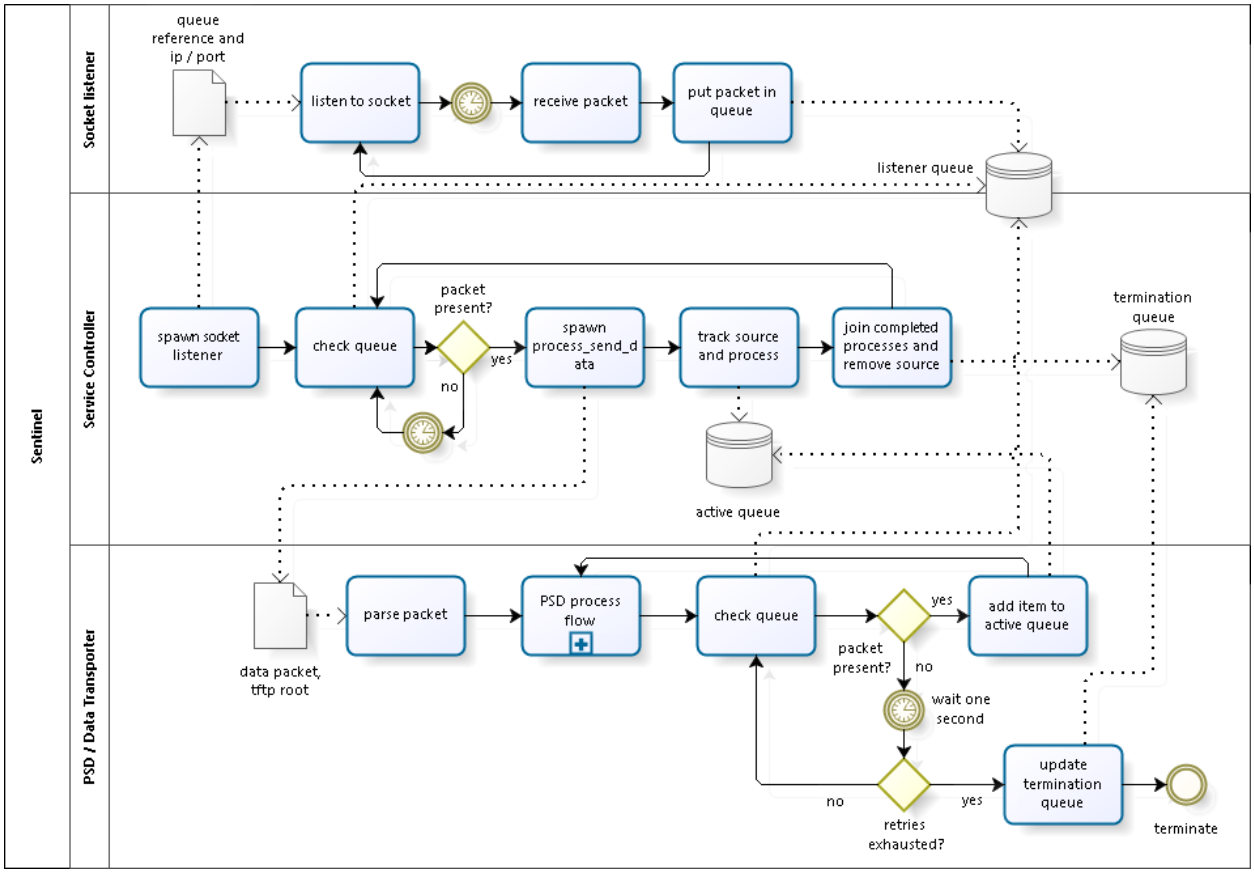

Figure 28 - Sentinel process flow with performance enhancements

Using these enhancements considerably boosts the applications performance when having to deal with >100 requests per second as can be seen in figure 29, the raw data can again be found in appendix A.

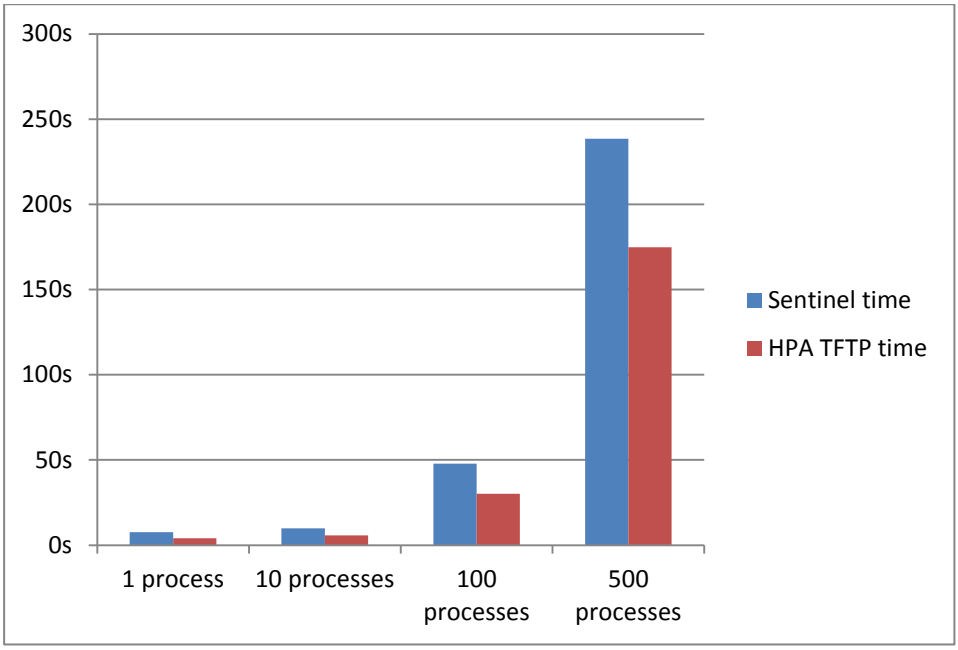

Figure 29 - parallel download runtime, 100 downloads each, with Sentinel enhancements

With the performance enhancements in the code the Sentinel performs close enough to the legacy solution, it also shows that the bigger the file size and the number of sessions the lower the impact of the database query for the client validation.

6.4 Deploy prototype in test environment

The live test environment, as mentioned at the beginning of the chapter is part of the cable modem firmware validation setup. This environment utilizes an LDAP server that's directly connected to the production environment, a DHCP server that is in the production network as well as two CMTS that have all active modem types of the reference network attached to them, although none of them belong to actual customers.

This means that the test environment closely resembles the actual production systems and thus allows the accurate estimation of both the work involved to get the live environment to support the Sentinel as well as the impact on security and provisioning flows, without negatively affecting customers.

The following actions, as seen in table 5, were necessary on the live test environment to support the Sentinel application.

Required action	Time (in hours)
Install/compile Python 3.5.2 on target system	0,1
Create sentinel user on production LeaseDB	0,1
Install/copy Sentinel application	0,1
Load modem templates	0,1
Verify modem templates	2
Create custom DHCP server extension	3
Apply/Validate extension	1
Perform modem verification tests	2
Total	**8,4**

Table 5 - Modifications and time required for Sentinel support on live test environment

The most notable fact is that there were no modifications on the provisioning side necessary to support the Sentinel application, due to its ability to extract all the necessary information about the clients from the client-class name and the data provided by the custom DHCP server extension.

As verification candidates, the following modem types were used, which reflect the majority of devices in use in the reference network:

- Technicolor TC7200.U
- Ubee EVW3226
- Thomson TWG870U
- Cisco EPC3925
- Compal CH7465LG

The objective of this test was to verify that the clients were able to successfully download a configuration file, which was created on demand by the Sentinel and were able to interpret the contained parameters, providing internet and voice services. Depending on the client-class template used the internet connectivity would be provided via IPv4 or IPv6. The individual results can be seen in table 6.

Model/Service	IPv4	IPv6	IPv4+voice	IPv6+voice	Client verification
TC7200.U	OK	OK	OK	OK	Success
TWG870U	OK	OK	OK	OK	Success
EVW3226	OK	OK	OK	OK	Success
EPC3925	OK	N/A	OK	N/A	Success
CH7465LG	OK	OK	OK	OK	Success
Jackhammer/TFTP	FAIL	FAIL	FAIL	FAIL	FAIL

Table 6 - Modem/service live test results

Note: As the EPC3925 does not support IPv6, it was excluded from the v6 tests.

Given that the previous file difference tests, showed no discrepancy between the manually generated files and the ones created by the Sentinel, it comes to no surprise that all modem types properly booted with the “on-demand” configuration files. The more important part was that even though client verification was activated, it did not interfere with the booting of legitimate clients.

Security measures were also tested using both the Jackhammer application as well as regular TFTP clients, by attempting to download files from the server, posing as a potential attacker and validating the client verification capabilities of the Sentinel. As expected, all download attempts by unauthorized clients failed.

7 EVALUATE THE RESULTS

This chapter will focus exclusively on the evaluation of the results gained in the previous chapters. Starting with a comparison of the KPI's of the Sentinel application against those of the legacy TFTP service, then checking if all the previously defined required and desired objectives were met, as well the security impact in respect to the ISO 27000 and finally the impact to the provisioning flows of the reference network.

7.1 Compare key performance indicators of prototype with those of legacy solution

Using the predefined list of KPI and the performance and capability data gathered in the previous chapters, makes it easy to compare the Sentinel to the legacy TFTP solution.

Requests per second were defined as how many concurrent TFTP read requests a single server node can reliably handle. Using the Jackhammer application for simulated load tests, both the legacy solution and the Sentinel became unreliable past 500 concurrent sessions. As the target value was the amount of concurrent sessions the legacy solution, the performance is acceptable.

Number of nodes was defined as the maximum number of servers the Sentinel can be deployed on. As there are no restrictions to the number of deployable nodes and every node can operate independently from any other node, this KPI is satisfied.

Time to deploy, quantifies how long it takes to deploy a new server node within the reference network. The only recurring tasks for a new server that differ from the legacy solution are the Python interpreter installation, the copying to the Sentinel application and the modem templates, which sum up to 0.3 hours. The average time of a legacy solution installation was given as 0.25 hours, which puts the Sentinel's result within the allowed 0.5 hour time window required by this KPI.

Time required to create a configuration file, specifies the time it takes the application to create a configuration file. Using the Jackhammer application this time was measured with >1ms, which considerably undercuts the >1000ms goal.

Time required to change a set of configuration files, quantifies the time an administrator must spend to change the contents of several configuration files. Using the byte array block construction within the Sentinel, changing the values of preexisting parameters equals the modification of a single value inside a text file, which takes effect for all files that use that block.

This stands in contrast to applying changes on the legacy solution, where it is required to manually update every file in the set of files. This KPI is met, if the file set is larger than 2, as the single update action takes the same amount of time.

Failover convergence time, defines how long it takes the system to return to a working state if any component fails. The only external dependency the Sentinel has is the connection to the LeaseDB, which is implemented via the MySQL connector module. The connector module itself allows internal failover, by specifying a secondary database server address. This secondary is used if the main server is unresponsive. In case both servers are not working, the Sentinel will switch to fallback mode, which bypasses client verification and permits the download, regardless of source. The convergence time for this is dependent on the response from the failed server. In the worst case scenario, this would be 2 seconds, which is way below the required 60 seconds for this KPI.

Amount of modifications required on the DHCP server, specifies the amount of man days required to create the necessary modifications to the DHCP servers. The effort involved to create the custom extension was specified as 3 hours, which is less than 0.5 MD, satisfying this KPI requirement with 5 MD maximum.

Number of modifications required on the Provisioning system, defines the work required to make the provisioning backend compatible with the Sentinel application. As the Sentinel is completely transparent to the provisioning system, the effort was 0 MD, also satisfying this KPI requirement of 5 MD maximum.

Report quality, specifies how much logging data the service provides. The Sentinel provides extensive logging capabilities up to individual data packet logging for debugging purposes, which can be configured via the global option parameters. Default operational logs include, read request, filename, options, client information and MySQL query response, as well as success or rejection messages. These logging options exceed those of the legacy solution, meeting the requirements for this KPI.

RFC conformity specifies how accurately the application follows the RFC 1350 guidelines. Tests with the Jackhammer application indicate that Sentinel is fully compliant to RFC 1350 RRQ messages using octet transfer mode, but neither supports WRQ messages, for security reasons, nor netascii transfer modes, as its not supported by cable modems. Resulting in the fact that this KPI is 100% fulfilled.

Unauthorized client rejection rate, measures the accuracy of the service in regards to being able to identify clients entitled for downloads. Tests with both the Jackhammer application and a regular TFTP client, show that unauthorized clients were rejected 100% of the time, whereas legitimate clients had no problems downloading the requested files. The fulfillment of this KPI can be temporarily suspended if the Sentinel enters emergency fallback mode, as reduced security is preferable to total systems outage. Otherwise, the Sentinel fulfills this KPI to 100%.

Data throughput rate, measures the number of bytes transferred per second. As there are several parameter influencing the data throughput rate like file size, packet size and number of parallel sessions, the Jackhammer test results from listings 17 and 18 were used as reference.

This test consisted of 500 parallel downloads of 512kB files, which were downloaded 100 times each, resulting in a total of 25GB of data. The legacy solution took 174.72 seconds, which equals about 142MB/s, whereas the Sentinel took 238.49 seconds which equals about 104MB/s. ~73% of the throughput of the legacy solution meets the required 50% minimum requirement for this KPI.

For easier reference, the individual KPI and their results can be seen in table 7.

KPI	Sentinel value	Reference value
Requests per second	500 concurrent	500 concurrent
Number of nodes	only limited by infrastructure	num existing nodes
Time to deploy	0.3 h	0.5 h
Time to create file	< 1 ms	1000 ms
Time to change parameters	0,1 h	0,1 h * num files
Failover convergence time	2 s	60 s
Amount of modifications on DHCP	< 0.5 MD	5 MD
Number of modifications on Provisioning	0 MD	5 MD
Report quality	> 100,00%	100,00%
RFC conformity	100,00%	100,00%
Unauthorized client rejection rate	100,00%	100,00%
Data throughput	73,00%	50,00%

Table 7 - KPI overview

7.2 Compare objectives with results

As the KPI were based off the requirements it is very unlikely that any required or desired objective was not met by the Sentinel application, but for completeness, this sub-chapter will re-examine the objectives and match them to application capabilities.

7.2.1 Required capabilities

Required capabilities are the baseline requirements of the application, that technically match the capabilities of the legacy solution, these were:

- RFC 1350 compliant binary TFTP downloads must be supported. Fulfilled by RFC conformity KPI.
- Must be compatible with existing server and network infrastructure. Fulfilled, as both development and testing Sentinel instances were deployed on systems matching the production environment.
- Must be compatible with existing provisioning system. Fulfilled, as sentinel is transparent to the provisioning backend.

- Must be compatible with Cisco CNR. Fulfilled as test environment DHCP server was running Cisco CNR, with custom extension.
- Must be scalable. Fulfilled by “number of nodes“ KPI.
- Must be compatible with existing TFTP directory structure. Fulfilled as Sentinel can operate with the existing directory structure and generate missing files automatically.
- Must provide feedback/logging on a per client basis. Fulfilled by “report quality” KPI.
- Must not allow modifications of the underlying OS or file system. Fulfilled, as Sentinel does not support WRQ’s and prevents clients from ”escaping“ the defined TFTP download directory.
- Must work autonomously and generate regular usage reports. Fulfilled, as Sentinel operates autonomously and independently from other Sentinel installations and generates meaningful syslog entries for any successful or attempted downloads.

7.2.2 Desired capabilities

Desired capabilities set the application apart from the legacy solution, improve system performance, security or reduce the amount of manual intervention:

- Implement RFC 2347, 2348, 7440 option capabilities for TFTP transfers, as well as the hardware and network address option draft, for increased performance. Fulfilled in code development and verified by Jackhammer application.
- Client source verification and transfer request validation, for increased security. Already fulfilled by “unauthorized client rejection rate” KPA. Implemented by SQL queries to LeaseDB and verified by Jackhammer and cable modems.
- Automatic failover, for increased resilience. Already fulfilled by “failover convergence time” KPI, implemented by MySQL connector failover and timeout functionality.
- Client specific on-the-fly configuration file encoding, to lower operational effort. Fulfilled by byte array block based configuration file construction that allows on demand creation of files tailored to a specific group of clients.
- Modular construction, to allow for more modifications later on. Fulfilled by modular, object oriented code design.

7.2.3 Optional capabilities

Optional capabilities defined some “nice to have“ features for the Sentinel, which improve its usefulness but are not part of the core application, these were:

- Administrative graphical user interface (GUI). Not implemented, scheduled for future release.

- Real-time reporting. Not implemented, partially replicated by existing reference network statistics collection tools.
- On demand MTA configuration file generation and encryption. Not implemented, considerable additional development effort needed. Encryption key distribution varies on a device level. Scheduled for future implementation
- Individual client adjustments. Not implemented, as adding individual adjustments would have caused impact to the provisioning system, by adding a new configuration layer. Future implementation likely.
- The ability to handle hypertext transfer protocol (HTTP) traffic in addition to TFTP. Not implemented. Feature currently only used for firmware downloads, which do not benefit from the Sentinels added security features.
- RFC 2349 support. Fulfilled, due to trivial implementation effort.

7.2.4 Unwanted characteristics

Unwanted characteristics represent objectives that the Sentinel needed to avoid at all costs, they were:

- Licensing costs. Fulfilled as the Sentinel was built exclusively using free software.
- Only useable in the reference network. Fulfilled, as it was shown that the Sentinel could serve as a drop-in replacement for existing TFTP server solutions, provided that lease and client-class information could somehow be made available.
- Specialized hardware or software. Fulfilled, as Sentinel does not rely on special hard or software. Its modular nature also allows it to interface with different types of databases with minimal extra effort.

Overall, the capabilities and characteristics can be summed up as shown in table 8.

Capability/Characteristic	**Fulfilled?**
RFC 1350 compliant	Yes
Compatible with existing infrastructure	Yes
Compatible with existing provisioning system	Yes
Compatible with Cisco CNR	Yes
Scalable	Yes
Compatible with existing TFTP directory structure	Yes
Provides feedback/logging	Yes
Does not allow modifications of underlying OS	Yes
Works autonomously	Yes
RFC 2347, 2348, 7740 and netaddr draft support	Yes
Performs client source verification	Yes

Capability/Characteristic	Fulfilled?
Implements automatic failover	Yes
Client specific on-the-fly configuration file creation	Yes
Modular design/construction	Yes
Administrative GUI	No
Real-time reporting	No
MTA configuration file creation	No
Individual client adjustments	No
HTTP transfer capability	No
RFC 2439 support	Yes
No licensing costs	Yes
Useable outside of reference network	Yes
No Specialized hard-/software needed	Yes

Table 8 - Requirements fulfillment

7.3 Evaluate impact on system security

As the ISO 27000 is synonymous with systems security, the Sentinel application and its impact on systems security will need to be reviewed specifically with the applicable ISO 27001 control clauses in mind. These were:

- Asset management
- Access control
- Cryptography
- Operations security
- Communications security
- Acquisition, development and maintenance
- Information security incident management
- Redundancies
- Compliance

As stated in chapter 3.2, the Sentinel introduction changes neither the responsibility nor the information classification, causing limited impact to *asset management*. Corporate engineering retains ownership of the server assets, does implementations and upgrades as well as last line troubleshooting. Local operations keep ownership of day to day operations, basic troubleshooting and configuration data management.

While there are no changes to the predefined administrative user access and responsibilities, the Sentinel's client source verification functionality considerably increases the security of the

information access aspect of the *access control* clause. Using both the Jackhammer application and a regular TFTP client, a series of tests was run against the Sentinel, to try to download different existing and non-existing files from the server. All these attempts failed as the verification was done against the LeaseDB, which is exclusively fed by the DHCP server and did not contain an entry matching the Jackhammer or TFTP client. This functionality is further enhanced by the fact that even a legitimate client only has a small time window available after acquiring a lease in which it can download a configuration file, as the DOCSIS specification dictates that clients should only be downloading files shortly after getting a new IP lease. This aspect can now be considered secure.

While the TFTP packets are still not encrypted, as the standard does not allow this, the communication between the Sentinel and the MySQL LeaseDB however uses the MySQL server's built-in *cryptography* functionality. This is also the reason when the SQL queries almost double the average transfer time for a single file, if client verification is enabled.

From an *operations security* perspective, two changes will need to be implemented should the Sentinel be introduced into a production environment. The operational procedures handbook will need to be updated to reflect changes and basic troubleshooting for the Sentinel application, detailing both expected behavior and communication matrices, as well as necessary measures that can be taken if any unexpected behavior is seen. In addition a comprehensive log parser will be required to perform regular audits so abnormal behavior and potential attacks can be spotted. As access to the central version control system as well as access to the central monitoring solution (Nagios) is already in place, no additional tasks are necessary on the monitoring and backup management side.

Communications security is established by the Sentinel's ability to ascertain, that a client is not just eligible to download a file within a certain time window after boot, but also that the file its attempting to download actually matches what the client is supposed to download, by checking the contents of the LeaseDB client-class information against the transfer request file name. This measure prevents a client that managed to receive from or spoof a lease on the DHCP server to download any random file from the Sentinel, during the allowed time period.

The security requirements for information systems and test data protection of the *acquisition, development and maintenance* clause require that further code development still needs to adhere to the same security guidelines as the initial code and that re-tests can be performed with each new iteration of the service. The former will be dealt with in the way that further development of the Sentinel will be done by the original author who is aware of the existing guidelines and is also informed of any relevant changes to them. Whereas the latter is mainly provided by the Jackhammer application that can always test and verify existing functionality and will also receive mandatory upgrades to support testing of any new functionality.

To enable proper *information security incident management*, the Sentinel was fitted with extensive logging capability, that does not just keep track of successful downloads like the legacy solution, but also provides information on abnormal requests, illegal packet requests, detailed client source information and the reason for rejection. To make use of this information

however, the reports would need to be extracted and parsed on a daily level by the existing monitoring solution.

The information security aspect of *redundancy* was implemented in the Sentinel in multiple ways. All Sentinel nodes operate individually and do not interact with each other. They merely share the same code and configuration base, which is distributed by the central repository. The only external data dependency present is the link to the LeaseDB server, which consists of a cluster with two nodes that operate independently, reside in two different geographic locations, but receive the same data feed, resulting in two independent but identical data sources. The Sentinel accesses these servers using the built-in MySQL-connector failover functionality, as well as a global 1 second timeout in case both nodes are unavailable. In fallback mode the Sentinel is unable to validate client requests, resulting in a temporarily lowered security level. As mentioned in the KPI section, this temporary loss of security is preferable to a total systems outage. Sentinel also automatically switches client verification back on as soon as connectivity the databases is reestablished. However, given that the legacy solution did not require this kind of redundancy, the overall impact on systems security is minimal.

Finally, Sentinel was designed to be *compliant* with existing security guidelines as it does not "speak" any other protocols than TFTP to the client side. Additionally it shares the secured container environment, IP access lists and network infrastructure with existing legacy TFTP installations, meaning that the application can't negatively affect other systems or be compromised by non TFTP targeted attacks.

The impact on general systems security and the effort involved to implement these measures can be seen at a glance in table 9. Items tagged with minimal effort, have already been included in the Sentinel code.

ISO 27000 security clause	Impact	Effort
Asset management	Minimal improvement	Minimal effort
Access control	Considerable improvement	Minimal effort
Cryptography	Minimal improvement	Minimal effort
Operations security	Moderate improvement	Moderate effort
Communications security	Moderate improvement	Minimal effort
Acquisition, development and maintenance	Minimal improvement	Minimal effort
Information security incident management	Moderate improvement	Moderate effort
Redundancies	Minimal improvement	Minimal effort
Compliance	Moderate improvement	Minimal effort

Table 9 - Sentinel impact on system security and effort required

7.4 Evaluate impact on provisioning flow

As it was possible to deploy the Sentinel application in a near live environment, with full functionality and not a single modification required on the provisioning system, the application can be considered vendor agnostic and fully transparent to the provisioning backend. As a result, the overall impact on provisioning service flow equals zero.

8 SUMMARY AND PROSPECTS

The final chapter of this thesis will focus on the conclusion drawn from the results, give thoughts on the overall development process as well as prospects for the created prototype and possible future upgrades to the system.

8.1 Conclusion

With the Sentinel it was possible to create a prototype implementation of a dynamic TFTP service that can serve as a drop-in replacement for existing TFTP servers with very little additional effort, while generally hardening the system in respects to the ISO 27000. It has no impact on legitimate clients or the provisioning service flow, requires minor modifications on the DHCP platform and considerably raises the security of the service against attackers and non-authorized clients, while reducing the overall workload on the administrative staff. Viewed from a system theory perspective this results in the picture seen in figure 30.

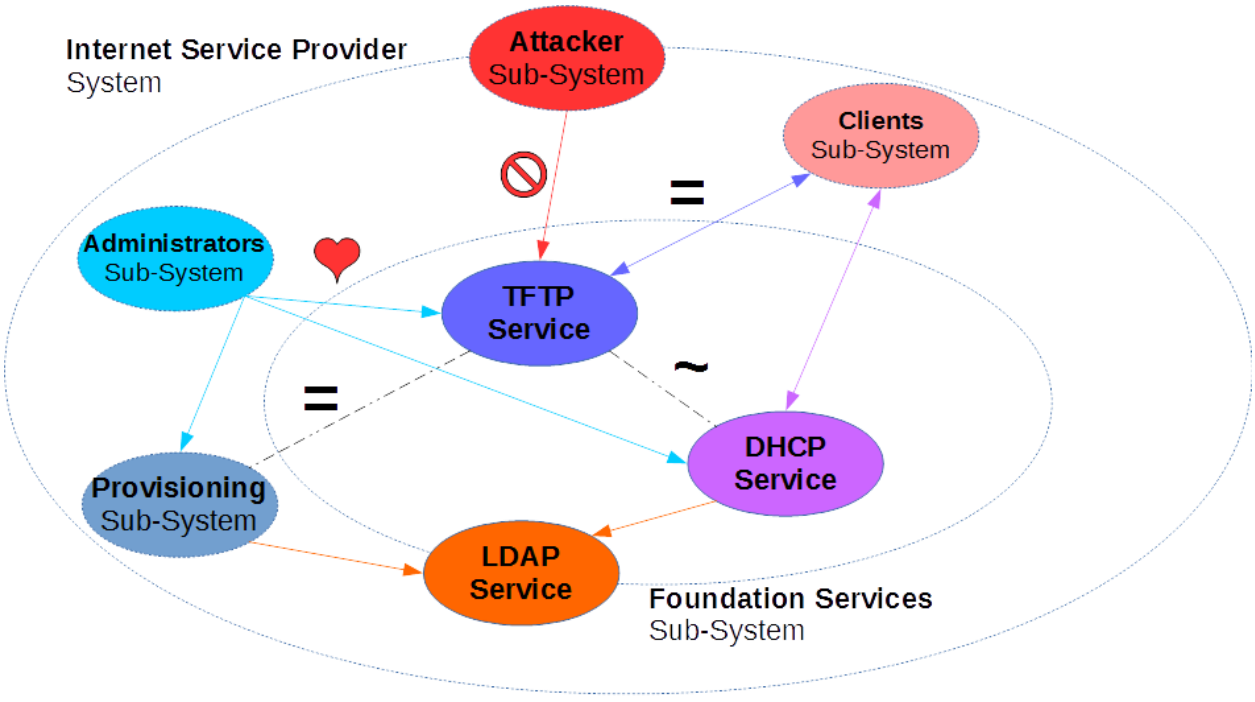

Figure 30 - The Sentinel's impact on related (sub-) systems

Using reference parameters gathered from the legacy solution and tested with both existing tools and the Jackhammer application, which was created exactly for this purpose, it was possible to show that the initial H0 hypothesis was correct.

- H0: The introduction of a properly designed dynamic TFTP service in a large provider network reduces the operational effort involved in creating DOCSIS configuration files, increases systems security against attackers and is completely transparent to the clients, while having a calculable effect on the DHCP service and the provisioning system.

From the authors perspective this makes the prototype an outstanding success and the development effort time well spent.

8.2 Thoughts on the development process

In hindsight using Python as the language of choice for the Sentinel implementation was maybe not the optimal decision. While the language overall is very easy and comfortable to code with, there are certain quirks to it, especially when it comes to multiprocessing, multithreading and the passing of data between threads/processes, that other languages deal with in a much better way. While the resulting code matched the requirements, there is still the lingering feeling that things could have been done better.

The biggest surprise during development certainly was the fact that creating a configuration file on the fly was possible in less than one millisecond, which is far less that initially anticipated as the reference value from a manually generated file was somewhere around 1 second per file. Arguably, the manual creation process does a lot more things in background before actually writing the file, that the “binary only” implementation of the Sentinel does not need, hence the tremendous speed difference.

The need for a management tool that simplifies the tasks of loading configuration parameters into the Sentinel became quite obvious at the end of the development cycle, which will most likely be the first addition to the Sentinel service once it gained approval for widespread production implementation within the Liberty Global affiliates.

8.3 Consider stage 4 and beyond implementations

Even before the stage 3 development was finished, it was obvious that the Sentinel could do more than just generate configuration files for cable modems. One such item would be the on-the-fly creation of configuration files for MTA’s. With the added encryption and the somewhat inhomogeneous implementation of the provisioning and encoding flows for these devices it makes the implementation quite challenging and worst of all, requires considerable, non-negotiable adjustments on the provisioning backend. Once the initial stage 3 Sentinel has entered widespread service, the author is positive that development budget will be made available to overcome the required changes and implement a stage 4 service, which would then include configuration file creation for all client types present in the network. Meanwhile, the added layer of security provided by the Sentinel’s client and file authentication will need to be sufficient.

Another provisioning related change that the author has been approached with, by the responsible team, includes a drastic reduction of the present client classes, which currently need to be doubled, every time a new feature needs to be implemented. The idea would be to create only a single client-class, which mandates the speed values of the service, while all additional production options would be encoded in LDAP parameters. As now with the custom extension it would be the DHCP server's job to encode these class names and values into a configuration file name that the Sentinel can create a custom bootfile of. As can be expected a change like this requires considerable changes on the provisioning backend, but would greatly simplify the introduction of new client features.

8.4 Additional challenges

The Sentinel was created as generic as possible, with all configuration and DOCSIS parameters made adjustable. This still means however that there needs to be a one-time adjustment for every new deployment it will work in, as well as updates every time a new feature is introduced. The effort for this should be low, so a certain set of tools will be required for local staff to work with. These tools, as well as the database frontend mentioned above, will very likely be required as a baseline for the Sentinel service.

GLOSSARY

ITU	International Telecommunication Union
ISO	International standards organization
DOD	Department of Defense
OSI	Open Systems Interconnection
IP	Internet Protocol
RFC	Request for comment
IETF	Internet engineering task force
DSL	Digital subscriber line
DOCSIS	Data over cable interface specifications
ISDN	Integrated services data network
POTS	Plain old telephone system
CMTS	Cable modem termination system
DSLAM	Digital subscriber line access multiplexer
ANSI	American national standards institute
SCTE	Society of cable telecommunications engineers
TFTP	Trivial file transfer protocol
CM	Cable modem
ISP	Internet service provider
HFC	Hybrid fibre cable
CRM	Customer relationship management
WAN	Wide area network
LAN	Local area network
eSAFE	Embedded service/application functional entity
CPE	Customer premises equipment
DHCP	Dynamic host configuration protocol
UDP	User datagram protocol
RRQ	Read request

WRQ	Write request
FTP	File transfer protocol
SCP	Secure copy
TCP	Transfer control protocol
MTA	Media terminal adapter
API	Application programming interface
IEC	International electro technical commission
KPI	Key performance indicators
Wi-Fi	Wireless fidelity
EUI	Extended unique identifier
CD	Collision domain
IEEE	Institute of electrical and electronics engineers
VLAN	Virtual local area network
VRRP	Virtual router redundancy protocol
OSPF	Open shortest path first
BGP	Border gateway protocol
IS-IS	Intermediate system to intermediate system
LDAP	Lightweight directory access protocol
STB	Set-top box
ToD	Time of day
MAC	Media access control
BOOTP	Boot protocol
BPI+	Baseline privacy plus
VoD	Video on demand
DoS	Denial of service
DDoS	Distributed denial of service
RAID	Redundant array of independent disks
OS	Operating system
SPOF	Single point of failure
GUI	Graphical user interface
HTTP	Hypertext transfer protocol

DES	Data encryption standard
AES	Advanced encryption standard
MD	Man days
DTFTP	Dynamic trivial file transfer protocol
CNR	Cisco network registrar
SL	Socket listener
CV	Client validator
CFC	Configuration file creator
DT	Data transporter
IDE	Integrated development environment
CPU	Central processing unit
MP	Main process
SC	Service controller
DB	Database
LDB	Lease database
CDB	Client database
SMTP	Simple mail transfer protocol
LDBC	Linux daemon base class
GIL	Global interpreter lock
PSD	Process send data
PRD	Process receive data
ACK	Acknowledge
PP	Packet parser
OACK	Option acknowledge
MTU	Maximum transfer unit
TLV	Type, length, value
MIC	Message integrity check
MD5	Message digest 5
HMAC	Hashed machine authentication code

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

International Telecommunication Union. (1997). Open Systems Interconnection – Model and Notation
Retrieved from http://www.itu.int/rec/T-REC-X.200-199407-I

V. Cerf, E. Cain. (1983). The DoD Internet Architecture Model
Retrieved from
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.7505&rep=rep1&type=pdf

Information Sciences Institute University of Southern California. (1981). DARPA Internet Program Protocol Specification
Retrieved from https://tools.ietf.org/html/rfc791

S. Deering, R. Hinden. (1988). Internet Protocol, Version 6 (IPv6) Specification
Retrieved from https://tools.ietf.org/html/rfc2460

K. Sollins. (1981). The TFTP Protocol
Retrieved from https://tools.ietf.org/html/rfc783

K. Sollins. (1992). The TFTP Protocol (Revision 2)
Retrieved from https://tools.ietf.org/html/rfc1350

Liberty Global. (2015). Liberty Global Reports Fiscal 2015 Results
Retrieved from http://www.libertyglobal.com/pdf/press-release/earnings/LG-Earnings-Release-Q4-15-FINAL.pdf

M. Cotton, L. Vegoda, R. Bonica, B. Haberman. (2013). Special-Purpose IP Address Registries
Retrieved from https://tools.ietf.org/html/rfc6890

Cisco Systems. (2016). Cisco Network Registrar
Retrieved from http://www.cisco.com/c/en/us/products/cloud-systems-management/prime-network-registrar/index.html

ANSI/SCTE. (2013). DOCSIS 3.0 Part 1: Physical Layer Specification

Retrieved from http://www.scte.org/documents/pdf/Standards/ANSI_SCTE%20135-1%202013.pdf

ANSI/SCTE. (2008). Graphic Symbols For Cable Systems Part 1: HFC Symbols
Retrieved from http://www.scte.org/documents/pdf/Standards/ANSI_SCTE%2087-1%202008.pdf

ANSI/SCTE. (2013). DOCSIS 3.0 Part 4: Operations Support Systems Interface, fiber node definition
Retrieved from http://www.scte.org/documents/pdf/standards/top%20ten/ansi_scte%20135-4%202013.pdf

IEEE. (2003). Virtual Bridged Local Area Networks
Retrieved from http://standards.ieee.org/getieee802/download/802.1Q-2003.pdf

IEEE. (2015). IEEE Standard for Ethernet
Retrieved from http://standards.ieee.org/getieee802/download/802.3-2015.zip

R. Droms, Bucknell University. (1997). Dynamic Host Configuration Protocol
Retrieved from http://tools.ietf.org/html/rfc2131

M. Shand, L.Ginsberg. (2014). OSI IS-IS Intra-domain Routing Protocol
Retrieved from https://tools.ietf.org/html/rfc7142

J. Moy. (1998). OSPF Version 2
Retrieved from https://tools.ietf.org/html/rfc2328

K. Lougheed. Y. Rekhter. (1989). A Border Gateway Protocol (BGP)
Retrieved from https://tools.ietf.org/html/rfc1105

Cable Television Laboratories, Inc. (2001). Radio Frequency Interface Specification
Retrieved from http://www.cablelabs.com/wp-content/uploads/specdocs/SP-RFI-C01-011119.pdf

J. Sermersheim. (2006). Lightweight Directory Access Protocol (LDAP): The Protocol
Retrieved from https://tools.ietf.org/html/rfc4511

Cisco Systems. (2011). User Guide for Cisco Network Registrar 7.2
Retrieved from http://www.cisco.com/c/en/us/td/docs/net_mgmt/network_registrar/7-2/user/guide/cnr72book.html

B. Croft, J. Gilmore. (1985). Bootstrap Protocol (BOOTP)
Retrieved from https://tools.ietf.org/html/rfc951

International Organization for Standardization. (2013). ISO/IEC ISO 27001 / BS 7799-2:2000
Retrieved from http://www.iso.org/iso/catalogue_detail?csnumber=54534

International Organization for Standardization. (2013). ISO/IEC 27002 / BS 7799-1:2000
Retrieved from http://www.iso.org/iso/catalogue_detail?csnumber=54533

Communications Fraud Control Association (2016). Global Fraud Loss Survey 2015
Retrieved from http://cfca.org/fraudlosssurvey/2015.pdf

Department for Business, Innovation and Skills (2015). 2015 Information security breaches survey 2015
Retrieved from http://www.pwc.co.uk/assets/pdf/2015-isbs-technical-report-blue-digital.pdf

Neustar, Inc. (2016), April 2016 Neustar DDoS Attacks & Protection Report
Retrieved from https://ns-cdn.neustar.biz/creative_services/biz/neustar/www/resources/whitepapers/it-security/ddos/2016-apr-ddos-report.pdf

Cable Television Laboratories, Inc. (2008). Baseline Privacy Plus Interface Specification CM-SPBPI+-C01-081104
Retrieved from http://www.cablelabs.com/wpcontent/uploads/specdocs/CM-SP-BPI+-C01-081104.pdf

J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler. (2002). SIP: Session Initiation Protocol
Retrieved from https://tools.ietf.org/html/rfc3261

Cable Television Laboratories, Inc. (2009). PacketCable 1.5 Specification MTA Device Provisioning
Retrieved from http://www.cablelabs.com/wp-content/uploads/specdocs/PKT-SP-PROV1.5-I04-090624.pdf

Cable Television Laboratories, Inc. (2009). MAC and Upper Layer Protocols Interface Specification
Retrieved from http://www.cablelabs.com/wp-content/uploads/specdocs/CM-SP-MULPIv3.0-I29-151210.pdf

G. Malkin, A. Harkin. (1998). TFTP Option Extension
Retrieved from https://tools.ietf.org/html/rfc2347

G. Malkin, A. Harkin. (1998). TFTP Blocksize Option
Retrieved from https://tools.ietf.org/html/rfc2348

G. Malkin, A. Harkin. (1998). TFTP Timeout Interval and Transfer Size Options
Retrieved from https://tools.ietf.org/html/rfc2348

P. Masotta. (2015). TFTP Windowsize Option
Retrived from https://tools.ietf.org/html/rfc7440

E. Gamma, R. Helm, R. Johnson, J. Vlissides. (1995). Design Patterns: Elements of Reusable Object-Oriented Software.
Addison-Wesley. ISBN 0-201-63361-2.

National Institute of Standards and Technology. (2001). Advanced Encryption Standard (AES)
Retrieved from http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

L. von Bertallanfy. (1969). General System theory: Foundations, Development, Applications
George Braziller Inc., ISBN 0-8076-0453-4

N. Diakopoulos, S. Cass. (2016). The Top Programming Languages 2016
Retrieved from http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016

Advanced Micro Devices. (2013). AMD64 Architecture Programmer's Manual Volume 1
Retrieved from http://support.amd.com/TechDocs/24592.pdf

Python Software Foundation. (2015). Python – Case studies & Success Stories
Retrieved from http://brochure.getpython.info/media/releases/psf-python-brochure-vol.-i-final-download.pdf/at_download/file

J. Littlefield. (2004). Vendor-Identifying Vendor Options for Dynamic Host Configuration Protocol version 4 (DHCPv4)
Retrieved from https://tools.ietf.org/html/rfc3925

S. Marechal, Anonymous. (2009). Generic Linux daemon base class for python 3.x
Retrieved from http://web.archive.org/web/20131025230048/http://www.jejik.com/articles/2007/02/a_simple_unix_linux_daemon_in_python/

Python Software Foundation. (2009). PEP 3143 -- Standard daemon process library
Retrieved from https://www.python.org/dev/peps/pep-3143/

Python Software Foundation. (2009). PEP 371 -- Addition of the multiprocessing package to the standard library

Retrived from https://www.python.org/dev/peps/pep-0371/

Cable Television Laboratories. (2016). MAC and Upper Layer Protocols Interface Specification, CM-SP-MULPIv3.1-I09-160602

Retrieved from https://community.cablelabs.com/wiki/plugins/servlet/cablelabs/alfresco/download?id=6695168c-fc27-4aed-9179-4f68613ecc8a

R. Rivest. (1992). The MD5 Message-Digest Algorithm

Retrieved from https://tools.ietf.org/html/rfc1321

H. Krawczyk, M. Bellare, R. Canetti. (1997). HMAC: Keyed-Hashing for Message Authentication

Retrieved from https://tools.ietf.org/html/rfc2104

APPENDIX A

Console output of binary compare tests, chapter 5.6.2 for Jackhammer, listing 14 and “diff”, listing 15.

```
Jackhammer TFTP binary compare test:

Downloading file: 6y3fp7 v4.bin
Comparing with file: /tftpboot/6y3fp7_v4.bin
Result: Files match, 8400 bytes

Downloading file: 6y3fp7_v4-sip.bin
Comparing with file: /tftpboot/6y3fp7_v4-sip.bin
Result: Files match, 8532 bytes

Downloading file: 6y3fp7_v6.bin
Comparing with file: /tftpboot/6y3fp7_v6.bin
Result: Files match, 4856 bytes

Downloading file: 6y3fp7_v6-sip.bin
Comparing with file: /tftpboot/6y3fp7_v6-sip.bin
Result: Files match, 4988 bytes

Downloading file: 6y8fp12_v4-sip.bin
Comparing with file: /tftpboot/6y8fp12_v4-sip.bin
Result: Files match, 8536 bytes

Downloading file: 6y8fp12_v4.bin
Comparing with file: /tftpboot/6y8fp12_v4.bin
Result: Files match, 8404 bytes

Downloading file: 6y8fp12_v6-sip.bin
Comparing with file: /tftpboot/6y8fp12_v6-sip.bin
Result: Files match, 4998 bytes

Downloading file: 6y8fp12_v6.bin
Comparing with file: /tftpboot/6y8fp12_v6.bin
Result: Files match, 4856 bytes
```

Listing 14 – Jackhammer Stage 3 test suite vs. Sentinel Stage 3

```
[root@viezcnrat98 /]# diff /tftpboot/6y3fp7_v4.bin /sentinel/6y3fp7_v4.bin
[root@viezcnrat98 /]# diff /tftpboot/6y3fp7_v4-sip.bin /sentinel/6y3fp7_v4-
sip.bin
[root@viezcnrat98 /]# diff /tftpboot/6y3fp7_v6.bin /sentinel/6y3fp7_v6.bin
[root@viezcnrat98 /]# diff /tftpboot/6y3fp7_v6-sip.bin /sentinel/6y3fp7_v6-
sip.bin
[root@viezcnrat98 /]# diff /tftpboot/6y8fp12_v4-sip.bin /sentinel/6y8fp12_v4-
sip.bin
[root@viezcnrat98 /]# diff /tftpboot/6y8fp12_v4.bin /sentinel/6y8fp12_v4.bin
[root@viezcnrat98 /]# diff /tftpboot/6y8fp12_v6-sip.bin /sentinel/6y8fp12_v6-
sip.bin
[root@viezcnrat98 /]# diff /tftpboot/6y8fp12_v6.bin /sentinel/6y8fp12_v6.bin
```

Listing 15 – TFTP client and “diff” file comparison output

Jackhammer performance tests, console output, chapter 6.2.5, versus Sentinel, listing 16 and HPA TFTP, listing 17, as well as the re-run against Sentinel with performance enhancements in listing 18.

```
Jackhammer TFTP performance test:

Total runtime for 1 process with 100 downloads each, is 8.95 seconds

Total runtime for 10 processes with 100 downloads each, is 10.56 seconds

Total runtime for 100 processes with 100 downloads each, is 47.83 seconds

Total runtime for 500 processes with 100 downloads each, TEST FAILED

Total runtime for 1000 processes with 100 downloads each, TEST FAILED
```

Listing 16 – Jackhammer performance test vs. Sentinel with client verification

```
Jackhammer TFTP performance test:

Total runtime for 1 Processes with 100 downloads each, is 4.11 seconds

Total runtime for 10 Processes with 100 downloads each, is 5.71 seconds

Total runtime for 100 Processes with 100 downloads each, is 30.16 seconds

Total runtime for 500 Processes with 100 downloads each, is 174.72 seconds

Total runtime for 1000 processes with 100 downloads each, TEST FAILED
```

Listing 17 – Jackhammer performance test vs. HPA TFTP

```
Jackhammer TFTP performance test:

Total runtime for 1 process with 100 downloads each, is 7.62 seconds

Total runtime for 10 processes with 100 downloads each, is 9.91 seconds

Total runtime for 100 processes with 100 downloads each, is 47.83 seconds

Total runtime for 500 processes with 100 downloads each, is 238.49 seconds

Total runtime for 1000 processes with 100 downloads each, TEST FAILED
```

Listing 18 – Jackhammer performance tests vs. Sentinel with performance enhancements