UNIVERSITY OF OSLO
Department of Informatics

The future is back; IPv6 transition mechanisms

Joachim Tingvold
joachim@tingvold.com

Spring 2016

Abstract

The worlds consumption of IPv4 addresses has exhausted the RIRs resource pools of available IPv4 address space. By the end of 2015, every RIR, except AFRINIC, had allocated all their blocks not reserved for IPv6 transitioning. This has been anticipated since the late 1980s, and was one of the causes of making its successor protocol, IPv6. New and existing networks needs transition mechanisms to migrate away from IPv4, while at the same time maintain IPv4 connectivity. One of these mechanisms is MAP, Mapping of Address and Port, that makes it possible to have IPv4 connectivity to and across an IPv6 network. We'll look at how this mechanism affects the user quality of service, and see if there are any noticeable difference in performance using standard consumer CPE equipment.

Contents

List of Figures

List of Tables

LIST OF TABLES

List of Listings

1

Introduction

We surround ourselves with more and more devices that relies on being connected to each other, either directly or via the Internet. In 2005, around 19% of the world's population had access to Internet. By the end of 2015, almost 50% had access. Even in LDCs[1] we see a massive increase of people using the Internet, going from 1% in 2005, to almost 10% by the end of 2015 (1).

This expansion has led to the exhaustion of IPv4 addresses from the regional Internet registries (RIR) resource pools of available IPv4 address space (2). On February 3rd, 2011, Internet Assigned Numbers Authority (IANA) allocated the last available IPv4 pools to the RIRs (3)(4). By the end of 2015, every RIR except African Network Information Centre (AFRINIC), had allocated all their blocks not reserved for IPv6 transitioning. This has been anticipated since the late 1980s, and was one of the causes of making its successor protocol, IPv6. Containing 7.9×10^{28} times as many usable addresses as IPv4, it has room for future expansion.

The Internet is facing a huge change. IPv6 has to be deployed with the ultimate goal of not relying on IPv4 at all. Since these two protocols are not compatible with each other, transition mechanisms have to be used so that IPv4 connectivity is maintained while we migrate to IPv6.

This thesis will take a look at these mechanisms, with the focus being on MAP, Mapping of Address and Port.

[1] A least developed country (LDC) is a country that, according to the United Nations, exhibits the lowest indicators of socioeconomic development, with the lowest Human Development Index ratings of all countries in the world.

1.1 Motivation

My interest for IPv6 in general was the motivation for writing this thesis. The original concept of the Internet was that each end-device should be able to communicate directly with each other. Having too few addresses available with IPv4, combined with the rapid growth of the Internet, concepts such as Network Address Translation (NAT) and Port Address Translation (PAT) was introduced, which broke this fundamental goal of having end-to-end connectivity. The introduction of IPv6 solves this, but brings along several challenges that I find interesting.

1.2 Overview

This thesis will introduce the different IPv6 transition mechanisms in chapter 2. An overview of what considerations needs to be taken in the transition to IPv6, which challenges there are, and how to address them, will also be discussed. In chapter 3 we'll look in detail on the chosen technologies. Thereafter, in chapter 4, we'll look at the implementation of our scenario, followed by test results in chapter 5. We make our conclusion in chapter 6.

2

Scenario

IPv6 was introduced almost two decades ago in 1998 (5). When the RIRs entered the last stage of IPv4 exhaustion in 2011, the worldwide deployment of IPv6 was below 0.5%. Even today, 5 years later, that number has only increased to about 5% (6). There are many explanations as to why the deployment is slow (7)(8)(9). Previously it was mostly due to technical challenges like vendor support, client support, etc. This is no longer the case, and the biggest challenges (incentive, cost, competence, etc.) boils down to the fact that IPv6 is not backwards compatible with IPv4. This means that an IPv4-only node can't communicate with an IPv6-only node, and the other way around. Due to this, we cannot deploy IPv6-only in some places, without thinking about IPv4. Even if clients get IPv6 connectivity, they still need some kind of IPv4 connectivity to be able to communicate with non-IPv6 clients.

To get an overview of the different kind of deployment scenarios, we can take a look at Figure 2.1. We have the single-stack IPv4 that most network operators have, IPv6-over-IPv4, dual-stack, IPv4-over-IPv6, and single-stack IPv6. There are some different flavors of these categories, e.g. that scenario *a)* might use CGN, and therefore utilize shared address space (or private) in the provider network, hence only having public IPv4 towards the Internet. However, the figure outlines the basic principle of the different deployment methods, and the different transition mechanisms spans one or more of these scenarios in some way.

In the following sections we'll describe the different transition mechanisms and how they work. At the end of the chapter we'll look at some of the challenges.

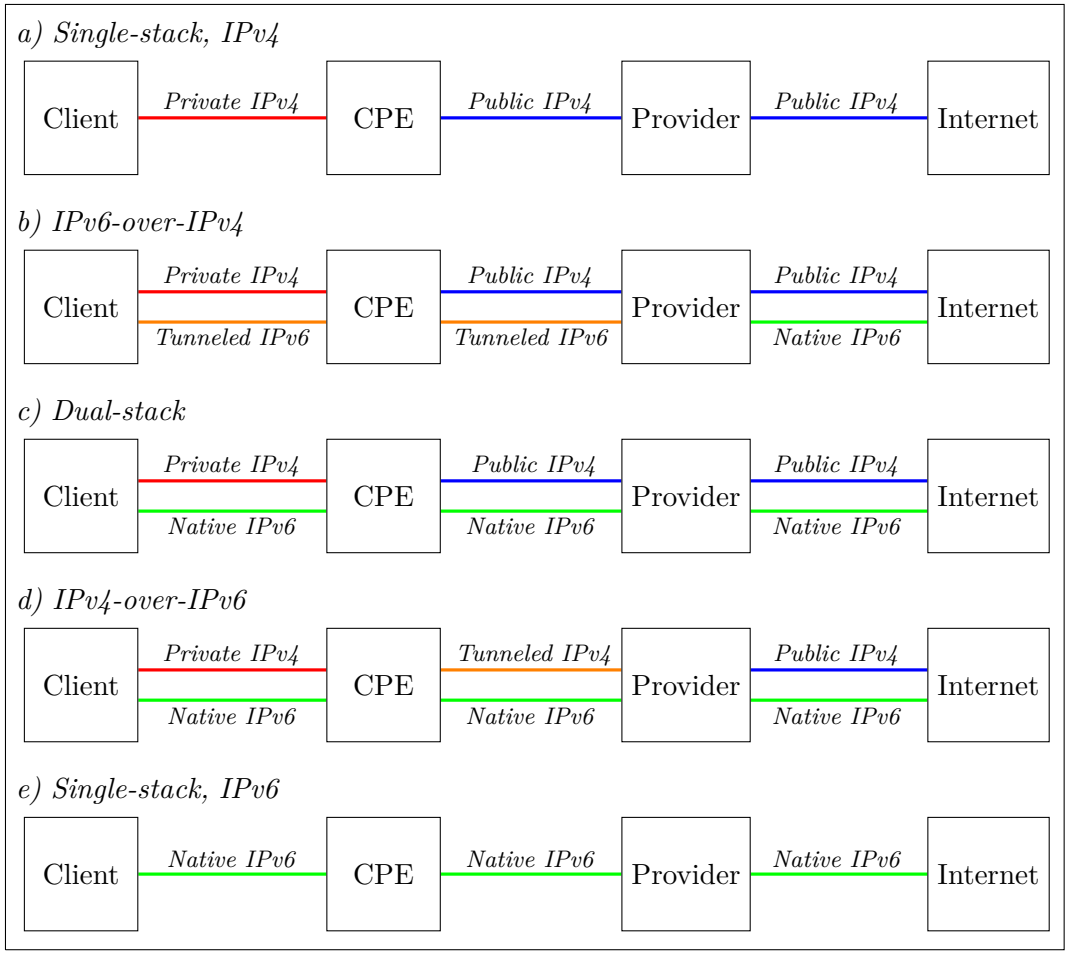

Figure 2.1: IP deployment scenarios - A simple overview of the different IP deployment scenarios. All of the transition mechanisms falls into one or more of these scenarios.

2.1 Deployment of IPv6

Internet service providers (ISP) and network operators that have existed for a while typically have IPv4 deployed everywhere in their network. Customers are assigned one or more public IPv4 address, which is used to either connect devices directly to the Internet, or as shared access where multiple devices share the same public IPv4 address. The latter has prolonged the life of IPv4, and is achieved with the use of Network Address and Port Translation (NAPT, commonly referred to as NAT, NAT44, PAT, or NAT overload), where a set of private IPv4 address space shares one public IPv4

address (10)(11)(12).

Some network operators have also implemented double NAPT – commonly referred to as Carrier-grade NAT (CGN), large-scale NAT (LSN), or just NAT444) – utilizing private address space together with shared address space (12)(13), even further prolonging the life of IPv4. However, the use of CGN introduces different problems regarding connectivity, geo-location, logging, among other things (14)(15)(16), even though it's proven to work fine for normal web browsing (17).

It's also been suggested to reclassify some of the reserved IPv4 ranges, so that the life of IPv4 can be prolonged (18). However, it wouldn't help for long, and as such, was never implemented.

The issues with NAPT, together with the fact that we are out of public IPv4 address space, makes implementing IPv6 inevitable. Some of the early adopters of IPv6 did it because they were out of RFC1918 address space for use internally in their data centers. Facebook observed that IPv6 also performs better than IPv4; after they switched to IPv6-only in their data centers a few years ago, they noticed about 15% increased performance over IPv6 compared to IPv4 (19)(20), which interestingly enough seems to contradicts other observations (21)(22)(23)(24).

The long term goal is therefore one single protocol, IPv6, but for the short term we need to rely on IPv4, and therein lies the challenge; how can existing network operators and ISPs scale their networks? And even more importantly; how can new network operators and ISPs build new networks, having scarce IPv4 resources?

2.1.1 Dual-stack

The ideal deployment is to use dual-stack, where IPv4 and IPv6 are deployed side-by-side, and all the routers and devices in the network gets both IPv4 and IPv6 connectivity. This is the best from both worlds; you get native IPv4 *and* IPv6 connectivity. Most, if not all, devices support this, and it's the least complex solution configuration-wise. When IPv4 becomes obsolete at some point, you could remove it, without affecting IPv6 connectivity.

However, since there are no more IPv4 address space available, deploying dual-stack implies that you either have enough IPv4 addresses available that you own, or plan to buy IPv4 address space. The latter has created a somewhat grey marked where network operators sells off some of their IPv4 allocations that they don't use. The Norwegian

ISP Altibox recently took advantage of such a situation, and paid about £600.000 for 150.000 IPv4 addresses that the UK Government didn't need (25). Back in 2011, Microsoft paid $7.5 million USD for 666.624 IPv4 addresses formerly owned by Nortel (26).

Deploying dual-stack is therefore not a scalable solution for most network operators, and especially not for startup networks; applying for IPv4 resources at the European RIR, Réseaux IP Européens Network Coordination Centre (RIPE NCC), as a new local Internet registry (LIR), you're only given one /22 of IPv4 address space (27). In a dual-stacked environment that would mean at most 1024 customers, which doesn't scale really well.

2.1.2 Transition mechanisms

The need for some kind of transition mechanisms therefore emerged, making it possible to deploy IPv6, while at the same time maintain IPv4 connectivity. In existing networks, the goal of the transition mechanisms is mainly to assist the transition to IPv6, while for new networks it's to give IPv4 connectivity over an IPv6-only network (since, for new networks, it makes sense to deploy it as IPv6-only, and then add "legacy" support in the form of IPv4 on top of that). Some of the transition mechanisms also help in utilizing the IPv4 address space, since by offloading more and more traffic to IPv6, more devices can share a single public IPv4 address.

There are several transition mechanisms, and over the years, the number has increased. Many of them are iterations of old mechanisms and/or incorporates more than one of the previous mechanisms. Figure 2.2 shows most of them, and how they are related to each other.

Historically, tunnel brokers have been the way people got IPv6 access where dual-stack was not available. This involved manually configured tunnels to a nearby point of presence (PoP) that had native IPv6 connectivity. There are several tunnel brokers that are still operational today, where Hurricane Electric and SixXS are the largest and well-known of them (29). They have PoPs all over the world, and due to the slow rollout of IPv6, they are still popular to this day. SixXS even experiences that ISPs redirects their paying customers to the SixXS service if they ask for IPv6 (30).

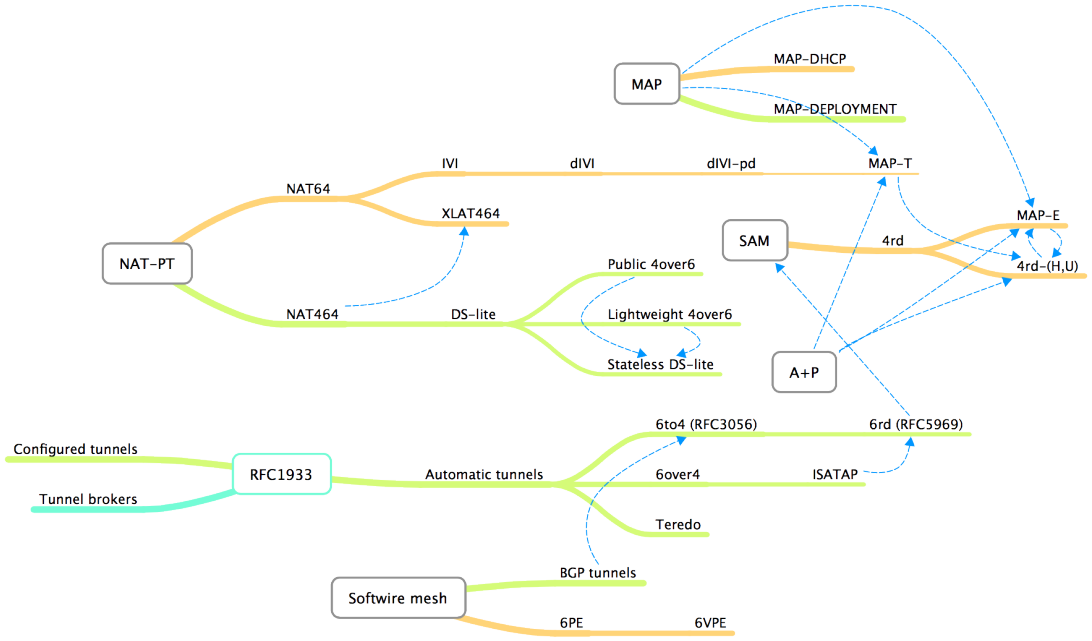

Figure 2.2: Transition mechanisms - Overview of most of the transition mechanisms available, and how they relate to each other (28).

Nejc Škoberne et al. in (16) did a thorough job at classifying the different transition mechanisms with the focus being on IPv4 address sharing. They divided them into classes, based on five dimensions;

1. Location of the IP address sharing function (CPE or Provider).
2. State storage in the gateway (per flow, per allocation or stateless).
3. Traversal method through the access network (routing, tunneling, translation).
4. Level of IPv6 requirement (required, partly required, not required).
5. IPv4 address and port allocation policy (static or dynamic).

Based on this, they looked at how the mechanism tradeoffs could be identified using the transition methods available. It boils down to five choices;

1. *Address+Port or CGN:* A+P gives end-to-end connectivity and scalability, and is cost-reducing for the ISPs, but is not widely available on CPEs. CGN is a well-known technology that is mature, and doesn't require specific functionality on the CPEs.

2. *Stateful or stateless:* Stateful methods gives flexible IP addressing, efficient IPv4 sharing, and supports scattered address-space. Stateless makes CPE-to-CPE communication possible without traversing large parts of the network (to reach the provider gateway), gives easy load-balance and high-availability since no state-synchronization is needed, and doesn't need expensive equipment to keep track of states.
3. *Tunneling or double translation:* Tunneling is a mature and widespread method and keeps the IPv4 packets intact. Double translation is not prone to the routing loops and MTU issues that tunnels are, and also requires less processing in the path between CPE and gateway.
4. *IPv6 required or IPv6 not required:* IPv6 required means one less protocol to configure in the network, which is easier to administer. With IPv6 not required, it's easier to deploy on existing networks (since IPv6 doesn't need to be deployed).
5. *Static port/address allocation or dynamic port/address allocation:* Static allocations makes it possible for many customers to share one single public IPv4 address, while at the same time maintain easy and efficient logging of which customer uses what port. Dynamic allocation allows for even more customers to share one single public IPv4 address, and is also considered more secure (since the ports are random, and not static).

We'll take a quick look of the transition mechanisms below, where we have put them into three generic categories.

2.1.2.1 Single translation

In single translation methods, there is only one translation done; either at the CPE, or somewhere in the operator's network.

- **NAT64 + DNS64** in combination provides a way to facilitate IPv6 in a single-stack environment, but at the same time maintain connectivity to IPv4 devices. DNS64 works by synthesizing AAAA records when only A records are returned. It embeds the address of an IPv6/IPv4 translator (a NAT64 server) together with the original A record. There are no translations before the NAT64 server, which means native IPv6 single-stack all the way out to the clients. The solution

only works when the application supports the use of IPv6, and DNS is used to resolve the remote address. In cases where IPv4 are statically used, or where the application has no knowledge of IPv6, it does not work. It also breaks DNSSEC. NAT64 is defined in RFC6146 (31), while DNS64 is defined in RFC6147 (32).

2.1.2.2 Double translation

The difference with double translation methods, compared to single translation methods, is the need for an extra translation device as part of the operator's network. This means translation finds place at the CPE, and somewhere in the operator's network.

- **464XLAT** is similar to NAT64 in the sense that clients only receive an IPv6 address. All devices need a software, CLAT, which is a customer-side translator, hence it's not really intended for residential or business networks behind CPEs – the primary usage is for mobile networks on devices like cellphones, tablets, etc. If IPv4 connectivity is needed, the CLAT provides a private IPv4 address with a default gateway, which will be translated to IPv6, and then back to IPv4 in the provider network. It solves the issue with NAT64+DNS64 where applications without IPv6-awareness doesn't work. Defined in RFC6877 (33).
- **MAP-T** is a stateless mechanism. CPEs are assigned a public IPv4 address and a set of ports, which it uses to do NAPT44. The CPE then uses stateless NAT46 to translate into a IPv6 packet and sends it to the border relay (BR) which does stateless NAT64, and forwards it. Since the IPv4- and port-mappings is algorithmically, the BR knows where to send returning packets without keeping any states. Defined in RFC7599 (34).

2.1.2.3 Encapsulation

In encapsulation methods, the network traffic is encapsulated either over IPv4 or IPv6. The traffic passes through at least two devices; encapsulator and decapsulator. The overhead is larger in encapsulation methods, but keeps the original IP packet intact (increasing support for security measures, QoS, etc.).

- **4in6** is plain IPv4-in-IPv6 using statically configured tunnels. It provides a way to give IPv4 connectivity over IPv6 only networks. This can be combined with NAT44 or NAT444. Defined in RFC2473 (35).

- **DS-Lite** was conceived prior to MAP. It consists of IPv4-over-IPv6 tunnels between the CPE (known as Basic Bridging Broadband Element, or B4), and a massive, stateful tunnel concentrator known as Address-Family Transition Router (AFTR). The B4 router encapsulates IPv4 packets, which is decapsulated at the AFTR. The CPE does not do NAT44, hence DS-lite is a NAT44 solution (i.e. only single NAT, which is done by the AFTR). The solution is stateful, and also requires CPE-to-CPE traffic to traverse the AFTR. Defined in RFC6333 (36).
- **Lightweight 4over6** functions similar to DS-lite, but moves the NAT from the AFTR down to the CPEs, making the AFTR state go from a per-flow basis, to a per-subscriber basis. It does not solve the issue of CPE-to-CPE connections. Defined in RFC7596 (37).
- **MAP-E** is a stateless alternative to CGN and DS-lite. CPEs are assigned a public IPv4 address and a set of ports, which it uses to do NAPT44. The CPE then encapsulates the packet into an IPv6 packet, and sends it to the BR which decapsulates it and forwards it as-is (since it already has the public IPv4 address as the source, done by the CPE). Since the IPv4- and port-mappings is algorithmically, the BR knows where to send returning packets without keeping any states. Defined in RFC7597 (38).
- **4rd**, or IPv4 residual deployment, is similar to MAP. It solves some of the issues, but is different implementation-wise compared to MAP-E and MAP-T. It's still in the experimental track, defined in RFC7600 (39). Free, a French ISP, has been testing 4rd in smaller deployments on their FTTH subscribers since December 2015 (40).

There are several mechanisms to provide IPv6 connectivity over IPv4-only networks. These are not considered in this thesis, but they are mentioned as they have been deployed for many years to provide IPv6 connectivity around the globe, and hence, participated in the transition to IPv6.

- **6in4** is plain IPv6-in-IPv4 using statically configured tunnels. It provides a way to give IPv6 connectivity over IPv4 only networks. This is the method usually used by tunnel brokers. Defined in RFC2893 (41), which obsoletes RFC2893 and RFC1933.

- **6over4** is similar to 6in4 in which it utilizes proto-41, but differs in the way that there is no need to statically configure tunnels, and that it utilizes IPv4 multicast. Defined in RFC2529 (42).
- **ISATAP** is similar to 6over4, but does not rely on IPv4 multicast. Defined in RFC5214 (43).
- **6to4** assigns a block of IPv6 space to all public IPv4 addresses, and needs to be configured on the device. It also utilizes proto-41, but there is no need to statically configure tunnels. It's been prone to misconfiguration, so an advisory explaining the different caveats has been made (44). Defined in RFC3056 (45).
- **Teredo** builds on the same principle that 6to4 does, but does not rely on devices or networks having public IPv4 addresses. It can traverse NAT. Defined in RFC4380 (46).
- **6rd**, or IPv6 rapid deployment, is derived from 6to4, and its difference is mainly that the relays are placed internally in the operators network, and can only be used by the users of that network (i.e. same administrative entity). It is the most deployed IPv6-over-IPv4 solution amongst ISPs (47). The first major implementation happened in 2007, where Free, a French ISP, deployed it in their entire network in just five weeks. At the time, they had over 4 million subscribers. (48)(49).

2.2 Challenges

We acknowledge that there are many aspects to take into consideration when transitioning to IPv6, and that the choice of what transitioning mechanism to choose will vary between network operators based on their current network setup. With this in mind, we want to look at the mechanism that we find most suited for new network operators with limited IPv4 resources, and where a IPv6-only core- and distribution network is the basis for operation. We'll look at how to deploy this mechanism, what limitations it has, and how the experience for the end-users are regarding usability (i.e. what works/doesn't work), throughput and latency. We also want to utilize commodity hardware, especially regarding the CPE.

In the next chapter, we'll look more closely into the technologies we've chosen to address these challenges with.

3

Technologies

According to the classifications outlined in the previous chapter, we find that MAP-T is the best suitable transition mechanism for several reasons;

1. It utilizes A+P, rather than CGN.
2. It's stateless, which is more fault-tolerant, and doesn't require expensive equipment to do states.
3. It's double translation, meaning that we avoid tunneling, and also reduce the processing of packets in the network path from the CPE to the gateway.
4. It relies only on IPv6, making it possible to have an IPv6-only distribution- and core network, which, in turn, makes it easy to un-deploy it whenever IPv4 becomes obsolete.
5. It utilizes static port allocation, meaning that end-users can actually host services at home (e.g. SSH). Since the port allocation is centralized (via DHCPv6-options), the operator can dynamically change the number of devices sharing one public IPv4 address without doing changes in their network (other than on the DHCPv6 server and the BR), making it possible to put more and more devices behind one public IPv4 address as the need for IPv4 connectivity is reduced (thus making it possible to expand without the need for more IPv4 address space).

Marius Georgescu et al. in (50) has also looked at the security of the different transition mechanisms, and based on this, we find that MAP-T is as secure as the alternatives. Static port allocation could be a security issue, but MAP-T implements

port allocation offsets that makes guessing the port-ranges for specific end-users harder. We'll look more at this in the next section.

Guo-liang Han et al. in (51) suggests a more scalable IPv4 sharing solution than MAP, where a hybrid between A+P and CGN is considered. This allows for more devices sharing a single IPv4 address. We believe that it's only needed for extremely large providers that has large amounts of users, where the number of users that can share one public IPv4 address matters. In our case, we still find MAP-T to be a better suited mechanism, and is also a simpler mechanism that doesn't rely on states or complexity on the CPEs.

Roberta Maglione et al. in (52) also gives several examples of scenarios where MAP-T is beneficial over MAP-E.

The only downside with MAP-T, that we could find, is an issue with path MTU Discovery. Since MAP-T does NAT64, any IPv4-originated ICMP-independent Path MTU Discovery, as specified in RFC4821 (53), ceases to be entirely reliable. This is because the DF=1/MF=1 (defined in RFC4821) results in DF=0/MF=1 after a double NAT64 translation. As long as there is a consistent MTU within the MAP domain, this shouldn't really have any particular impact for end-users.

In the following sections we'll look at how MAP works, how the CEs can be provisioned, and what network equipment we'll use, before closing of with the different software being used.

3.1 MAP

Mapping of Address and Port, or MAP, is not a new concept. It has been described in many mechanisms before, primarily in IPv6-over-IPv4 mechanisms. This includes, but is not limited to, 6over4, 6to4, ISATAP and 6rd. In common, they all facilitate automatic provisioning of an IPv6 address for a host, or a prefix for a site. They also provide ways to determine the IPv4 endpoint of a tunnel, given the IPv6 destination address. With MAP, the tables have turned compared to the earlier implementations, since we now want to facilitate IPv4 connectivity in IPv6-only environments.

MAP utilizes a mapping algorithm for port indexing and IPv4 to/from IPv6 address conversion. It's a stateless mechanism, which gives us scalability and good redundancy.

The CPE uses stateful port-restricted NAT44, but it's still considered a stateless mechanism since there is no need for state synchronization or central state keeping. It then uses stateless NAT64 translation or stateless encapsulation to carry the IPv4 information over an IPv6-only network.

MAP has evolved from NAT64 and IVI, and undergone several drafts in the Softwires Working Group of Internet Engineering Task Force, IETF (54). MAP has two operating modes; translation and encapsulation. The Working Group had several discussions whether to make one or two tracks. It was decided by throwing a coin, and the result was two tracks (55). MAP-E had 13 drafts, and MAP-T had 8, until they became RFC7597 and RFC7599 respectively in July, 2015. Currently, MAP (both MAP-E and MAP-T) does not support multicast over IPv4. The Softwires Working Group is working on how to solve this challenge (56). This is outside the scope of this thesis as we find IPv4 multicast to be of little importance to end-users (with the exception of IPTV in triple-play settings, but then the network operator can facilitate their IPTV services over IPv6 multicast).

3.1.1 Terminology

A MAP network contains Customer Edge (CE, or CPE) and Border Relay (BR) routers, as illustrated in Figure 3.1. It also contains a set of parameters and rules, which define how MAP should operate. These parameters can be configured manually via the CLI (usually done on the BRs), or automatically using protocols such as DHCP (usually on the CE nodes). MAP supports both mesh and hub-and-spoke topologies, meaning that CEs can send traffic destined to another CE directly, without going through the BRs.

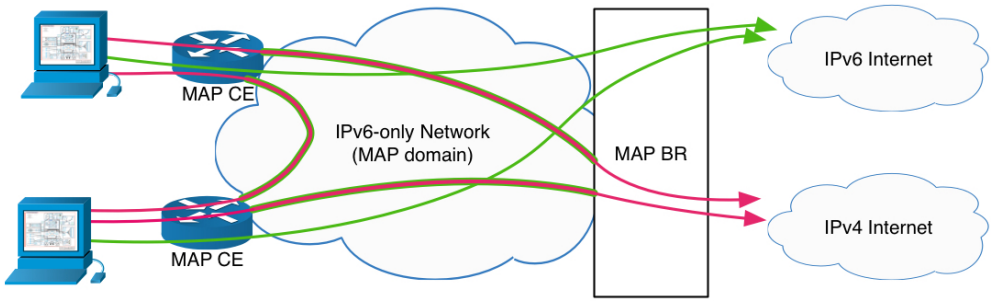

Figure 3.1: MAP illustration - Simple overview of a MAP network.

- A *MAP domain* consists of one or more CE devices and BRs utilizing the same MAP ruleset. A network may be comprised of several MAP domains, even within the same administrative entity.
- One or more *MAP rules* defines the forwarding behavior for a MAP domain, and the mapping between an IPv4 address or prefix, and the IPv6 address.
- A *MAP rule set* or Mapping Rule Table (MRT) is made up of one or more MAP rules, and effectively serves as the routing table for the BR and CE to make forwarding decisions..
- *MAP Border Relay (BR)* is a MAP-enabled router that is managed by the network operator, and defines the edge of the MAP domain. The BR has at least one IPv6-enabled interface, and an IPv4-enabled interface with a native IPv4 address. In the context of MAP, we usually refer to it as "BR".
- *MAP Customer Edge (CE)* is a MAP enabled router placed with the customer. The CE has one IPv6-only interface towards the network, and one or more LAN interfaces facing the customer devices. In the context of MAP, we usually refer to it as "CE" or "CPE".
- A *port set* defines what part of the transport-layer port space a CE utilizes. Each CE has its own separate part of ports.
- The *Port Set ID (PSID)* algorithmically defines the port set assigned to a CE.
- A *shared IPv4 address* is an IPv4 address shared among multiple CEs.
- The *end-user IPv6 prefix* is the prefix assigned to the CE by other means than MAP, and is unique for each CE. The prefix assignment can be done with static configuration, DHCPv6 Prefix Delegation (57), or Stateless Address Autoconfiguration; SLAAC (58).
- The *MAP IPv6 address* is used to reach the MAP function of a CE from other CEs and BRs.
- *Rule IPv6 prefix* is assigned by a network operator for a mapping rule, and defines the prefix that the CEs are a part of.

- *Rule IPv4 prefix* is assigned by a network operator for a mapping rule, and defines the prefix of IPv4 addresses that the CEs will share.
- The *Embedded Address (EA) bits* identifies an IPv4 prefix or address (or part of) and a PSID.

3.1.2 MAP-E

Since MAP-T shares most of the inner workings of MAP-E, we'll look at how MAP-E works, and then follow up with explaining what MAP-T does different.

MAP-E utilizes existing building blocks. It uses the existing Network Address and Port Translator, NAPT (10), on the CE, with additional support for restricting the transport-protocol ports, ICMP identifiers and fragment identifiers to the assigned port set. When crafting outbound packets, the NAPT is restricted to use ports within the CEs assigned port range. In turn, the packets are encapsulated using the mode specified in RFC2473 (35).

For full IPv4 addresses, or 1:1 mapping, the restrictions of the transport-protocol ports do not take place. This means that you could utilize MAP as a way of doing "dual-stack" with a IPv6-only distribution- and core network, yielding the same functionality of dual-stack (i.e. all 65536 ports available to the users). When the time comes when the operator has no more IPv4 address space, they can activate IPv4 address sharing without changing the setup on their distribution- and core networks, making MAP a really scalable and future-proof transition mechanism.

3.1.3 Mapping algorithm

One or more mapping rules define the forwarding behavior of a MAP domain. Every MAP node must be provisioned with a Basic Mapping Rule (BMR). It's used by the node to configure the IPv4 address or prefix. A node can also have one or more of the optional Forwarding Mapping Rule (FMR). Several mapping rules can be specified to allow for multiple IPv4 subnets to exist within the same MAP domain, and to optimize forwarding between them (without the need to go through the BR). The BMR and FMR have the same parameters;

- Rule IPv6 prefix (including prefix length)

- Rule IPv4 prefix (including prefix length)
- Rule EA-bit length (in bits)

MAP nodes finds their BMR by doing a longest match between end-user IPv6 prefix and the rule IPv6 prefix in the MRT. The rule is then used to assign it's IPv4 address or prefix. From the end-user IPv6 prefix, the node forms its MAP IPv6 address that is assigned to one of its interfaces. The node uses this address to terminate all MAP traffic being sent or received.

For each FMR, a port-restricted IPv4 route is installed. A default route is also added, which is for all destinations outside the MAP domain. The FMRs allow direct communication between MAP CEs. This is known as mesh mode. In a hub-and-spoke mode there are no FMRs, and all traffic is forwarded to the BRs.

3.1.3.1 Basic Mapping Rule (BMR)

The Basic Mapping Rule (BMR) is mandatory. It consists of the aforementioned parameters; IPv6 prefix, IPv4 prefix, EA-bit length. The CE can have multiple BMRs, but there can only be one BMR per end-user IPv6 prefix. All CEs within the same (aggregated) rule IPv6 prefix must use the same BMR. The structure of the IPv6 address format is shown in Figure 3.2.

n bits	o bits	s bits	$128 - n - o - s$ bits
rule IPv6 prefix	EA bits	subnet ID	interface ID
← end-user IPv6 Prefix →			

Figure 3.2: MAP IPv6 Address Format - Shows the structure of the complete MAP IPv6 address.

The EA bit field, which is unique for each rule IPv6 prefix, encodes the CE-specific IPv4 address and port set. It contains a full or partial IPv4 address, and if doing address sharing, a PSID is also included. If the EA bit field length is set to 0, it signals that all relevant IPv4 addressing information is passed directly in the BMR (and thus, it's not derived from the EA bit field in the end-user IPv6 prefix).

If the subnet identifier is > 0, the MAP IPv6 address is created by concatenating the end-user IPv6 prefix with the subnet identifier and the interface identifier as per Figure

3.3. The first subnet is to be used, so all the s bits is set to zero. If the subnet identifier is 0 bits long, only the end-user IPv6 prefix and interface identifier is concatenated.

$128 - n - o - s$ bits		
16 bits	32 bits	16 bits
0	IPv4 address	PSID

Figure 3.3: MAP IPv6 Interface Identifier - Shows the interface identifier format of a MAP node.

If we use an IPv4 prefix, the IPv4 address field has to be right-padded with zeros, up to a total of 32 bits. As for the PSID field, we left-pad that with zeros, and if we use an IPv4 prefix or complete IPv4 address, we set all 16 bits to zero. Technically it's possible to use end-user IPv6 prefix lengths longer than 64, but that would e.g. break SLAAC, and is therefore not recommended. In such a case, the largest prefix supported is 112 ($64 + 48$, since 48 is the largest size of the EA bit field), but that leaves out embedding the IPv4 address and PSID in the IPv6 address. If those are to be embedded, the largest prefix supported is 80 ($64 + 16$), and the most significant parts of the interface identifier are overwritten by the prefix.

If we define that $\boldsymbol{o}$ = the length of the EA bit field as given by the MBR, $\boldsymbol{p}$ = the length of the IPv4 suffix contained in the EA bit field, and $\boldsymbol{r}$ = the length of the IPv4 prefix as given by the BMR. The length of $\boldsymbol{r}$ can be zero, and if so, the complete IPv4 address or prefix is encoded in the EA bits (in which case $o \geq 32$). However, if only a part of the IPv4 address or prefix is encoded in the EA bits, the rule IPv4 prefix has to be provisioned by other means (DHCPv6 option, TR-69, manually configured, etc). In such cases, to create the complete IPv4 address or prefix, the IPv4 address suffix ($\boldsymbol{p}$) from the EA bits is concatenated with the rule IPv4 prefix ($\boldsymbol{r}$).

The EA bit field is offset equal to the length of the BMR rule IPv6 prefix, and the length of the EA bit field ($\boldsymbol{o}$) is given by the BMR, and $0 \leq o \leq 48$. If $o = 48$, the complete IPv4 address and port is embedded in the end-user IPv6 prefix, and if $o = 0$, no part of the IPv4 address or port is embedded in the address.

To illustrate, if $o + r < 32$, then an IPv4 prefix is assigned, and PSID is not set, as shown in Figure 3.4 **a)**. If $o + r = 32$, then a full IPv4 address is assigned (with no

PSID set). The IPv4 address is created by concatenating the rule IPv4 prefix and the EA bits. This is also shown in Figure 3.4 **a)**.

a)

r bits	o bits $= p$ bits
rule IPv4	IPv4 address suffix

b)

r bits	p bits	q bits
rule IPv4	IPv4 address suffix	Port Set ID

Figure 3.4: MAP EA-bits - Shows the two different versions of EA bits, where **a)** is when $o + r \leq 32$ and **b)** when $o + r > 32$.

If $o+r > 32$, then a shared IPv4 address is to be assigned, where $p = (32-r)$. The remainder of the bits defines the PSID, $(q = o-p)$. We can see this in Figure 3.4 **b)**. r can be 32, where no part of the IPV4 address is embedded in the EA bits, resulting in a mapping with no dependency between the IPv4 address and the IPv6 address. o can also be 0, where no EA bits are embedded in the end-user IPv6 prefix, and as such, the PSID has to be provisioned otherwise. An overview of the EA bits and port mapping is seen in Figure 3.5.

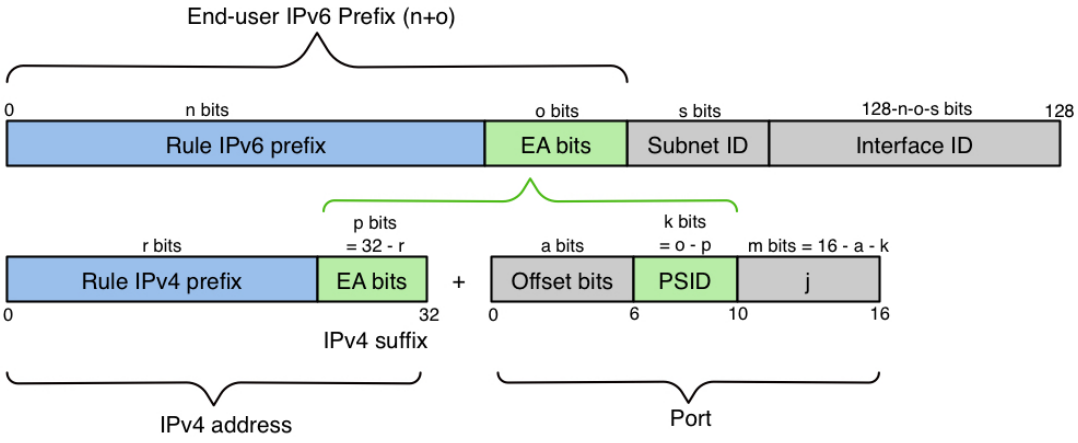

Figure 3.5: MAP parameter overview - Shows an overview of the MAP parameters.

3.1.3.2 Forward Mapping Rule (FMR)

The Forwarding Mapping Rule (FMR) is optional, and is used for forwarding. It is used in mesh mode to facilitate direct CE-to-CE connectivity. A BMR can also be a FMR. Each FMR results in an entry in the MTR for the rule IPv4 prefix. A MAP node can, given a destination IPv4 address and port, use the matching FMR to calculate the end-user IPv6 address, and as such, send a packet directly to the CE without going through the BRs. In a hub-and-spoke mode there are no FMRs, and all the traffic has to go through the BRs.

3.1.3.3 Destinations outside the MAP-E domain

All IPv4 traffic between MAP nodes in the same MAP domain is encapsulated in IPv6. The senders MAP IPv6 address is the source IPv6 address, and the receiving MAP nodes MAP IPv6 address is the IPv6 destination address. Traffic destined for an IPv4 address outside the MAP domain is also encapsulated, but the IPv6 destination address is set to the IPv6 address of the MAP BR (which can be an anycasted IPv6 address for redundancy). The BR decapsulates the received packet, and forwards it. If the IPv4 destination is within the MAP domain (but no FMRs are defined), the packet is treated as if it came from the "outside", hence encapsulated according to the MRT.

3.1.4 Port-mapping algorithm

The port-mapping algorithm used in MAP, Generalized Modulus Algorithm (GMA), allows for IPv4 address sharing by assigning a specific port set to each CE that are non-overlapping with other CEs sharing the same IPv4 address. The algorithm is defined such that the port set is derivable from the PSID embedded in the end-user IPv6 prefix, and reversible so that given the port number, the PSID can be calculated. It allows for a broad range of address-sharing ratios, where subsets of the port numbering space can be excluded (which e.g. allows us to exclude the system ports $0 - 1023$).

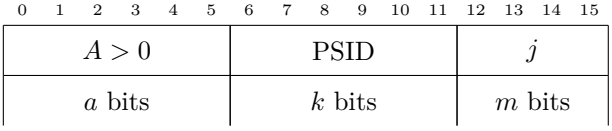

Figure 3.6: Port-Restricted Port Field; PSID - Shows the structure of the PSID.

Figure 3.6 shows the structure of the PSID.

- **a bits** are the number of offset bits. It's set to 6 by default, as this excludes the system ports $(0-1023)$. The offset has to be the same for every MAP CE sharing the same IPv4 address, so that non-overlapping ports are guaranteed.
- **A** selects the range of the port number, and if $a > 0$, A has to be > 0. Selecting a smaller value for a excludes a larger initial port range. The interval between port ranges is given by $2^{(16-a)}$.
- **k bits** is the length in bits of the PSID field. The length of k has to be the same for every MAP CE sharing the same IPv4 address, so that non-overlapping ports are guaranteed. The sharing ratio is given by 2^k, and the total number of assigned ports to the CE is given by $2^{(16-k)} - 2^m$.
- Different **PSID** values are guaranteed non-overlapping port sets due to the restrictions on a and k.
- **m bits** defines the number of contiguous ports, and is given by 2^m.
- **j** defines a specific port within a particular range by concatenation of A and the PSID.

To summarize, we can now calculate the properties of a MAP domain as follows;

- The total number of consumed IPv4 addresses is defined by $2^{(32-r)}$.
- The maximum number of CEs that can use the same BMR is defined by 2^o.
- The number of CEs that share on a single IPv4 address is defined by 2^k.
- The number of ports per range (x) for a given CE is defined by $x = 2^m$.
- The number of ranges (y) for a given CE is defined by $y = 2^a - 1$.
- The total number of ports (z) for a given CE is defined by $z = x \times y$.
- The upper port (v) that is reserved is defined by $2^{(16-a)} - 1$, where all the ports reserved $V \in [0, v]$.
- The low-port (P_{low}) of port range number Y (where $Y \leq y$) is defined by $P_{low} = 2^{(16-a)} \times Y$. The high-port (P_{high}) of the same range Y is defined by $P_{high} = 2^{(16-a)} \times Y + x - 1$.

3.1.5 MAP-T

MAP-T shares most of the building blocks of MAP-E. Where MAP-E uses encapsulation and decapsulation to carry the IPv4 packet between the CEs and BRs, MAP-T uses a stateless NAT64 function which is extended to allow stateless mapping of the IPv4 address and ports to IPv6 (59)(60). Ergo, the main difference between MAP-E and MAP-T boils down to encapsulation/decapsulation vs. double translation. In an encapsulation situation, the extra overhead is typically 40 bytes, while in a translation situation, the extra overhead is only 20 bytes. In either case, extra care has to be given when deploying MAP, such as to ensure that no fragmentation occurs within the boundary of the MAP domain.

3.1.5.1 Default Mapping Rule (DMR)

MAP-T introduces a third rule, in addition to the BMR and FMR, called Default Mapping Rule (DMR). It is used for destinations outside the MAP domain. Destinations outside the MAP domain is represented using an IPv4-embedded IPv6 address using the IPv6 prefix configured at the BR. This method is defined in RFC6052 (61). The CE will use the DMR to populate its routing table with a default IPv4 route.

When the CE sends packets to IPv4 destinations outside the MAP domain, the source address will be the CEs MAP IPv6 address, and the destination will be the IPv4-embedded IPv6 address. The BR announces the whole DMR prefix, and hence, all destinations will end up at the BR, which will translate and forward the packet.

The IPv6 prefix used as DMR will by default be 64 bits long, as defined in RFC6052 (61). It can be longer, but must not exceed 96 bits. Any trailing bits after the IPv4 address is set to 0x0.

3.1.5.2 Destinations outside the MAP-T Domain

Destinations outside the MAP-T domain utilizes the DMR defined in the previous section.

3.1.6 DHCPv6

For a node to function in a MAP domain, it needs to be provisioned with the appropriate rules and parameters, as discussed in the previous sections. On the BR this is usually

done manually via the CLI, but that would be time-consuming to do on the CEs. Since it's normal to delegate the end-user IPv6 prefix by the means of DHCPv6 Prefix Delegation as defined in RFC3633 (57), we find it natural to use DHCPv6 to provision the CEs with the MAP parameters. This also gives us the extra benefit that the provisioned MAP parameters follows the lifetime of the delegated IPv6 prefix, making it possible to easily change the MAP parameters on-the-fly (e.g. to increase the sharing ratio of an IPv4 address).

The DHCPv6 options relating to MAP-E and MAP-T is defined in RFC7598 (62). We see that both MAP-E and MAP-T uses the OPTION_S46_RULE option to define BMRs and FMRs, and the OPTION_S46_PORTPARAMS option to specify parameters for the port mapping algorithm. The CEs also needs a way to communicate outside the MAP domain, which for MAP-E means the IPv6 address for the BR, defined in option OPTION_S46_BR. For MAP-T this means the DMR, which is defined in option OPTION_S46_DMR.

3.1.6.1 S46 Rule Option

The S46 Rule Option (OPTION_S46_RULE) is used to convey BMRs and FMRs, and is used by both MAP-E and MAP-T. The format of the rule is shown in Figure 3.7, and the fields are defined as follows;

- *option-code*, 16 bits, defines the DHCPv6 option code 89, OPTION_S46_RULE.
- *option-length*, 16 bits, defines the length of the entire option expressed in octets, excluding the option-code and option-length fields, but including the length of all encapsulated options.
- *flags*, 8 bits, defines the flags applicable to the rules contained in the option. The structure is illustrated in Figure 3.8, where the first 7 bits are reserved for future usage, while the last bit, the F-flag, defines whether or not the rule is to be used for forwarding (FMR). If the bit is set to 1, the FMR is to be used, if it's set to 0, the rule is treated only as a BMR.
- *ea-len*, 8 bits, defines the length of the Embedded Address (EA) bit length, and allowed values range from 0 to 48.
- *prefix4-len*, 8 bits, defines the prefix length of the rule IPv4 prefix specified in the ipv4-prefix field. Allowed values range from 0 to 32.

- *ipv4-prefix*, 32 bits, defines the IPv4 prefix.
- *prefix6-len*, 8 bits, defines the prefix length of the rule IPv6 prefix specified in the ipv6-prefix field. Allowed values range from 0 to 128.
- *ipv6-prefix*, variable bits, defines the IPv6 domain prefix..
- *S46_RULE-options*, variable bits, that may contain zero or more options that specifies additional parameters. Currently the only option is the S46 Port Parameters Option (OPTION_S46_PORTPARAMS) defined in a later section.

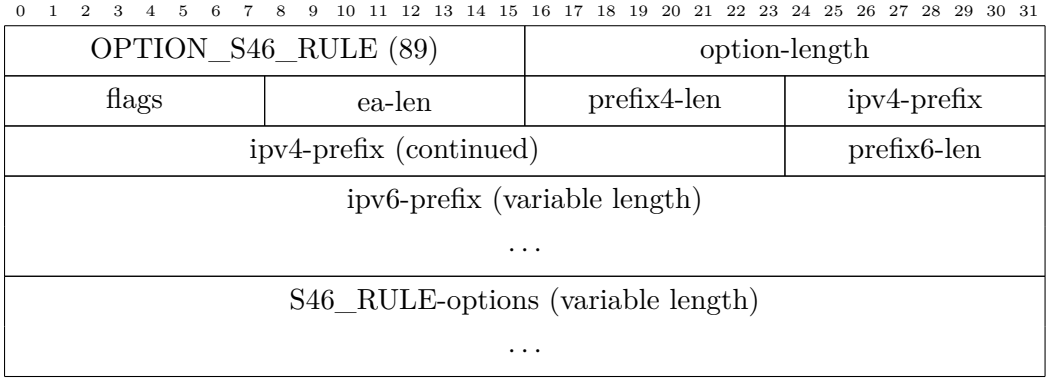

Figure 3.7: DHCPv6 S46 Rule Option - DHCPv6 option 89 is used to convey BMRs and FMRs to the MAP node.

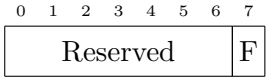

Figure 3.8: DHCPv6 S46 Rule Flags field - The S46 Rule Flags field, which currently defines whether or not the rule is to be used as a FMR or just a BMR.

3.1.6.2 S46 BR Option

The S46 BR option is used to define the IPv6 address of the BR in a MAP-E domain, where BR redundancy can be achieved by using an anycast IPv6 address. The format for this option is illustrated in Figure 3.9, where the fields are defined as follows;

- *option-code*, 16 bits, defines the DHCPv6 option code 90, OPTION_S46_BR.
- *option-length*, 16 bits, defines the length of the entire option expressed in octets, excluding the option-code and option-length fields. Since the br-ipv6-address field has a fixed length of 16 octets (128 bits), the value of this field is always 16.

- *br-ipv6-address*, 128 bits, defines the IPv6 address of the BR in a MAP-E domain.

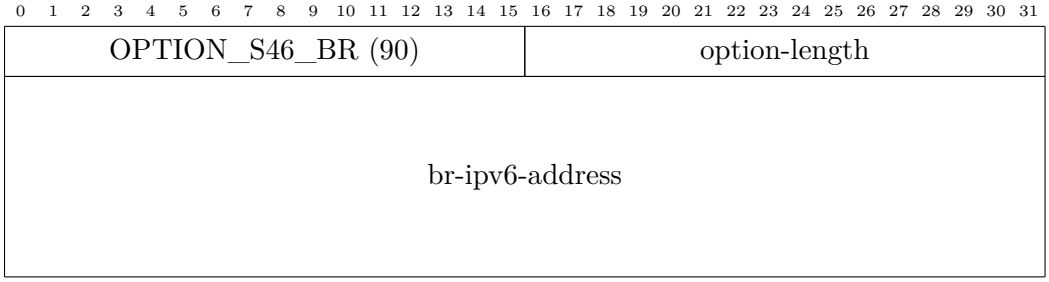

Figure 3.9: DHCPv6 S46 BR Option - DHCPv6 option 90 is used to convey the IPv6 address for the BR in a MAP-E domain.

3.1.6.3 S46 DMR Option

The S46 DMR option is used to define the values of the DMR used in MAP-T domains. The format is defined in Figure 3.10, where the fields are defined as follows;

- *option-code*, 16 bits, defines the DHCPv6 option code 91, OPTION_S46_DMR.
- *option-length*, 16 bits, defines the length of the entire option expressed in octets, excluding the option-code and option-length fields. The value is 1 + the length of the dmr-ipv6-prefix (in octets).
- *dmr-prefix6-len*, 8 bits, defines the bitmask length of the IPV6 prefix specified in the dmr-ipv6-prefix field. Allowed values range from 0 to 128.
- *dmr-ipv6-prefix*, variable bits, defines the IPv6 prefix for the BR in the MAP-T domain.

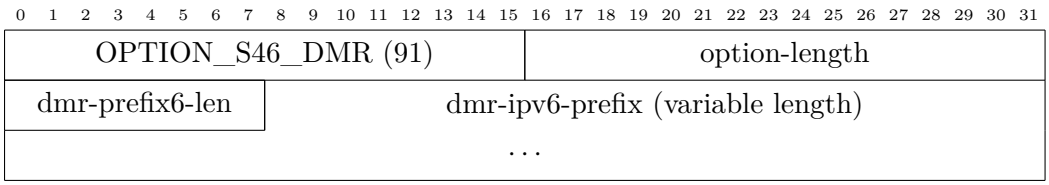

Figure 3.10: DHCPv6 S46 DMR Option - DHCPv6 option 91 is used to convey the IPv6 prefix for the BR in a MAP-T domain.

3.1.6.4 S46 Port Parameters Option

The S46 Port Parameters Option is used to define the optional port parameters used by the mapping algorithm of MAP when IPv4 address sharing takes place, and uses

some of the same parameters previously discussed and shown in Figure 3.6. The option is contained within an OPTION_S46_RULE option, and the format of the option is defined in Figure 3.11, and the fields are defined as follows;

- *option-code*, 16 bits, defines the DHCPv6 option code 93, OPTION_S46_PORTPARAMS.
- *option-length*, 16 bits, defines the length of the entire option expressed in octets, excluding the option-code and option-length fields. Since all the fields are fixed, the value of this field is always 4.
- *offset*, 8 bits, defines the PSID offset. Allowed values range from 0 to 15. The default value for this field in a MAP-E/MAP-T context is 6.
- *PSID-len*, 8 bits, defines the number of significant bits in the PSID field (defined as k in the port mapping algorithm section). Allowed values range from 0 to 15, and if set to 0, the PSID field is ignored.
- *PSID*, 16 bits, defines the PSID to be used in the MAP domain. The first k bits contains the PSID value, while the remaining $16 - k$ bits are zero-padded on the right.

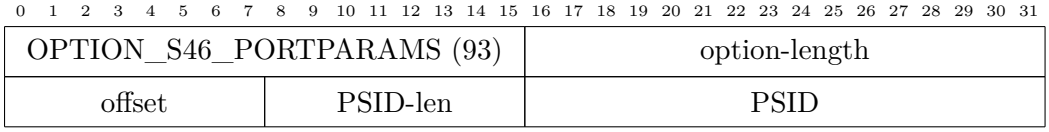

Figure 3.11: DHCPv6 S46 Port Parameters Option - DHCPv6 option 93 is used to convey the PSID of a MAP domain.

3.1.6.5 S46 MAP-E Container Option

The S46 MAP-E Container option specifies an DHCPv6 option that allows sending all of the relevant options and rules for a MAP-E domain as one single DHCPv6 option (rather than several). This is useful, especially when having multiple MAP domains. The current supported encapsulated options are OPTION_S46_RULE and OPTION_S46_BR. There must be at least one of each encapsulated within the option. The format is illustrated in Figure 3.12, and the fields are defined as follows;

- *option-code*, 16 bits, defines the DHCPv6 option code 94, OPTION_S46_CONT_MAPE.

- *option-length*, 16 bits, defines the length of the encapsulated options expressed in octets.
- *encapsulated-options*, variable bits, defines the encapsulated options.

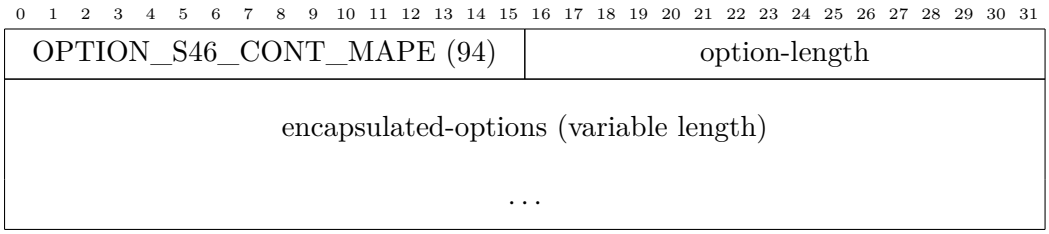

Figure 3.12: DHCPv6 S46 MAP-E Container Option - DHCPv6 option 94 is used to convey multiple S46 options within one option for a MAP-E domain.

3.1.6.6 S46 MAP-T Container Option

The S46 MAP-T Container Option specifies an DHCPv6 option that allows sending all of the relevant options and rules for a MAP-T domain as one single DHCPv6 option (rather than several). This is useful, especially when having multiple MAP domains. The current supported encapsulated options are OPTION_S46_RULE and OPTION_S46_DMR. There must be at least one of the first, and exactly *one* one of the latter. The format is illustrated in Figure 3.13, and the fields are defined as follows;

- *option-code*, 16 bits, defines the DHCPv6 option code 95, OPTION_S46_CONT_MAPT.
- *option-length*, 16 bits, defines the length of the encapsulated options expressed in octets.
- *encapsulated-options*, variable bits, defines the encapsulated options.

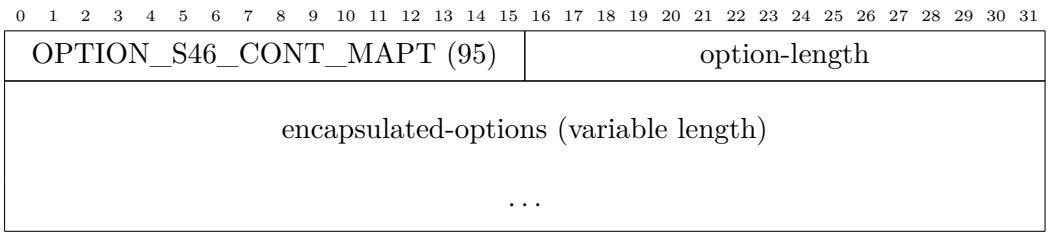

Figure 3.13: DHCPv6 S46 MAP-T Container Option - DHCPv6 option 95 is used to convey multiple S46 options within one option for a MAP-T domain.

3.2 Network

To test our chosen IPv6 transition mechanism, we need a network. At minimum, this needs to consist of one CPE, one distribution switch, and one Border Relay router. We also need Internet access in the form of both native IPv4 and IPv6. Acquiring the Internet access is beyond the scope of this thesis.

3.2.1 Distribution switch

To connect each CPE to the core network, we need one or more distribution switches. The requirements for what features these switches need will vary between network operators depending on how they build their networks.

To follow the principles of IPv6 regarding route summarizations, we find it preferable that having the L3 interfaces as close to the CPEs as possible. As a minimum for our MAP tests, it should support IPv6 unicast routing, DHCPv6 relaying and DHCPv6-PD. It should also have full-duplex gigabit interfaces to facilitate measurements above 100Mbps.

We chose to use a Cisco ME3400 layer 3 gigabit switch (ME-3400EG-12CS-M) as it was easily available for us to test with. It has a total of 16 full-duplex gigabit ports, where 12 of them are combo RJ-45 / SFP ports, while 4 of them are SFP-only. It is pictured in Figure 3.14.

Figure 3.14: Cisco ME-3400EG-12CS - The distribution switch used in our tests.

3.2.2 BR router

We need a BR that supports MAP-T. There seems to be few vendors that has good MAP support in general, but even fewer with MAP-T support. We know the OS made

by IPInfusion supposedly have support (63)(64), but we found no routers running this OS that were suitable as a BR. Looking at open-source alternatives, we find OpenWrt, but it only has the CE part implemented. The CERNET MAP implementation is a three year old proof of concept based on the Softwire-drafts (65), rather than the standard released in 2015. We looked at tiny-map-e, but it only supports MAP-E (66). We also looked into commercial software, specifically Cisco's IOS-XRv, with support for both MAP-E and MAP-T. However, the free version of the software is rate-limited to 2Mbps (67), and is therefore unsuitable for our tests.

Eventually we concluded on using Cisco's CSR1000v (68), which is a x86 based virtual router that can be run in KVM, which were suitable for our setup. It's not free, but there are 60 days' evaluation licenses that are rate-limited to 2.5Gbps, which is more than enough for our use case.

3.2.2.1 Vyatta

We also considered Vyatta, specifically the kernel patches written by Masakazu Asama (69). We found the solution to be more of a proof of concept, rather than something you'd put into production. It consists of several kernel-patches to a Vyatta release more than two years old. In addition, Vyatta is not an open-source product anymore (it was acquired by Brocade in 2012), and therefore future updates and kernel patches are not possible. We wanted to do our tests with solutions that is possible to deploy in production networks. Looking into VyOS, the fork of Vyatta before it became closed-source, it doesn't look like it has any MAP support, and the kernel patches for Vyatta are not applicable on the VyOS fork.

3.2.2.2 VPP

The open-source Vector Packet Processing (VPP) platform was announced by the Linux Foundation as a joint project, FD.io [Fido], on 11th of February 2016 (70). VPP has been in development since 2002, and was donated to the project by Cisco. It's production ready code running in current shipping products all over the globe, and runs on multiple architectures including x86 and Arm, on both x86 servers and embedded devices. The benefits of VPP is high performance, proven technology, modularity, flexibility, and rich feature set. It has support for both MAP-E and MAP-T (71).

Unfortunately, the project came to our attention too late to be considered in this thesis, but we find the project really interesting, and would be suited as BR on commodity servers (which is usually cheaper than proprietary routers).

3.2.3 CPE

The CPE is what the network operator will place at the premises of the end-users they provide Internet connectivity for. Usually there is a difference between CPEs placed at industrial premises compared to residential premises. In our case we want a CPE for residential use in a Fiber-to-the-Home (FTTH) context where the fiber is installed in the resident, and the CPE is connected via fiber.

There are primarily two methods to utilize the fiber; active (Active Optical Network, AON) or passive (Passive Optical Network, PON). The difference between these, and pros/cons for the two solutions, are outside the scope of this thesis, however, since AON is simpler in the way that it doesn't require special hardware, and the fact that each CPE has a 1:1 connection to the active equipment (i.e. the distribution switch), we're looking for a CPE that can be used in AON environments.

CPEs can be divided into two generic categories; those where the fiber can connect directly into the CPE (no need for a media converter), and those where the fiber has to be terminated in a separate media converter, and then connected to the CPE. This also varies between network operators, and there are several pros/cons to consider. Since there are more CPEs without a direct fiber connection, than there is with, we have chosen to go for CPEs that only have RJ-45 interfaces.

Since most CPEs don't support MAP natively (neither MAP-E or MAP-T), we have to find a CPE that we can load alternative software on (or one that already runs some kind of *nix[1] that we can compile on or use custom software).

We'd also like to have support for the latest 802.11ac wireless, with at least Wave 1, but Wave 2 preferable. The CPE should support IPv6-only management, VLAN (so that we can separate the management and client networks, and also provide L2 bridge for IPTV if preferable). With this in mind, we can compile a list of requirements for our CPE;

[1] A Unix-like (sometimes referred to as UN*X or *nix) operating system is one that behaves in a manner similar to a Unix system.

- At least 1x Gigabit Ethernet WAN port
- At least 4x Gigabit Ethernet LAN port
- 802.11AC wireless
- IPv6-only management
- VLAN-support
- MAP-E and MAP-T support

Turns out that this was no feasible task, and at first, it seemed as if we were in a "pick two out of three" scenario (or rather, "pick one out of three"). Support for MAP somewhat implies IPv6-only management, but as we stated before, there are not many CPEs out there with MAP support out-of-the-box. The 802.11ac requirement really narrowed it down (even without looking at MAP support).

By looking at just the MAP support (without looking at other criteria), we came up with the following alternatives (in random order);

1. SEIL, a NetBSD-based x86 series of CPEs developed by Internet Initiative Japan [IIJ] (72)(73).
2. OpenWrt supported router (74).
3. *nix based router that could run CERNET MAP, an open source CPE implementation of MAP-E/MAP-T (65).
4. Vyatta supported router (69).

We found references to vendors with MAP support (63)(64), like Linksys[1], FNSC and Yamaha, but we couldn't find any products with official support for it.

Looking at the alternatives above, we don't really find alternative 3 and 4 suitable. They both rely on an x86 enabled CPE, which there aren't many of with wireless support, and especially 802.11ac. We could leave out the 802.11ac requirement (or wireless in general), and go for a router with only wired connections. But since we want a CPE to be placed in residential homes, we find that good wireless support should be part of it. There's also the case of making sure the wireless network gets working IPv6 – we know there are some network operators that only provide cabled Internet

[1]We know there are Linksys routers that supports OpenWrt, but we found no CPEs with native MAP support.

access, and leaves the decision of wireless networks to the end-user (i.e. they have to supply their own wireless network). Usually that results in the end-user purchasing a wireless *router*, hooks it up, and ends up with double NAPT and no IPv6 on the wireless network.

Alternative 1 looks promising, but it doesn't look like they have a CPE with 802.11ac support. It also seems that they are very Japan-specific, e.g. since they don't seem to have any English documentation (75). This leaves us with alternative 2, an OpenWrt supported router.

Looking at the OpenWrt websites (76), we find that there are numerous routers that supports OpenWrt. By narrowing it down to 802.11ac enabled with 4 or more gigabit ports, we are left with about 30 routers. Among these, there are different levels of support, and especially regarding the wireless network performance. After a lot of research, reading forums up and down, we concluded that the TP-Link Archer C7 AC1750 would suit us the best (77). It has 1x GE WAN, 4x GE LAN, 802.11ac Wave 1, and very good support for OpenWrt (78). It's pictured in Figure 3.15.

Figure 3.15: TP-Link Archer C7 AC1750 - The CPE used in our tests.

After acquiring the Archer C7, it came to our attention that OpenWrt has made an effort in making a list of recommended hardware (79), so that one doesn't have to read forums up and down to figure out what routers have good support.

3.2.3.1 WiTi

There is a CPE that came to our attention at a later date after acquiring the Archer C7, namely the WiTi board from mqmaker (80). It's an open and powerful router platform built to support OpenWrt. It was funded through Indiegogo, and they've released the source codes, schematics and documentation, making it a completely open platform.

It meets all of our requirements by having 2x GE WAN, 4x GE LAN, 802.11ac, and support for OpenWrt. It also has USB3.0 and SATA-ports, the latter making it possible to attach hard drives to it, and as such, it could operate as a NAS, while at the same time be a really good CPE.

Unfortunately, it came to our attention too late to be reviewed in this thesis.

3.2.3.2 Media converter

Since we chose a CPE without fiber connectivity, we also need a media converter. We want one that is 1Gbps capable, bidi-support (that is, RX/TX is sent using separate wavelengths), and most importantly we want it to have a fiber tray. The latter is important, since the fiber entering the house needs to be spliced somewhere. You could do this in a separate unit, but since this is the mounted in residential homes, we want as few boxes as possible. The requirement for a fiber tray built-in really narrowed it down, and ultimately we found the Raycore RC-OE2ATR (81), an unmanaged 100/1000Mbps media converter with built-in fiber tray. It's pictured in Figure 3.16.

Figure 3.16: Raycore RC-OE2ATR - The media converter used.

3.3 Software

Having chosen what transition mechanism to use, and what network equipment to utilize, we're going to look at different types of software to be used in the following sections.

3.3.1 OpenWrt

The OpenWrt project started after Linksys was forced to release the source code of the firmware for their WRT54G series routers. The firmware was modified versions of publicly available code licensed under the GPL (82)(83). This made it possible for developers to add features not previously found on commodity routers, and support for other chipsets, manufacturers and devices types were added.

OpenWrt is an operating system based on the Linux kernel primarily made for embedded systems. All the applications are optimized for size so that the entire OS is made small enough to be put on embedded devices with limited storage. Configuration of OpenWrt is made via CLI, or the custom web interface codenamed LuCI.

Support for MAP was first introduced in r40822 and r40823 (MAP-E), and then finalized in r41003 (MAP-T)[1]. The initial proof of concept code was written by Andrew Yourtchenko (84)(85), which included the CERNET MAP implementation, and his own implementation of MAP-options sent via DHCPv6 that was based on draft-ietf-softwire-map-dhcp-03 (86), which later became RFC7598 (62). We'll be using the latest stable release, Chaos Calmer 15.05.1, released March 2016.

3.3.2 Measurement

Since we want to measure the performance of MAP-T using the chosen CPE and network equipment, we need utilize some tools that can measure at least throughput and latency, but preferably also jitter and other parameters. We review how these tools are used in the next chapter.

3.3.2.1 iperf

iperf was originally a reimplementation of ttcp (Test TCP), and was developed by the Distributed Applications Support Team (DAST) of the National Laboratory for Applied

[1] OpenWrt changesets: r40822, r40823 and r41003.

Network Research (NLANR). iperf can test the network performance in different ways, setting a variety of parameters. It allows testing of TCP, as well as UDP. The latest stable version of iperf is 1.7.0, released in March 2003. iperf later became available as iperf2, and the latest stable release is 2.0.8, released April 2015. iperf2 was rewritten from scratch into iperf3 by ESnet in 2009, and was completed in 2014 (87). The latest stable version is iperf-3.1.2, released February 2016. We'll be using the unstable / source version (88).

3.3.2.2 Netperf

Netperf was originally written in the 90s, and does measurements of both TCP and UDP, amongst other things. The latest stable release is 2.7.0, released in July 2015, and is the version we will use (89).

3.3.2.3 Flent

The FLExible Network Tester (Flent) is a network testing tool written in Python (90). It utilizes several benchmarking tools such as iperf, Netperf and ping. Flent has a modular support for tests, which are defined in configuration files. This allows for identical tests to be ran in different environments, and also supports plotting of graphs using matplotlib. The data it gathers is saved as gzipped JSON file which can be used later on to extract numbers from the tests, making it easy to gather first, analyze later. The latest stable release is 0.14.0, released February 2016. We'll be using the source version (91).

3.3.3 Virtualization

We need something that can virtualize the BR software chosen (Cisco CSR1000v), and preferably also the clients and servers needed for our setup. There are many solutions that could be used, like VMware Player, VMware ESXi, Hyper-V, KVM, Parallels, QEMU, VirtualBox, Xen, and many others (92). We already had Proxmox at hand, and we found it convenient to use in our setup.

Proxmox is an open-source server virtualization environment based on Debian, and supports both containers, using LXC, and full virtualization using KVM (93). The BR software supports using KVM, and for the server and client, a lightweight container is preferable.

3.3.4 Client + server

We also need one server (hostname **map-server**) and one client (hostname **map-client**) that can be used for performance testing, and also to host the DHCPv6 server. Both hosts will be made using LXC containers using Debian Jessie (94)(95). For the DHCPv6 server, we're going to use *isc-dhcp-server* version 4.3.1-6+deb8u2, developed by the Internet Systems Consortium (ISC).

4

Implementation

We've looked at the different transition mechanisms available, and concluded that MAP-T is preferable. We also looked at how MAP-T works, and chosen the different network- and software parts we're going to use. This chapter looks at the setup and configuration, before we look at the measurements in the next chapter.

Since we want to measure the MAP-T performance of our chosen CPE, we also want to have measurements that we can compare it with. We'll do initial measurements with native dual-stack, where the CPE, distribution switch, and the BR gets public IPv4 and IPv6 addresses. This implies normal stateful NAT44 on the CPE for IPv4, and normal IPv6 unicast routing for IPv6. After doing measurements on this setup, we'll convert it to an IPv6-only network between the CPE, distribution switch, and BR. We'll then deploy MAP-T and perform new measurements.

4.1 Network

The different devices will be connected as illustrated in Figure 4.1. This is the physical/logical setup, showing all the devices, how they are connected to each other, and also how they logically connect to the Proxmox virtualization environment via separate VLANs. Proxmox is where the client, server and BR will be hosted, and the physical server hosting Proxmox is connected with four individual gigabit connections to a gigabit core switch. This is done to ensure that there are no bottlenecks, and that all devices are guaranteed 1 Gbps throughput in all directions.

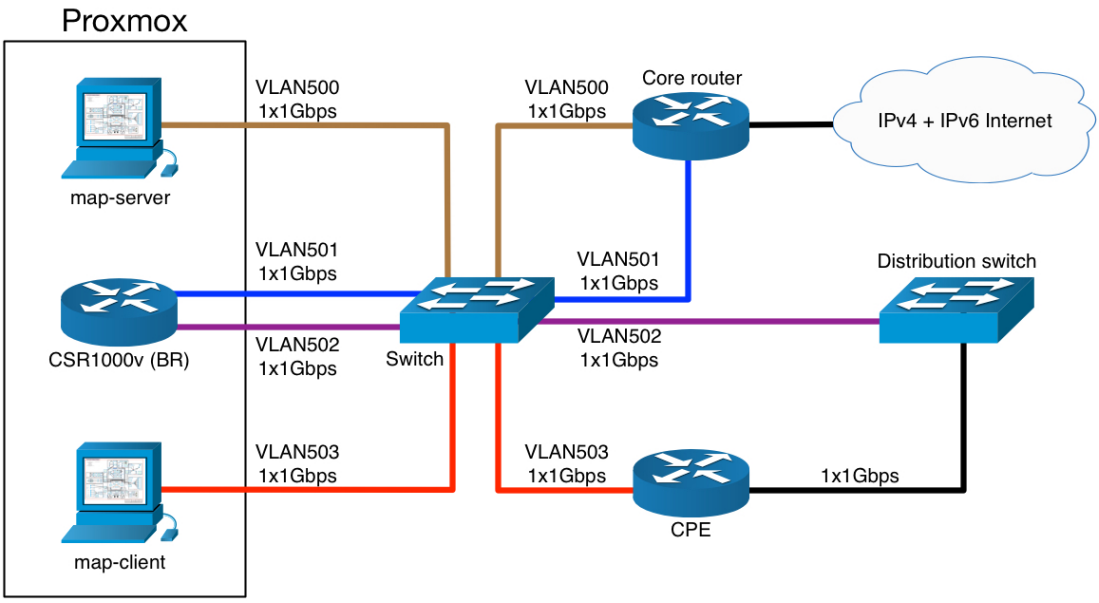

Figure 4.1: Network topology - The network topology showing how everything is connected.

Even if Proxmox complicates the network setup to some degree, the connections between the **map-client**, CPE, distribution switch, BR and **map-server** are working as illustrated in Figure 4.2, which shows the logical setup of our environment. Connecting the **map-server** on the outside of our MAP domain ensures that the measurements we're doing takes both the CPE and BR into account.

The configuration of the core switch, core router and the Internet access is outside the scope of the thesis, but the configuration for the core router is found in Listing A.5.

4.1.1 Distribution switch

The distribution switch is configured for dual-stack, as seen in Listing A.6. Measurements are done, and then the distribution switch is configured for IPv6-only to facilitate the MAP-T measurements. There are no requirements that the network has to be IPv6-only, and as such, the dual-stack setup could've been kept as-is on the distribution switch and BR, as long as the CPE only receives an IPv6 address (which can be done by disabling IPv4 on the WAN interface on the CPE). The IPv6-only configuration, including the DHCPv6 configuration, can be seen in Listing A.9. The distribution switch runs IOS version 12.2(60)EZ8, released in June 2015.

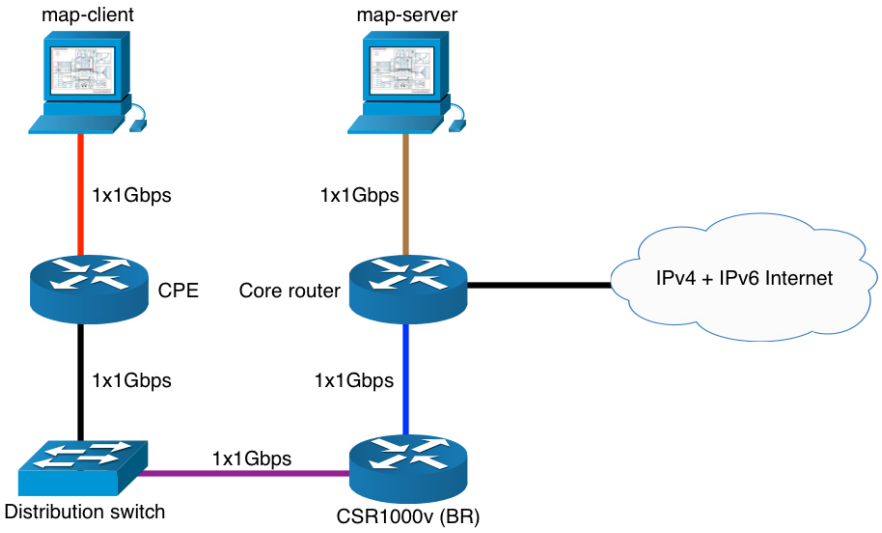

Figure 4.2: Logical network topology - The logical network topology.

4.1.2 BR

The BR is also configured for dual-stack first, and is shown in Listing A.7. After the dual-stack measurements were done, we reconfigured it to IPv6-only within the MAP-T domain (i.e. only the inside interface). The outside interface is still dual-stacked since it's providing the IPv4 public connectivity for the MAP-T domain. The IPv6-only configuration, including the MAP-T parameters discussed later in this chapter, is seen in Listing A.10. The BR runs IOS-XE version Denali 16.2.1, released in March 2016.

MAP-T supports redundant BRs without any form of state synchronization. This is possible due to the stateless NAT64 translation taking place. It's achieved by having two or more BR's using the same DMR IPv6 prefixes, which could be anycasted using a routing protocol. This is outside the scope of the thesis.

4.1.3 CPE + OpenWrt

The stock firmware on our selected CPE does not support MAP, and as such, we'll replace it with OpenWrt. The installation is straight forward, and includes downloading a new firmware made specifically for the Archer C7[1]. The new firmware is then uploaded to the router using the stock web interface. Take note that the original filename of the OpenWrt firmware have to be shorted, and not use special characters, as the stock

[1]OpenWrt CC15.05.1: openwrt-15.05.1-ar71xx-generic-archer-c7-v2-squashfs-factory.bin

firmware is a bit picky about it (i.e. it refuses to upload the new firmware if the filename is long or use special characters). After uploading the new firmware, the router rebooted, but never came online again. Inspecting the device page closely (78), we see that there is a note mentioning that hardware version 2 devices starting late 2015 has a new flash chip, gd25q128, that is not supported in Chaos Calmer 15.05 or 15.05.1.

We therefore had to use the trunk version, Designated Driver, which is the developer version. There are nightly binary builds[1] of trunk which are suitable. Luckily for us, the Archer C7 has a TFTP recovery mode, where the router looks for a file named "ArcherC7v2_tp_recovery.bin" on a TFTP server listening on the IP address 192.168.1.66. The recovery mode is activated by holding the WPS/reset button during power up until the WPS LED turns on. After a reboot, the router booted fine into OpenWrt, as seen on Listing 4.1.

```
BusyBox v1.24.1 () built-in shell (ash)


  -------                       --------         --
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 DESIGNATED DRIVER (Bleeding Edge, r49161)
 -----------------------------------------------------
  * 2 oz. Orange Juice         Combine all juices in a
  * 2 oz. Pineapple Juice      tall glass filled with
  * 2 oz. Grapefruit Juice     ice, stir well.
  * 2 oz. Cranberry Juice
 -----------------------------------------------------
root@OpenWrt:~#
```

Listing 4.1: OpenWrt Designated Driver running on Archer C7.

Next, we configure the CPE with dual-stack having a public IPv4 address on the wan interface, and an IPv6 address on the wan6 interface. We then statically route the 2a02:4260:f000:3::/64 prefix on the distribution switch, pointing to the IPv6 address of

[1] OpenWrt trunk build: openwrt-ar71xx-generic-archer-c7-v2-squashfs-factory.bin

the CPE. The /64 prefix is then assigned on the inside LAN interface to be used by clients. The complete CPE configuration can be seen in Listing A.8.

After measurements of the dual-stack setup were made, we move on to configure the CPE with MAP support. We need to install a few extra packages on the CPE to enable MAP, namely *map* and *map-t*. This is done via the OpenWrt package manager *opkg*. Trying to install these, however, yielded a dependency error claiming that the kernel module *kmod-ipv6* had to be installed, but no such package was found by the package manager, as shown in Listing 4.2.

```
root@OpenWrt:~# opkg install map-t
Installing map-t (6) to root...
Downloading http://downloads.openwrt.org/snapshots/trunk/ar71xx/generic/
    packages/routing/map-t_6_ar71xx.ipk.
Collected errors:
 * satisfy_dependencies_for: Cannot satisfy the following dependencies
     for map-t:
 *  kmod-ipv6 *
 * opkg_install_cmd: Cannot install package map-t.
```

Listing 4.2: OpenWrt Designated Driver: kmod-ipv6 issue.

It was unclear at first what caused this issue, but digging deeper, we find that *map-t* is just a meta package of the *nat46* package, which is a part of the OpenWrt Routing package feed. Looking at the Makefile[1], we see that *kmod-ipv6* is a dependency for the *nat46* package, and thus, also the *map-t* package. In the process, we stumbled upon a ticket on the OpenWrt bug tracker (96), where a circular dependency issue is caused by a commit changing the *+kmod-ipv6* dependency to *+@IPV6* (97). Eventually we found that *kmod-ipv6* kernel module is included by-default in trunk (i.e. it's obligatory, and cannot be removed), hence the dependency scheme for IPv6 has changed, and all packages that relies on *kmod-ipv6* now needs to reference *@IPV6* (but without the + sign, which was the mistake done in the aforementioned commit). To fix the problem, the Makefile of *nat46* needed to be changed *before* compiling, and it was clear that we had to compile our own version of OpenWrt.

[1]OpenWrt nat46 Makefile: openwrt-routing/packages/master/nat46/Makefile.

After looking at the developer pages for the OpenWrt project (98), we figured out how to build a custom version of OpenWrt based on the source files, which was easier than what we thought at first. After cloning the source files with git, we updated the feeds, and then replaced the invalid dependency. After that, we prepared the needed packages (the complete list was fetched from the already running OpenWrt on the CPE), followed by additional package selections via *make menuconfig*. The latter is also where we chose the platform and device profile, so that the correct images could be built. Finally, it was a matter of compiling it all. The process is summarized in Listing 4.3.

```
1 git clone http://git.openwrt.org/openwrt.git openwrt-trunk
2 cd openwrt-trunk/
3 ./scripts/feeds update -a
4 sed s/+kmod-ipv6/@IPV6/ -i feeds/routing/nat46/Makefile
5 ./scripts/feeds install ath10k-firmware-qca988x base-files busybox
    dnsmasq dropbear firewall fstools hostapd-common ip6tables iptables
    iptables-mod-conntrack-extra iw iwinfo jshn jsonfilter kernel kmod-
    ath kmod-ath10k kmod-ath9k kmod-ath9k-common kmod-cfg80211 kmod-gpio-
    button-hotplug kmod-ip6-tunnel kmod-ip6tables kmod-ipt-conntrack kmod
    -ipt-conntrack-extra kmod-ipt-core kmod-ipt-nat kmod-iptunnel6 kmod-
    ledtrig-usbdev kmod-lib-crc-ccitt kmod-mac80211 kmod-nf-conntrack
    kmod-nf-conntrack6 kmod-nf-ipt kmod-nf-ipt6 kmod-nf-nat kmod-nls-base
     kmod-ppp kmod-pppoe kmod-pppox kmod-slhc kmod-usb-core kmod-usb-ohci
     kmod-usb2 libblobmsg-json libc libexpat libgcc libip4tc libip6tc
    libiwinfo libiwinfo-lua libjson-c libjson-script liblua libnl-tiny
    libpolarssl libubox libubus libubus-lua libuci libuci-lua libuclient
    libustream-polarssl libxtables lua luci luci-app-firewall luci-base
    luci-lib-ip luci-lib-jsonc luci-lib-nixio luci-mod-admin-full luci-
    proto-ipv6 luci-proto-ppp luci-ssl luci-theme-bootstrap map mtd
    netifd odhcp6c odhcpd opkg ppp ppp-mod-pppoe procd px5g rpcd swconfig
     uboot-envtools ubox ubus ubusd uci uclient-fetch uhttpd uhttpd-mod-
    ubus usign wpad-mini map-t nat46 map kmod-ath10k
6 make menuconfig
7 make -j20 V=s
```

Listing 4.3: OpenWrt Designated Driver: fixing the kmod-ipv6 issue.

After the build completed, we had binary firmware files ready to be installed on the CPE. We did a reboot, and it came online without issues. Looking at the installed

packages, we confirmed that the necessary packages were installed, as seen in Listing 4.4. We submitted an issue on the issue tracker for the OpenWrt Routing feed (97), so that the dependency issue gets fixed.

```
root@OpenWrt:~# opkg list_installed|egrep "map|nat46"
kmod-nat46 - 4.1.20+6-1
map - 4-5
map-t - 6
```

Listing 4.4: OpenWrt Designated Driver: map packages installed.

Having the needed packages in OpenWrt, we'll proceed to look at the MAP parameters and how to provision them in the next sections.

4.2 MAP parameters

Finding suitable MAP parameters depends on the network and how the network operators builds their networks. We found that deploying IPv4 as-a-service on top of an IPv6-only network would be beneficial – especially for new network operators with scarce IPv4 resources. By using a sharing ratio of 1:1, setting $o+r = 32$, such that each CPE has a full IPv4 address without port-restrictions, they can build a network as if it would be dual-stacked, but in fact is IPv6-only. When the time comes that they run out of IPv4 address space, they can decrease the suffix size (p), and as such, increase the sharing ratio. As a part of testing MAP-T, we tried parameters where $o + r = 32$, causing a sharing ratio of 1:1, as shown in Figure 4.3. Converting that to a BMR, we'd set the rule IPv6 prefix to 2a02:4260:f800:0::/52, rule IPv4 prefix to 185.54.31.128/28, and the EA bits to 4.

Trying to configure this on the BR yielded an error, stating that the "ports-parameter" *had* to be set, as seen in Listing 4.5. We reckon that this has to be a bug, as 1:1 sharing ratio should be configurable.

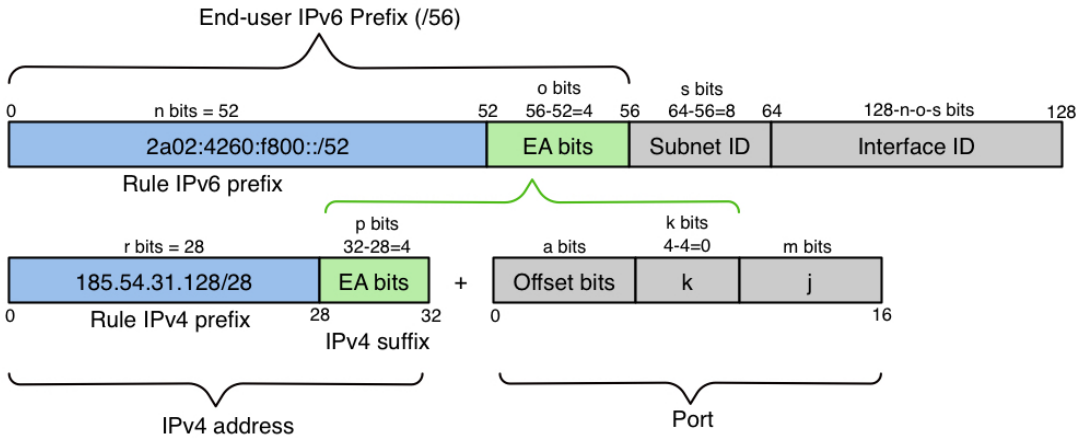

Figure 4.3: MAP-T BMR, sharing ratio 1:1 - Setting the BMR so that the sharing ratio is 1:1.

```
csr1000v(config)#nat64 map-t domain 1
csr1000v(config-nat64-mapt)# default-mapping-rule 2a02:4260:f000:4::/64
csr1000v(config-nat64-mapt)# basic-mapping-rule
csr1000v(config-nat64-mapt-bmr)#  ipv6-prefix 2a02:4260:f800:0::/52
csr1000v(config-nat64-mapt-bmr)#  ipv4-prefix 185.54.31.128/28
csr1000v(config-nat64-mapt-bmr)#exit
%NAT64 MAP-T: Basic-mapping-rule for domain 1 must include ipv6-prefix,
    ipv4-prefix, and ports-parameter before it can be used.
```

Listing 4.5: Cisco CSR1000v refuses to do 1:1 share ratio.

Since we want to measure the usability of the end-user, we find that implementing a sharing ratio higher than 1:1 should give a reduced native IPv4 experience, since a public IPv4 address would be shared between multiple end-users. Setting a reasonable sharing ratio boils down to how few ports each CPE should be assigned. Guo-liang Han et al. in (51) looks at how many ports each subscriber needs during peak time, and found that more than 99.6% of the customers at each time point required fewer than 128 ports. That number is likely to decrease as more services on the Internet becomes IPv6 enabled.

With that in mind, we tried a more liberal sharing ratio of 1:8, where $o + r = 35$, and setting the BMR rule IPv6 prefix to 2a02:4260:f800::/50, rule IPv4 prefix to 185.54.31.128/29, and the EA bits to 6. We also specified a PSID offset of 13 to spread

the ports with only 1 contiguous port per port range. Each user gets 8191 ports, which should be more than sufficient.

As a side note we won't utilize FMRs in our tests, since we only have one CPE. CPE-to-CPE communication is therefore not feasible.

4.2.1 OpenWrt MAP issues

We tried to get the CPE working with the above MAP parameters. We used DHCPv6-PD and the S46 DHCPv6 options, as covered in the next section. The CPE receives the option fine, assigns the IPv6 prefix, parses the MAP options received via DHCPv6, but MAP-T doesn't work. There is no IPv4 connectivity, and no default IPv4 route is inserted. Debugging the setup of the MAP interface reveals the error `wan6_4 (1272): sh: write error: Function not implemented`, as seen in Listing 4.6.

```
1  14:52:08 daemon.notice netifd: Interface 'wan6' is now up
2  14:52:08 daemon.notice netifd: Interface 'wan6_4' is setting up now
3  14:52:08 daemon.notice netifd: wan6 (1087): Command failed: Unknown
       error
4  14:52:08 user.notice firewall: Reloading firewall due to ifup of wan6 (
       eth0)
5  14:52:09 daemon.notice netifd: wan6_4 (1272): sh: write error: Function
       not implemented
6  14:52:09 kern.info kernel: [   26.566669] nat46: adding device (map-
       wan6_4)
7  14:52:09 kern.warn kernel: [   26.571113] [nat46] make_nat46_instance:
       allocated nat46 instance with 1 pairs
8  14:52:09 kern.warn kernel: [   26.579328] nat46: netdevice nat46 'map-
       wan6_4' created successfully.
9  14:52:09 kern.info kernel: [   26.674841] nat46: configure device (map-
       wan6_4) with 'local.style MAP local.v4 185.54.31.128/29 local.v6 2a02
       :4260:f800::/50 local.ea-len 6 local.psid-offset 13 remote.v4
       0.0.0.0/0 remote.v6 2a02:4260:f000:4::/64 remote.style RFC6052 remote
       .ea-len 0 remote.psid-offset 0'
10 14:52:09 daemon.warn odhcpd[902]: A default route is present but there
       is no public prefix on br-lan thus we don't announce a default route!
11 14:52:09 daemon.notice netifd: wan6_4 (1272): sh: write error: Function
       not implemented
12 14:55:24 daemon.notice netifd: wan6 (1087): Command failed: Unknown
       error
```

Listing 4.6: OpenWrt Designated Driver MAP issue.

Digging in the code, it seems the error is happening in `/lib/netifd/proto/map.sh`, and specifically in one of the for-loops, which iterates through all the port sets. We altered the code to aid in debugging, and the complete for-loop is displayed in Listing 4.7.

```
for portset in $(eval "echo \$RULE_${k}_PORTSETS"); do
    for proto in icmp tcp udp; do
        snat_ip=$(eval "echo \$RULE_${k}_IPV4ADDR")
        echo "`date +%H:%M:%S`: map.sh: map_setup: for-debug: start. prot:
            $proto, rule: $snat_ip, portset: $portset" >> /tmp/map-debug.txt
        json_add_object ""
        json_add_string type nat
        json_add_string target SNAT
        json_add_string family inet
        json_add_string proto "$proto"
        json_add_boolean connlimit_ports 1
        json_add_string snat_ip $(eval "echo \$RULE_${k}_IPV4ADDR")
        json_add_string snat_port "$portset"
        json_close_object
        echo "`date`: map.sh: map_setup: for-debug: end." >> /tmp/map-debug.
            txt
    done
done
echo "`date +%H:%M:%S`: map.sh: map_setup: for-debug: end of loop." >> /
    tmp/map-debug.txt
```

Listing 4.7: OpenWrt Designated Driver MAP issue, for-loop

Rebooting the CPE so that it re-provisions the MAP parameters, and checking the logs again, we see from Listing 4.8 that the for-loop is interrupted mid-cycle; the last echo is never output before the error. Our suspicion falls on the number of port ranges. Since we have a PSID offset $a = 13$, and PSID length $k = 6 - (32 - 29) = 3$, we know that the number of contiguous ports per port set is $x = 2^{16-13-3} = 1$. The number of port sets is $y = 2^{13} - 1 = 8191$. The for-loop makes the NAT rules, and has to make the same port specific rule for each of the three protocols TCP, UDP and ICMP. In turn, this means that the for-loop has to iterate through 24.573 permutations. We think that creating the rules takes too long time, and that it's interrupted before it's completed.

```
1 <snip>
2 15:41:53: map.sh: map_setup: for-debug: start. prot: icmp, rule:
    185.54.31.135, portset: 7719-7719
3 15:41:54: map.sh: map_setup: for-debug: end.
4 15:41:54: map.sh: map_setup: for-debug: start. prot: tcp, rule:
    185.54.31.135, portset: 7719-7719
5 15:41:54: map.sh: map_setup: for-debug: end.
6 15:41:54: map.sh: map_setup: for-debug: start. prot: udp, rule:
    185.54.31.135, portset: 7719-7719
7 15:41:55: map.sh: map_setup: for-debug: end.
8 15:41:55: map.sh: map_setup: for-debug: start. prot: icmp, rule:
    185.54.31.135, portset: 7727-7727
9 15:41:55: daemon.notice netifd: wan6_4 (1272): sh: write error: Function
     not implemented
```

Listing 4.8: OpenWrt Designated Driver MAP issue, for-loop output

Based on this, we selected new MAP parameters that would have more contiguous ports, so that the number of permutations in the for-loop is decreased. We also want to reserve the system ports 0-1023, since with the previous MAP parameters, only port 0-7 was reserved. New parameters were chosen giving a sharing ratio of 1:16, and we set the BMR rule IPv6 prefix to 2a02:4260:f800::/48, rule IPv4 prefix to 185.54.31.128/28, and the EA bits to 8. We left the PSID offset to the default value of 6. The CPE is assigned the prefix 2a02:4260:f800:3f00::/56 via DHCPv6-PD, and thus we can conclude the following;

- End-user IPv6 prefix: 2a02:4260:f800:3f00::/56
- BMR: rule IPv6 prefix: 2a02:4260:f800::/48
- BMR: rule IPv4 prefix: 185.54.31.128/28
- BMR: rule EA-bit length: 8
- PSID offset: 6

With that information we can calculate the number of port sets, ports per port set, the shared IPv4 address for that specific CE, and also the MAP IPv6 address for the CE;

- EA bits offset $n = 48$
- IPv4 suffix bits $p = 32 - 28 = 4$
- PSID start $s = n + p = 48 + 4 = 52$
- PSID length $q = o - p = 56 - 48 - 4 = 4$
- PSID value: 0xF
- IPv4 suffix value: 0x3
- CEs shared IPv4 address: 185.54.31.128 + 3 = 185.54.31.131 (hex 0xB9361F83)
- CEs MAP IPv6 address: 2a02:4260:f800:3f00:0000:b936:1f83:000f, or short version 2a02:4260:f800:3f00::b936:1f83:f
- The total number of consumed IPv4 addresses are $2^{(32-28)} = 16$.
- The maximum number of CEs that can use this BMR is $2^8 = 256$.
- The number of CEs that share on a single IPv4 address is $2^4 = 16$.
- The number of ports per range (x) for a given CE is $x = 2^6 = 64$.
- The number of ranges (y) for a given CE is $y = 2^6 - 1 = 63$.
- The total number of ports (z) for a given CE is $z = 64 \times 63 = 4032$.
- The upper port (v) that is reserved is $2^{(16-6)} - 1 = 1023$.

Using the new MAP parameters, the CPE provisioned fine, and we had a working MAP setup over a IPV6-only network. The final MAP configuration for the CPE can be seen in Listing A.11.

As a part of the debugging, we installed Chaos Calmer 15.05.1 on the CPE, using a custom build from the 15.05 source files, that had backported the flash chip support from trunk, making it possible for us to use that version. We did this because we thought the custom changes in the *nat46* package might be related. Chaos Calmer still uses the *kmod-ipv6* kernel module, and hence we could use the native *map*, *map-t* and *nat46* packages.

4.3 Server + client

Both **map-client** and **map-server** is set up in the Proxmox environment as two LXC containers. They get 4x vCPU and 4GB of RAM each, and their disks are SSD. Since the bandwidth requirements for the DHCPv6 server is low, we decided to have the DHCPv6 server on **map-server**. We don't think that it would impose any issues, even if **map-server** will be part of the measurements.

Having installed both containers, we proceeded to install the needed measurement tools. The build process is described in Listing 4.9, and the installed versions are shown in Listing 4.10.

```
mkdir ~/bin
cd ~/bin
# netperf
wget ftp://ftp.netperf.org/netperf/netperf-2.7.0.tar.gz
tar -xvf netperf-2.7.0.tar.gz
cd netperf-2.7.0
./configure --enable-demo
make -j20
make install

# iperf3
cd ~/bin
apt-get install git make build-essential
git clone https://github.com/esnet/iperf.git iperf3
cd iperf3/
make uninstall
make clean
./configure
make -j20
make install
rm /usr/lib/x86_64-linux-gnu/*iperf*
ldconfig

# flent
cd ~/bin
apt-get install python3.4 python3-matplotlib
git clone https://github.com/tohojo/flent.git
cd flent
python3 setup.py build
python3 setup.py install
```

```
31
32 # http-getter
33 cd ~/bin
34 apt-get install curl libcurl3 libcurl4-openssl-dev cmake
35 git clone https://github.com/tohojo/http-getter.git
36 cd http-getter/
37 make clean
38 make
39 make install
```

Listing 4.9: Measurement tools build process.

```
1 root@map-client:~# flent -V
2 Flent v0.14.0-git-5ebb2ae.
3 Running on Python 3.4.2 (default, Oct  8 2014, 10:45:20)  [GCC 4.9.1].
4 Using matplotlib version 1.4.2 on numpy 1.8.2.
5 No PyQt4. GUI won't work.
6
7 root@map-client:~# iperf3 --version
8 iperf 3.1
9 Linux map-client 4.2.8-1-pve #1 SMP Fri Feb 26 16:37:36 CET 2016 x86_64
10 Optional features available: CPU affinity setting, IPv6 flow label, TCP
       congestion algorithm setting, sendfile / zerocopy
11
12 root@map-client:~# netserver -V
13 Netperf version 2.7.0
```

Listing 4.10: Versions of the different measurement tools.

4.3.1 DHCPv6

Having a set of MAP parameters to use, we need to configure these on our DHCPv6 server. We need to define the IPv6 subnet that the server listens on (or else the DHCPv6 server refuses to start). Then we need to define the prefix delegation parameters, and also define the S46 MAP-T Container Option (option 95), so that we can assign the option within the subnet6 definition. The configuration thus far can be seen in Listing 4.11, and should be stored as `/etc/dhcp/dhcpd6.conf`. Then we can set the *isc-dhcp-server* to work in IPv6 mode, as shown in Listing 4.12.

```
subnet6 2a02:4260:f000:1::/64 {
}

option dhcp6.mapt-option code 95 = string;

subnet6 2a02:4260:f000:2::/64 {
  range6 2a02:4260:f000:2::100 2a02:4260:f000:2::1000;
  prefix6 2a02:4260:f800:: 2a02:4260:f800:ff00:: /56;
}
```

Listing 4.11: ISC DHCPv6 Prefix Delegation options, /etc/dhcp/dhcpd6.conf

```
sed s/OPTIONS=\"\"/OPTIONS=\"-6\"/ -i /etc/default/isc-dhcp-server
sed s/DHCPD_CONF=\/etc\/dhcp\/dhcpd.conf/DHCPD_CONF=\/etc\/dhcp\/dhcpd6.
    conf/ -i /etc/default/isc-dhcp-server
touch /var/lib/dhcp/dhcpd6.leases
/etc/init.d/isc-dhcp-server restart
```

Listing 4.12: Activating *isc-dhcp-server* IPv6 mode.

Then we need to define the content of option 95 as per the MAP parameters defined. We made a Perl script based on the S46 Option details in the previous chapter. We feed it with the MAP parameters, and out comes the proper DHCPv6 option. The script can be seen in Listing A.3. It doesn't support MAP-E, and doesn't do a lot of error checks, but it works for our purpose. It could easily be extended to support other parameters, and also do error checks. The complete DHCPv6 configuration file, together with the content of option 95, is shown in Listing A.12.

For a simpler rule generation, one can also use the MAP Simulation Tool made by Cisco (99). It allows for creation of multiple rules, both for MAP-E and MAP-T.

5

Evaluation

We've looked at different IPv6 transition mechanisms, and identified some of the challenges. We want to look at throughput and end-user usability using commodity hardware. During the last chapter we looked at the configuration and setup of our test environment, and also addressed different challenges that emerged during the configuration. We also set the MAP parameters that were to be used during our tests. In this chapter we'll look at how the measurements were done, and what they tell us.

5.1 Measuring

Our tests are split into three categories;

1. Pure throughput tests using only iperf3.
2. Combined throughput and latency tests using Flent.
3. Connectivity tests done on a client with Windows 7.

Marius Georgescu et al. in (100) did throughput- and latency measurements with both MAP-E and MAP-T (using ASAMAP, the Vyatta kernel patches). They measured with 1 and 10 nodes, and used UDP streams in 100Mbps-only environments. We find their results interesting, but since we're testing with 1Gbps, we can't really compare the results. Also, they did not utilize commodity hardware.

The OpenWrt community has a note on the Archer C7 page regarding throughput(78). The OpenWrt firmware doesn't support the hardware NAT capability of the router, and the expected throughput is "between 200- and 300Mbps".

In all of our tests we took note of the *sirq* levels on the CPE to verify if the bottleneck was the CPE or other parts of the network. In all cases the value of *sirq* were measured to 98-99%, meaning that OS is *very* busy servicing Ethernet interfaces interrupts. In other words; the router operates at maximum capacity, and thus, is the actual bottleneck.

Using the terms *download* and *upload*, we look at this from the end-user's perspective, which means that *download* means downstream from the Internet, and *upload* means upstream towards Internet. Also, we refer to *dual-stack* as *DS* for simplicity.

5.2 Throughput

Our throughput tests will use iperf3 using multiple TCP streams in different upload and download combinations to simulate different usage scenarios. Each graph consists of 100 tests done over a period of 60 seconds (i.e. each second has 100 samples). Average is the average of all samples with the same timestamp. The maximum and minimum values are the highest and lowest measurement of the 100 samples for each timestamp. We'll vary on the number of TCP streams, and also the direction of them. In addition, we'll use both IPv4 and IPv6 TCP streams in parallel.

5.2.1 MAP measurements with Chaos Calmer 15.05.1

Our dual-stack tests were done while we utilized the Designated Driver (trunk) version of OpenWrt. Due to the MAP issues we experienced, as explained in the previous chapter, we did our MAP-T tests on Chaos Calmer 15.05.1.

5.2.1.1 Download: DS vs. MAP

Our first test consists of pure download throughput using 8 parallel TCP streams. We tested IPv4 and IPv6 separately, and measured both dual-stack (using trunk) and MAP-T (using CC15.05.1). The results are shown in Figure 5.1.

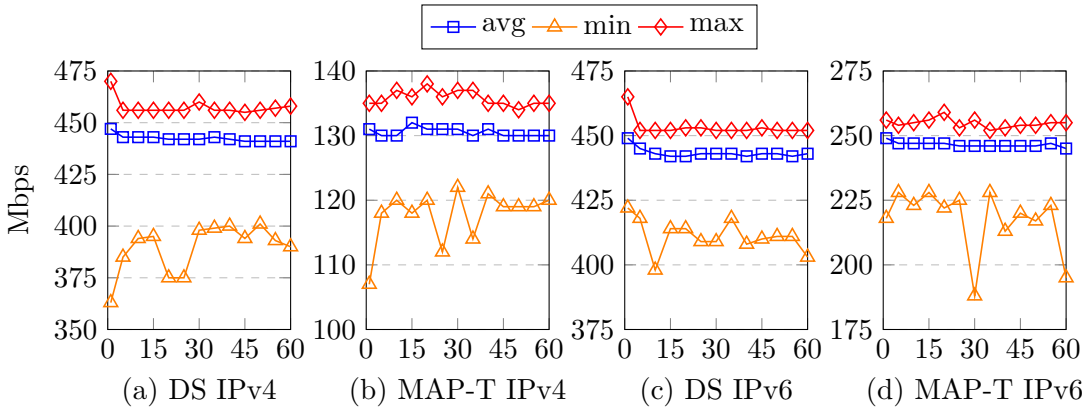

Figure 5.1: Download throughput, DS trunk vs. MAP CC - Average, minimum and maximum download speeds in Mbps with 8 parallel TCP streams. 100 individual measurements of 60 seconds. DS on trunk, MAP-T on CC15.05.1.

We notice that the throughput is drastically reduced in our MAP-T environment. On IPv4, the average download is 443Mbps on dual-stack and 131Mbps using MAP-T, which is a 70% reduction in speed. More interesting is the fact that IPv6 download throughput also is drastically reduced. The only IPv6 difference in terms of configuration, is statically configured IPv6 prefix vs. DHCPv6-PD, which, in theory, should not cause a 44% reduction (444Mbps vs. 247Mbps). In both scenarios IPv6 is only pure unicast routing.

5.2.1.2 Download, IPv4+IPv6: DS vs. MAP

Our next test is also download throughput, but this time we use 4 parallel TCP streams, and test IPv4 and IPv6 at the same time. We measured dual-stack (using trunk) and MAP-T (using CC15.05.1). The results are shown in Figure 5.2.

Also here we notice a drastically reduced throughput in our MAP-T environment. MAP-T IPv4 has a 61% speed reduction (78Mbps vs. 204Mbps) compared to dual-stack, which is actually better (measured in percent) than in our previous test. The reduction is almost the same for IPv6 (79Mbps vs. 200Mbps). Summarized, the reduction is also 61% (157Mbps vs. 404Mbps).

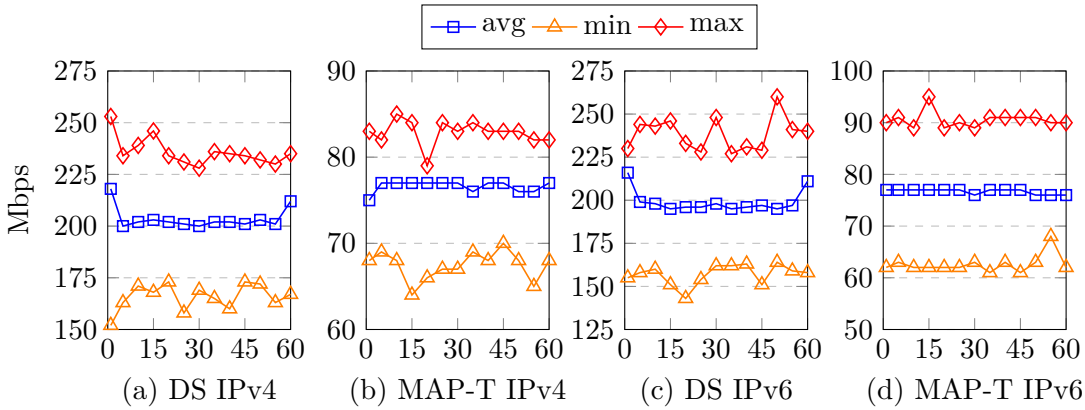

Figure 5.2: Download throughput, DS trunk vs. MAP CC, parallel IPv4+IPv6 - Average, minimum and maximum download speeds in Mbps with 4 parallel TCP streams on both IPv4 and IPv6 at the same time. 100 individual measurements of 60 seconds. DS on trunk, MAP-T on CC15.05.1.

5.2.1.3 Upload: DS vs. MAP

Next test is upload. As with the download test, we utilize 8 parallel TCP streams, and test IPv4 and IPv6 individually. We measured dual-stack (using trunk) and MAP-T (using CC15.05.1). The results are shown in Figure 5.3.

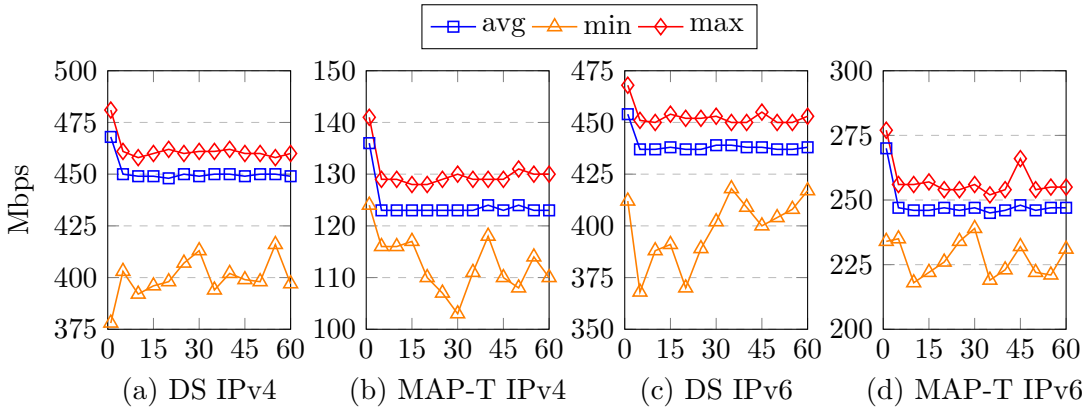

Figure 5.3: Upload throughput, DS trunk vs. MAP CC - Average, minimum and maximum upload speeds in Mbps with 8 parallel TCP streams. 100 individual measurements of 60 seconds. DS on trunk, MAP-T on CC15.05.1.

Also in upload the throughput is drastically reduced. On IPv4, the average upload is 451Mbps on dual-stack and 125Mbps using MAP-T, which is a 72% reduction in

speed, even worse than the download throughput. As with download, the IPv6 upload is also drastically reduced. We start to see a pattern regarding IPv6, and must figure out what's causing this. More on that in a later section.

5.2.1.4 Upload, IPv4+IPv6: DS vs. MAP

As with the dual download test, we do the same with upload. We use 4 parallel TCP streams, and test IPv4 and IPv6 at the same time. We measured dual-stack (using trunk) and MAP-T (using CC15.05.1). The results are shown in Figure 5.4.

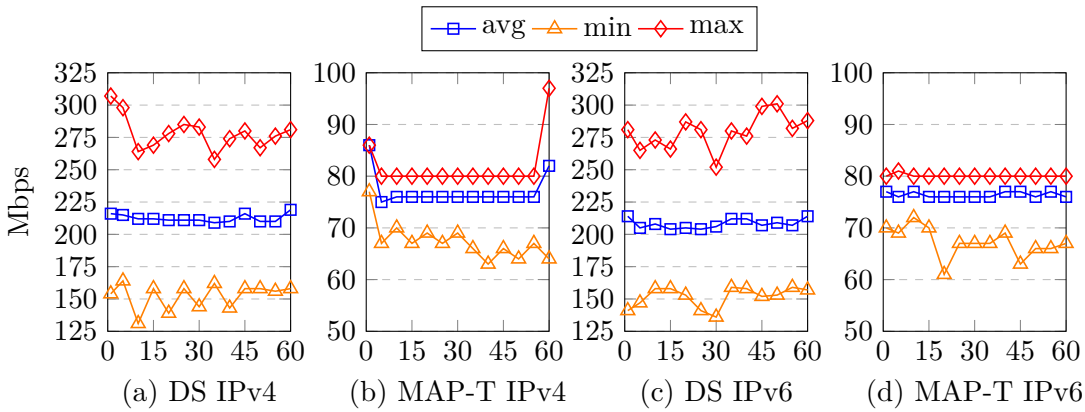

Figure 5.4: Upload throughput, DS trunk vs. MAP CC, parallel IPv4+IPv6 - Average, minimum and maximum upload speeds in Mbps with 4 parallel TCP streams using both IPv4 and IPv6 at the same time. 100 individual measurements of 60 seconds. DS on trunk, MAP-T on CC15.05.1.

The reductions are still very much present. MAP-T IPv4 has a 64% speed reduction (78Mbps vs. 218Mbps) compared to dual-stack, which is, as with the dual download test, better (measured in percent) than in our previous upload test. Reduction on IPv6 is almost the same (81Mbps vs. 214Mbps). The summarized reduction is 63% (159Mbps vs. 432Mbps).

5.2.1.5 Download + upload, IPv4+IPv6: DS vs. MAP

In this test we use 4 parallel TPC streams using IPv4 download and IPv6 upload at the same time. We measured dual-stack (using trunk) and MAP-T (using CC15.05.1). The results are shown in Figure 5.5.

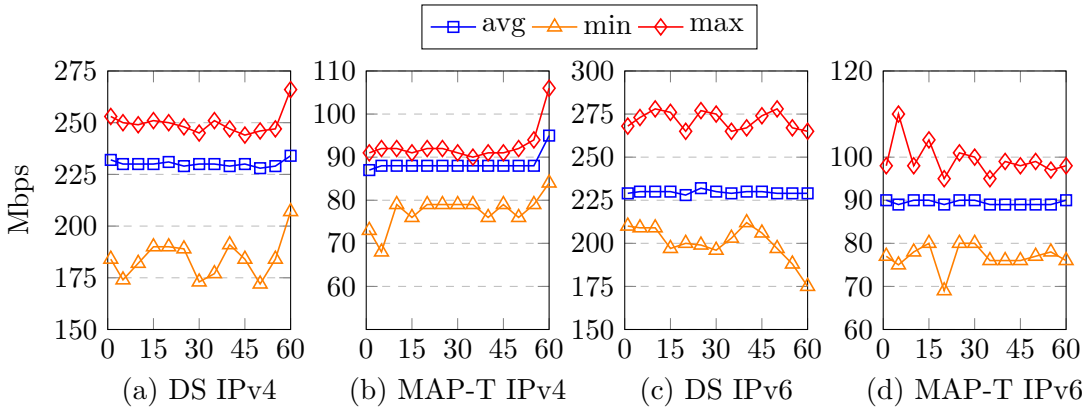

Figure 5.5: Upload + download throughput, DS trunk vs. MAP CC, parallel IPv4+IPv6 - Average, minimum and maximum upload speeds in Mbps with 4 parallel TCP streams using IPv4 download and IPv6 upload at the same time. 100 individual measurements of 60 seconds. DS on trunk, MAP-T on CC15.05.1.

As anticipated, the same reductions are seen. Comparing all of the tests, we clearly see that the measurements are consistent, both regarding the reduced speeds in the MAP-T environment, but also that the throughout it almost the same regardless of upload, download or a combination. As an example, in the test above, we see that the total upload + download throughput using DS is 468Mbps, while pure IPv4 and IPv6 download is 443Mbps and 444Mbps respectively. For pure upload, the numbers for IPv4 and IPv6 are 451Mbps and 439Mbps respectively.

5.2.2 MAP measurements with Designated Driver (trunk)

We wanted to do new measurements in our MAP-T environment using the same release as we did on the dual-stack tests. Due to our issues with the MAP parameters in the last chapter, we ended up with Chaos Calmer 15.05.1 when testing MAP-T. Except for changing the static IPv6 into DHCPv6-PD, the only difference was the OpenWrt version. Since there should be no reason for reduced IPv6 performance using DHCPv6-PD, at least in theory, we found it best to do the tests again, but this time by using the trunk version, Designated Driver, of OpenWrt. Since the MAP issues were resolved, we were confident that MAP-T now would work even with our *nat46* dependency fix.

And truly enough, MAP worked fine using the trunk version (r49161). We did the same throughput tests, using the same parameters. Since we already established the

dual-stack throughputs, we really want to compare the MAP-T (CC15.05.1) throughputs with the MAP-T (trunk) throughputs, which is what the next sections will compare.

5.2.2.1 Download: CC15 vs. trunk

Our first test consists of pure download throughput using 8 parallel TCP streams. We tested IPv4 and IPv6 separately, as before, and compare the previous CC15.05.1 performance with the performance of Designated Driver (trunk). The results are shown in Figure 5.6.

We clearly see that there is a huge difference, especially on IPv6. As suspected, the IPv6 performance issues were caused by that specific OpenWrt version, as the IPv6 numbers now exactly match that of the dual-stack environment (444Mbps vs. 444Mbps). We also notice that IPv4 has increased performance by 42%, going from 131Mbps to 187Mbps.

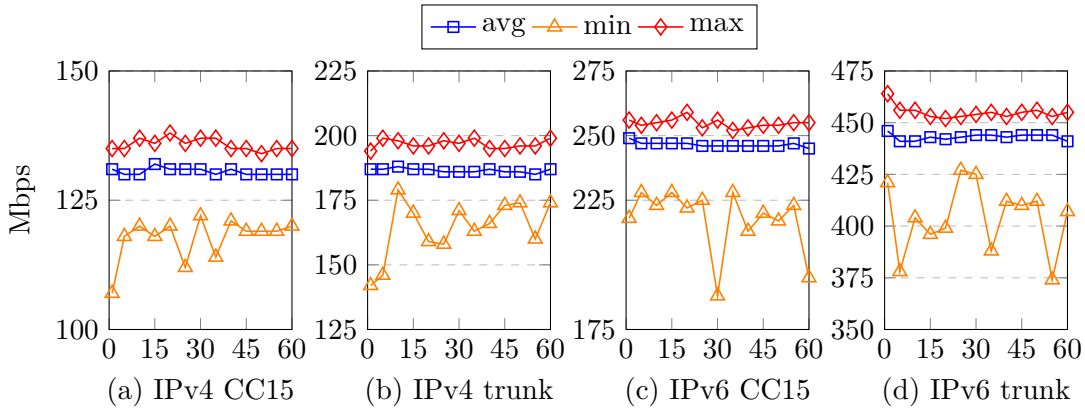

Figure 5.6: MAP-T download throughput, CC15 vs. trunk — Average, minimum and maximum download speeds in Mbps with 8 parallel TCP streams. IPv4 is MAP-T. 100 individual measurements of 60 seconds. Comparing Chaos Calmer (15.05.1) with Designated Driver (from trunk/source, r49161).

5.2.2.2 Download, IPv4+IPv6: CC15 vs. trunk

Next up is downloading with 4 parallel TCP streams, testing IPv4 and IPv6 at the same time, comparing the previous CC15.05.1 performance with the performance of Designated Driver (trunk). The results are shown in Figure 5.7.

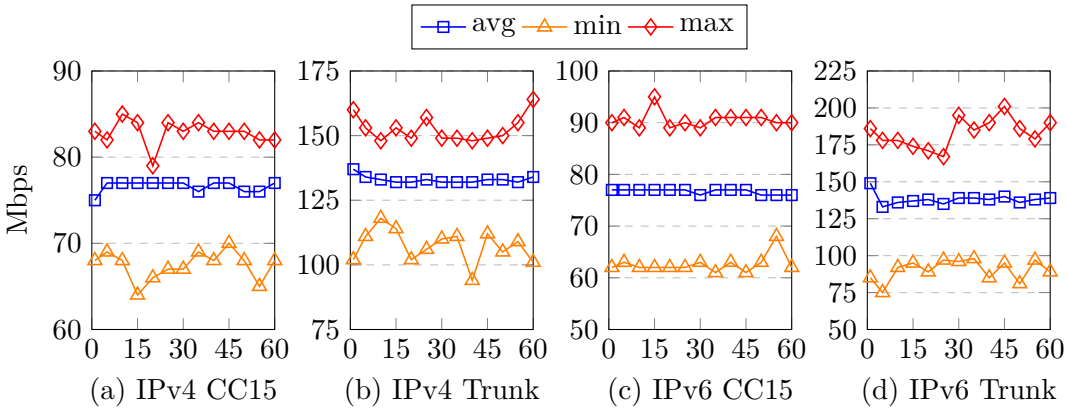

Figure 5.7: MAP-T download throughput, CC15 vs. trunk, parallel IPv4+IPv6 — Average, minimum and maximum download speeds in Mbps with 4 parallel TCP streams on both IPv4 (MAP-T) and IPv6 at the same time. 100 individual measurements of 60 seconds. Comparing Chaos Calmer (15.05.1) with Designated Driver (from trunk/source, r49161).

Again we clearly see an increased throughput with IPv4 increasing with over 70%, going from 78Mbps to 133Mbps. Similarly, IPv6 increases by over 75%, and the total throughput (IPv4 + IPv6) increased with 73%, going from 157Mbps to 272Mbps. We don't see equal IPv6 performance as with dual-stack, but that is to be expected since the IPv4 MAP-T transfer takes its toll on the router when doing parallel tests.

5.2.2.3 Upload: CC15 vs. trunk

We then do the same comparison for the upload speeds, testing 8 parallel TCP streams using IPv4 and IPv6 separately. As before, we compare the previous CC15.05.1 performance with the performance of Designated Driver (trunk). The results are shown in Figure 5.8.

Also on upload we notice that the throughput on IPv6 now is identical to that of dual-stack (439Mbps vs. 439Mbps). The IPv4 performance increased by 32%, going from 125Mbps to 165Mbps, which is less of an increase than with download.

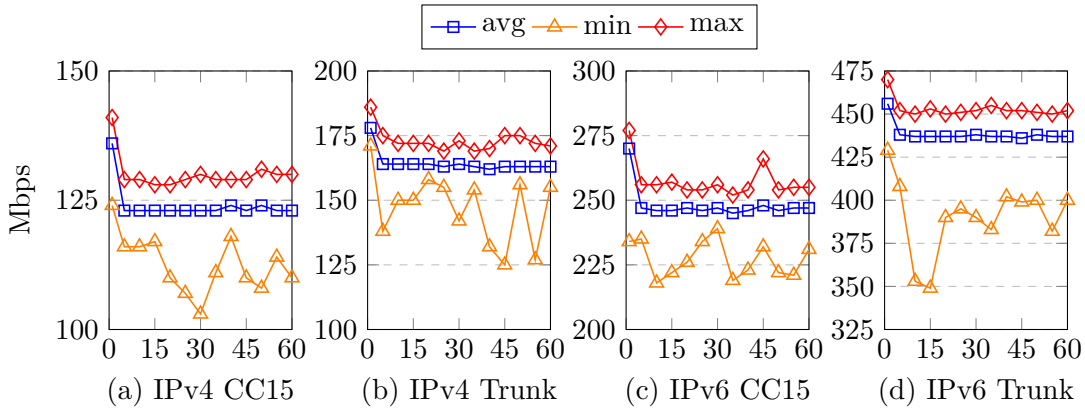

Figure 5.8: MAP-T upload throughput, CC15 vs. trunk — Average, minimum and maximum upload speeds in Mbps with 8 parallel TCP streams. IPv4 is MAP-T. 100 individual measurements of 60 seconds. Comparing Chaos Calmer (15.05.1) with Designated Driver (from trunk/source, r49161).

5.2.2.4 Upload, IPv4+IPv6: CC15 vs. trunk

Testing the upload speed using IPv4 and IPv6 at the same time, we use 4 parallel TCP streams, comparing the previous CC15.05.1 performance with the performance of Designated Driver (trunk). The results are shown in Figure 5.9.

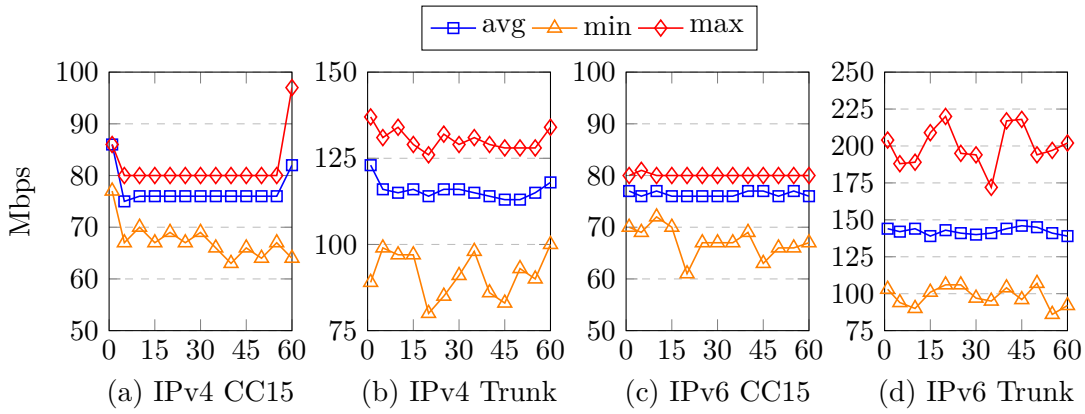

Figure 5.9: MAP-T upload throughput, CC15 vs. trunk, parallel IPv4+IPv6 — Average, minimum and maximum upload speeds in Mbps with 4 parallel TCP streams on both IPv4 (MAP-T) and IPv6 at the same time. 100 individual measurements of 60 seconds. Comparing Chaos Calmer (15.05.1) with Designated Driver (from trunk/source, r49161).

We notice that there is a 48% increase in IPv4 performance, going from 78Mbps to 116Mbps. As for IPv6, there is a 79% performance increase, going from 116Mbps to 145Mbps. The aggregated increase in performance is over 64%, going from 159Mbps to 261Mbps.

5.2.2.5 Download + upload, IPv4+IPv6: CC15 vs. trunk

Having consistently seen much better performance using the Designated Driver version of OpenWrt, we enter our last throughput test, where we measure IPv4 download and IPv6 upload at the same time, using 4 parallel TCP streams for each transfer. We measure the previous CC15.05.1 performance with the performance of Designated Driver (trunk). The results are shown in Figure 5.10.

We see that the IPv4 performance has increased by over 44%, going from 89Mbps to 129Mbps. The IPv6 performance increased by over 48%, going from 93Mbps to 138Mbps. The aggregated performance increased by over 46%, going from 182Mbps to 267Mbps.

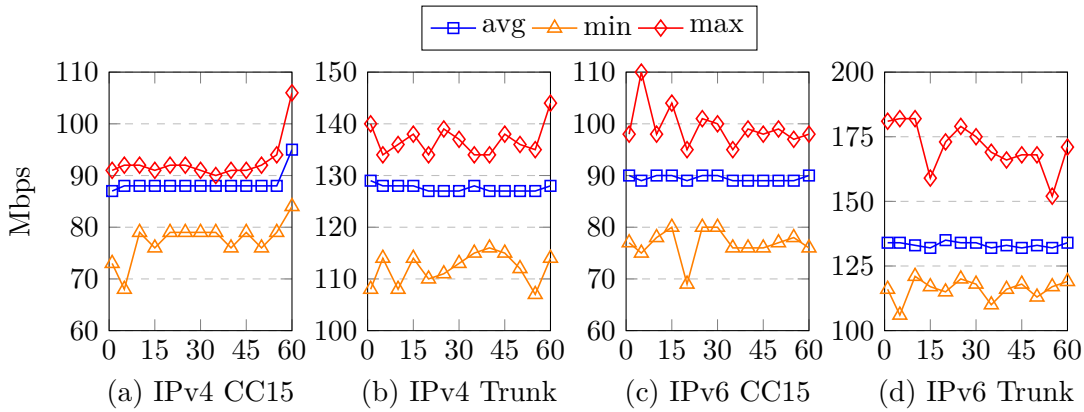

Figure 5.10: MAP-T upload + download throughput, CC15 vs. trunk — Average, minimum and maximum upload + download speeds in Mbps with 4 parallel TCP streams IPv4 (MAP-T) download and IPv6 upload at the same time. 100 individual measurements of 60 seconds. Comparing Chaos Calmer (15.05.1) with Designated Driver (from trunk/source, r49161).

5.3 Throughput + latency

After seeing a huge increase in performance switching from Chaos Calmer 15.05.1 to Designated Driver, we'll also look at some latency measurements. We'll be using Flent to measure latency while performing throughput tests. Flent have many tests available, with different measurements. The test we're using is *http_rrul*, which measures HTTP GET requests while a competing Realtime Response Under Load (RRUL) test is running. RRUL is a test specification written by Dave Taht (101), and measures a network under worst-case conditions, while measuring for latency and web page load times (HTTP GET). The *http_rrul* test runs RTT measurement using ICMP ping and UDP roundtrip time measurement, while loading up the link with eight TCP streams (four downloads, four uploads).

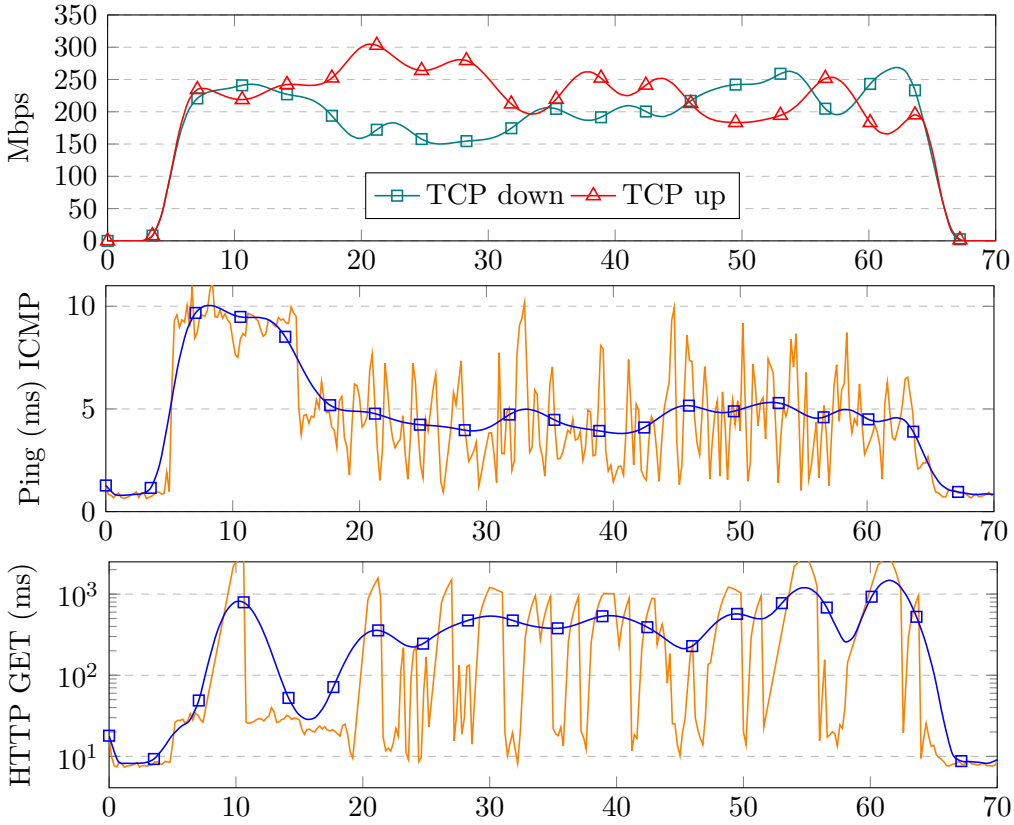

Figure 5.11: Dual-stack: ICMP + HTTP latency - Measuring HTTP and ICMP latency while performing a RRUL test. Measured over a period of 70 seconds on dual-stack while running Designated Driver (trunk).

5.3.1 Dual-stack

Measurements in our dual-stack setup shows comparable throughput performance as with the iperf3 tests, which is according to expectation. The ICMP latency have normal values throughout the test, whilst the HTTP GET latency is higher than expected. The measurements can be seen in Figure 5.11.

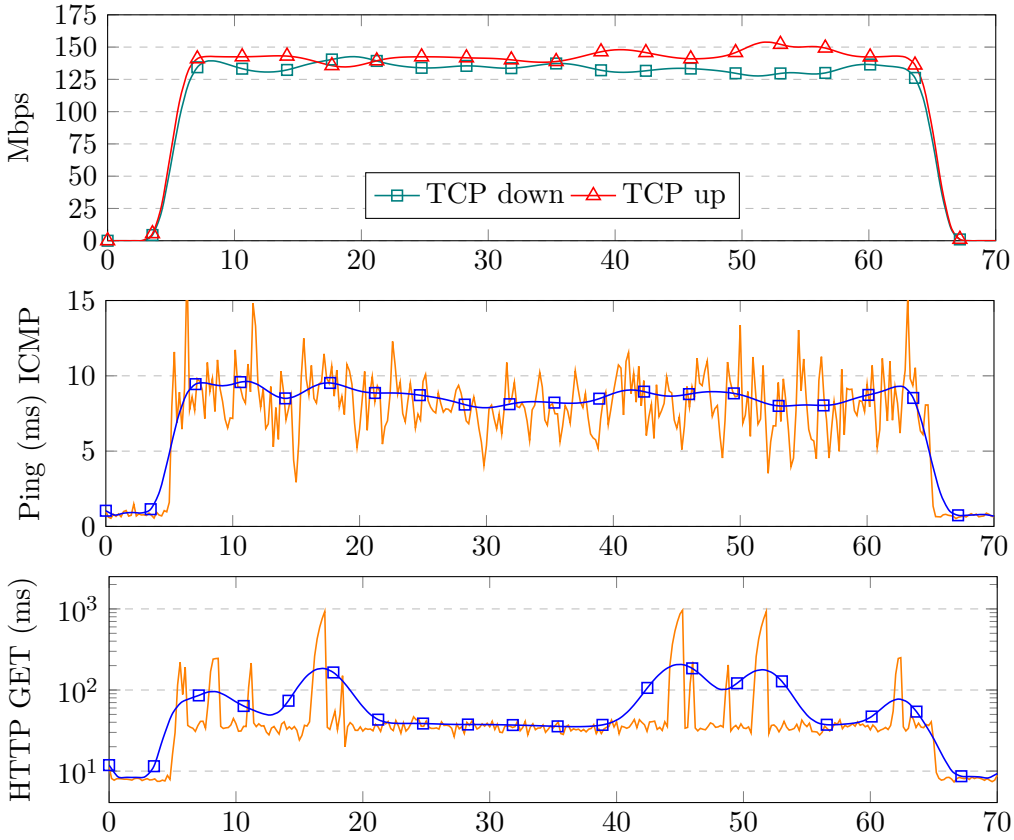

Figure 5.12: MAP-T CC15.05.1: ICMP + HTTP latency - Measuring HTTP and ICMP latency while performing a RRUL test. Measured over a period of 70 seconds on MAP while running Chaos Calmer 15.05.1.

5.3.2 MAP-T, Chaos Calmer 15.05.1

Moving to our MAP-T setup using CC15.05.1, the measurements show similar throughput performance as with the MAP-T CC15.05.1 iperf3 tests. Due to an aggregated throughput of around 250Mbps, the tests obviously have to be using IPv6 for the TCP streams. When launching the tests, we use the hostname, which resolves both the IPv4

and IPv6 addresses, and hence, it chooses IPv6 over IPv4. Later will be doing tests with IPv4 only, and combined IPv4 + IPv6 to see how that affects the measurements. Both ICMP- and HTTP GET latency are stable, and the HTTP GET latency is considerable lower and more stable in this test, compared to the dual-stack test. The measurements can be seen in Figure 5.12.

5.3.3 MAP-T, Designated Driver

Trying the same MAP-T measurements in Designated Driver, we see increased throughput, and also lower ICMP latency. The HTTP GET latency is the worst so far of our measurements, having latency just below a whole second. The HTTP GET only fetches one single file with a size of 4 bytes, so one second is quite much. The measurements can be seen in Figure 5.13.

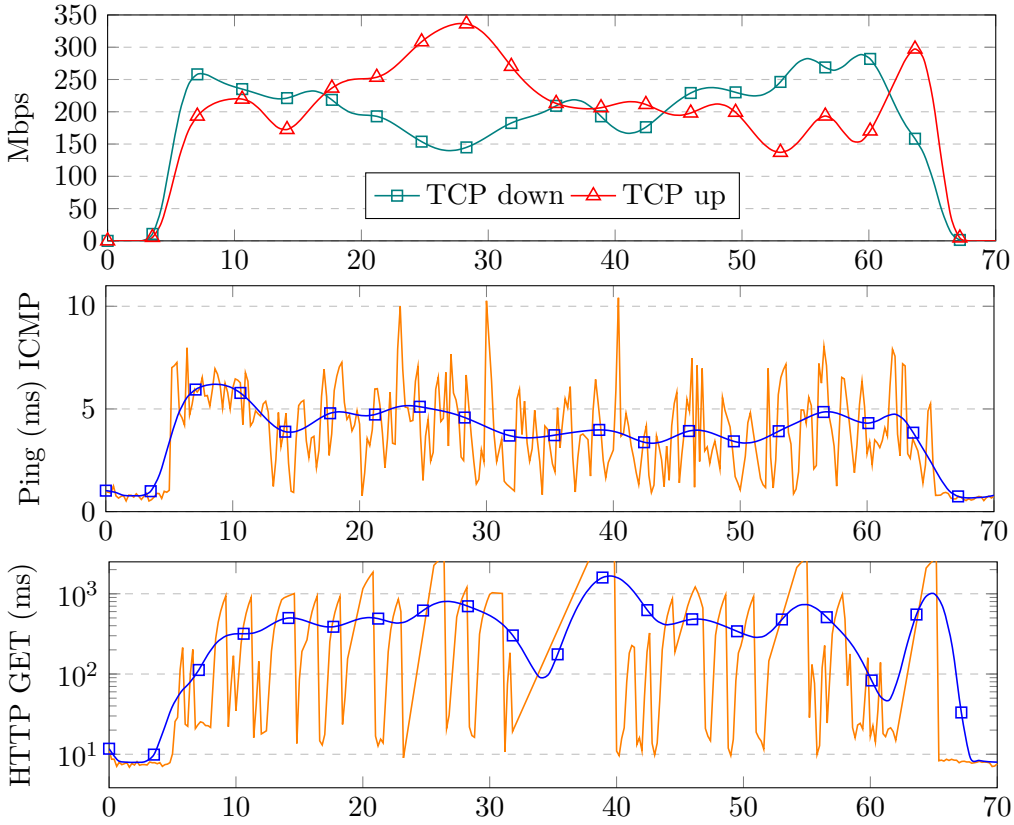

Figure 5.13: MAP trunk: ICMP + HTTP latency - Measuring HTTP and ICMP latency while performing a RRUL test. Measured over a period of 70 seconds on MAP while running Designated Driver (trunk).

5.3.4 MAP-T, Designated Driver, IPv4

Since the measurements in our MAP-T setup utilizes IPv6 before IPv4, we've basically only tested latency when doing IPv6 throughput. We therefore did an identical test, this time forcing the connection to IPv4. That means that the TCP streams, ICMP latency and HTTP GET latency are all going over IPv4, and hence, MAP-T. Looking at Figure 5.14, we see a drastic reduce in throughput, while having high latency. The HTTP GET requests timed out, or took too long, so we had to exclude those results.

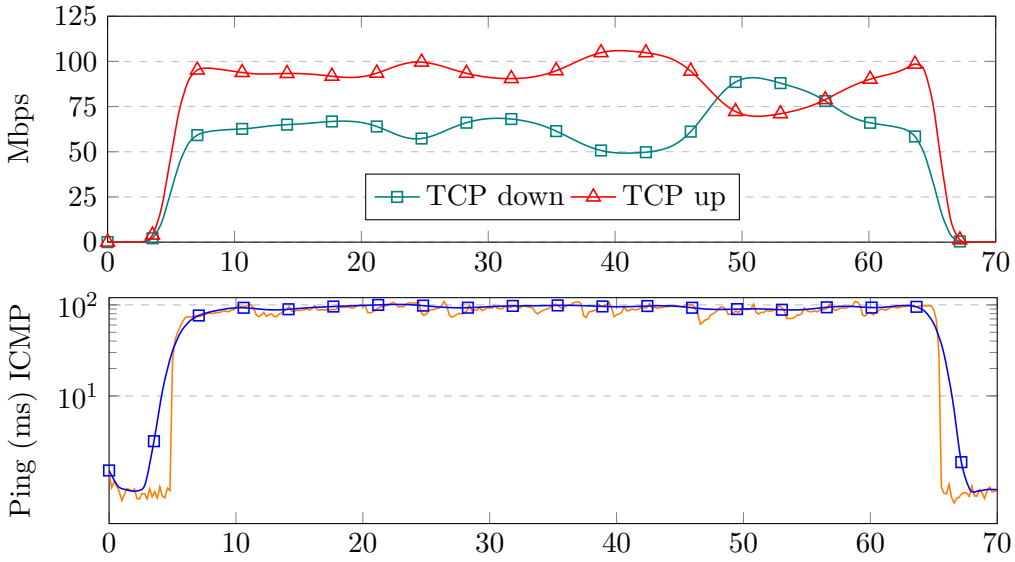

Figure 5.14: MAP trunk: IPv4, ICMP + HTTP latency - Measuring HTTP and ICMP latency while performing a RRUL test. Measured over a period of 70 seconds on MAP forced to use IPv4 while running Designated Driver (trunk).

5.3.5 MAP-T, Designated Driver, RRUL46

Our last measurement utilizes the *rrul46* test, which is a RRUL test, but with mixed IPv4 and IPv6 streams. Looking at Figure 5.15, we see higher throughputs than in the IPv4 only MAP-T test above, and we also see considerably lower ICMP latency. We believe this is the measurement that best depicts a real user scenario, where multiple types of upload and download are performed, utilizing both IPv4 and IPv6.

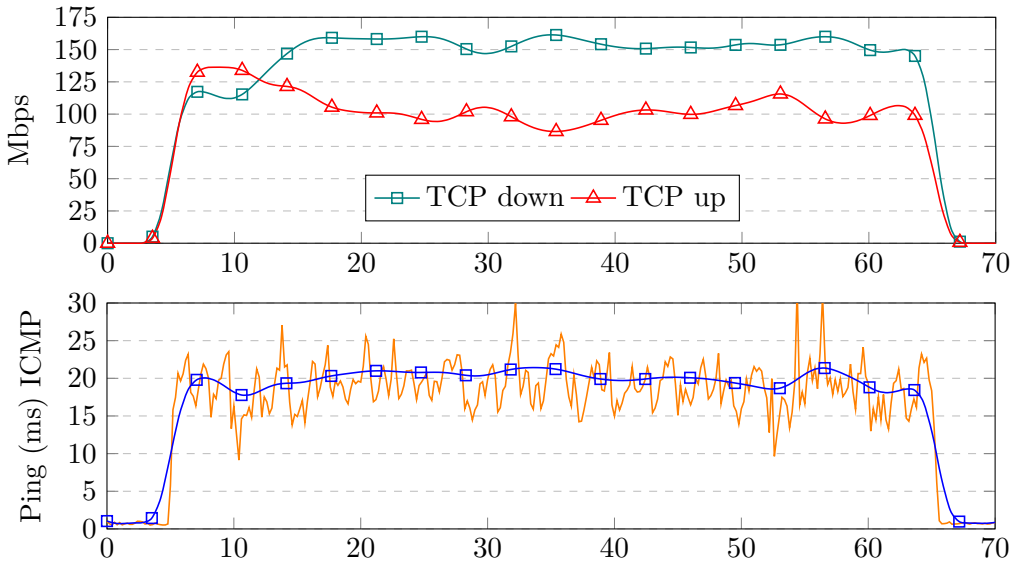

Figure 5.15: MAP trunk: RRUL46, ICMP + HTTP latency - Measuring ICMP latency while performing a RRUL46 test. Measured over a period of 70 seconds on MAP while running Designated Driver (trunk).

5.4 End-user usability

In our last type of measurement, we want to look at the usability for the end-user. These tests are somewhat empirical, in the way that we have not tested any of them in theory, and they have not been systematically tested. Most of them have been tested more than one time, confirming that it's not a one-off. Our tests are done by connecting a Windows 7 client to the CPE when in MAP-T mode. That computer was then used for daily tasks for four weeks, using it as we'd normally do.

As we tested different services, we noted them down in Table 5.1, and gave them passed/failed depending on the outcome. A similar test was done by Edwin Cordeiro et al. in (102). They had issues with torrent clients, and also FTP in passive mode. We did not experience these issues; we seeded numerous torrents without issues, having several MB/s in upload speeds. We also connected to FTP just fine on this Windows 7 client using WinSCP. We could browse files, list content of folders, and download files (even larger ones, 4+ GB).

We also tested several online games, and all of them were classified as "Moderate NAT" by the game when joining online multiplayer games. Normal web browsing had

no issues, even online streaming content. We also tested video streaming using Plex, and we got a 10/10 score on test-ipv6.com.

Service	Comment	Result
Games	Steam (IPv4)	Passed
Games	uPlay (IPv4)	Passed
Games	Origin (IPv4)	Passed
Online games	Counter Strike: Global Offensive (IPv4)	Passed
Online games	Rainbow Six: Siege (IPv4)	Passed
Online games	Black Mesa (IPv4)	Passed
Voice	Ventrilo (IPv4)	Passed
Voice	TeamSpeak (IPv4)	Passed
Torrent	uTorrent (IPv4)	Passed
FTP	WinSCP (passive, IPv4)	Passed
FTP	WinSCP (passive, IPv6)	Passed
FTP	WinSCP (active, IPv4)	Failed
FTP	WinSCP (active, IPv6)	Passed
Browsing	www.google.com (dual-stack)	Passed
Browsing	www.ba.no (IPv4)	Passed
IPv6 test	www.test-ipv6.com	Passed (10/10)
Video	Plex video streaming (IPv4)	Passed
Video	www.plex.tv (IPv4)	Passed
Video	tv.nrk.no (IPv4)	Passed
Video	www.youtube.com (dual-stack)	Passed

Table 5.1: End-user usability - A list of services, applications and web sites tested behind our MAP-T setup.

We clearly see that MAP-T works in favor of the end-user, and no major connectivity issues where seen. Our experience from using the Windows 7 client is that it feels like any other Internet access we've used, and we'll probably continue to use it as-is for the foreseeable future.

6

Conclusion

In this thesis we've looked at how the Internet has rapidly grown, and that there are no more IPv4 address space available. A transition to IPv6 is inevitable, and as such, IPv6 transition mechanisms are needed. We've looked at different transition mechanisms, how they work, and which of them we found most applicable for new network operators with scarce IPv4 resources. Looking at Mapping of Address and Port, we've addressed how it works, what the limitations are, and how to deploy it in our setup.

We've looked at different types of CPEs and BRs, where finding a CPE matching our criteria proved to be a challenge. We acknowledge the challenges of using a CPE with OpenWrt – especially since it's not an out-of-the-box solution – however, we've proven that a commodity CPE can deliver good results, both in features, and also throughput and performance.

The huge differences in throughput were interesting, especially the difference in IPv6 throughput on Chaos Calmer compared to Designated Driver – we thought they would be much more identical, but we also reckon that there are lots of changes in the codebase between the two versions that can explain the difference in performance.

IPv4 throughput using MAP mode on the CPE could be better, especially in FTTH situations where bandwidths above 100Mbps is more common. But, taking into consideration that it's a cheap CPE, MAP-T IPv4 throughputs of almost 200Mbps is good enough for most scenarios. Issues with high and varying latency was expected, as this is normal behavior in best-effort setups – high throughput usually equals high latency.

We find it strange that there's little movement in the CPE market regarding MAP support – almost a year after the release of the two finalized MAP standard tracks

there doesn't seem to be off-the-shelves CPEs that matches our requirements (and we don't think our requirements are unreasonable almost two decades after IPv6 came to life).

We believe MAP is a really mature platform to utilize IPv4 address sharing over IPv6-only networks. It scales very well, and is simple to deploy.

6.1 Future work

We find several things that could be done in future work;

- Performance testing using VPP as BR (70), looking at how far it could scale using commodity servers. Test how stable it would be, and how easily it could be deployed in production.
- Performance and latency testing on the WiTi CPE (80), comparing it to the Archer C7.
- Implement QoS and shaping to mitigate latency and bufferbloat issues. The Bufferbloat project has done much work in this area that looks really promising (103).
- Make administrative systems for deploying MAP rules and DHCPv6 MAP parameters. Currently it's a cumbersome and manual process.
- Look at how IPv4 multicast could be solved. Either through draft-sarikaya-softwire-map-multicast-04 (56), draft-ietf-softwire-dslite-multicast-11 (104), or other means.
- Website that utilizes the MAP parameters/rules to display public IPv4 and port-sets for an end-user visiting that page. Should only be accessible from within the network of an operator, and should only display information about that specific user visiting the page. This way end-users can know what ports and port-sets that are available. Useful if the user wants to host services on the shared IPv4 address.

References

[1] **Measuring the Information Society Report 2015** [online]. 2015 [cited 18-Apr-2016 08:20 CEST]. 1

[2] Geoff Huston. **IPv4 Address Report** [online, cited 16-Apr-2016 08:20 CEST]. 1

[3] **Available Pool of Unallocated IPv4 Internet Addresses Now Completely Emptied** [online]. February 3, 2011 [cited 15-Apr-2016 08:31 CEST]. 1

[4] **Free Pool of IPv4 Address Space Depleted** [online]. February 3, 2011 [cited 16-Apr-2016 18:20 CEST]. 1

[5] Stephen E. Deering and Robert M. Hinden. **RFC2460: Internet Protocol, Version 6 (IPv6)**. December 1998. 3

[6] **IPv6 measurements for The World** [online, cited 21-Apr-2016 23:20 CEST]. 3

[7] Alan Oxley. **Issues affecting the adoption of IPv6**. 2014. 3

[8] Neal Leavitt. **IPv6: Any Closer to Adoption?** September 2011. 3

[9] Mehdi Nikkhah and Roch Guérin. **Migrating to Ipv6 — The role of basic coordination**. June 2014. 3

[10] Pyda Srisuresh and Matt Holdrege. **RFC2663: IP Network Address Translator (NAT) Terminology and Considerations**. August 1999. 5, 17

[11] Pyda Srisuresh and Kjeld Borch Egevang. **RFC3022: Traditional IP Network Address Translator (Traditional NAT)**. January 2001. 5

[12] Yakov Rekhter, Robert G Moskowitz, Daniel Karrenberg, Geert Jan de Groot, and Eliot Lear. **RFC1918: Address Allocation for Private Internets**. February 1996. 5

[13] Jason Weil, Victor Kuarsingh, Chris Donley, Christopher Liljenstolpe, and Marla Azinger. **IANA-Reserved IPv4 Prefix for Shared Address Space**. April 2012. 5

[14] Chris Donley, Lee Howard, Victor Kuarsingh, John Berg, and Jinesh Doshi. **RFC7021: Assessing the Impact of Carrier-Grade NAT on Network Applications**. September 2013. 5

[15] Mat Ford, Mohamed Boucadair, Alain Durand, Pierre Levis, and Phil Roberts. **RFC6269: Issues with IP Address Sharing**. June 2011. 5

[16] Nejc Škoberne, Olaf Maennel, Iain Phillips, Randy Bush, Jan Zorz, and Mojca Ciglaric. **IPv4 Address Sharing Mechanism Classification and Tradeoff Analysis**. April 2014. 5, 7

REFERENCES

[17] Enrico Bocchi, Ali Safari Khatouni, Stefano Traverso, Alessandro Finamore, Valeria Di Gennaro, Marco Mellia, Maurizio Munafò, and Dario Rossi. **Impact of Carrier-Grade NAT on Web Browsing**. August 2015. 5

[18] Vince Fuller, Eliot Lear, and David Meyer. **Reclassifying 240/4 as usable unicast address space**. March 24, 2008. 5

[19] Paul Saab. **World IPv6 Congress 2015: Facebook** [online]. 2015. 5

[20] Paul Saab. **Scale 2015: Facebook IPv6** [online]. 2015. 5

[21] Jonghwan Hyun, Jian Li, Hwankuk Kim, Jae-Hyoung Yoo, and James Won-Ki Hong. **IPv4 and IPv6 Performance Comparison in IPv6 LTE Network**. August 2015. 5

[22] Yuk-Nam Law, Man-Chiu Lai, Wee Lum Tan, and Wing Cheong Lau. **Empirical Performance of IPv6 vs. IPv4 under a Dual-Stack Environment**. May 2008. 5

[23] Fuliang Li, Xingwei Wang, Tian Pan, and Jiahai Yang. **Packet Delay, Loss and Reordering in IPv6 World: A Case Study**. February 2016. 5

[24] Vaibhav Bajpai and Jürgen Schönwälder. **IPv4 versus IPv6 - Who connects faster?** May 2015. 5

[25] **UK sells off unused net addresses** [online]. 2015. 6

[26] **Microsoft Offers $7.5 Million to Buy 666,624 IPv4 Addresses** [online]. 6

[27] **RIPE NCC: Last /8 Phases** [online]. 6

[28] Mark Townsley. **Mapping Address + Port**, May 2012. 7

[29] **List of IPv6 tunnel brokers** [online, cited 27-Apr-2016 20:10 CEST]. 6

[30] **Call Your ISP for IPv6!** [online, cited 27-Apr-2016 20:13 CEST]. 6

[31] Marcelo Bagnulo, Philip Matthews, and Iljitsch van Beijnum. **RFC6146: Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers**. April 2011. 9

[32] Marcelo Bagnulo, Andrew Sullivan, Philip Matthews, and Iljitsch van Beijnum. **RFC6147: DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers**. April 2011. 9

[33] Masataka Mawatari, Masanobu Kawashima, and Cameron Byrne. **RFC6877: 464XLAT: Combination of Stateful and Stateless Translation**. April 2013. 9

[34] Xing Li, Congxiao Bao, Wojciech Dec, Ole Troan, Satoru Matsushima, and Tetsuya Murakami. **RFC7599: Mapping of Address and Port using Translation (MAP-T)**. July 2015. 9

[35] Alex Conta and Stephen Deering. **RFC2473: Generic Packet Tunneling in IPv6**. December 1998. 9, 17

[36] Alain Durand, Ralph Droms, James Woodyatt, and Yiu L. Lee. **RFC6333: Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion**. August 2011. 10

[37] Yong Cui, Qiong Sun, Mohamed Boucadair, Tina Tsou, Yiu L. Lee, and Ian Farrer. **RFC7596: Lightweight 4over6: An Extension to the Dual-Stack Lite Architecture**. July 2015. 10

[38] Ole Troan, Wojciech Dec, Xing Li, Congxiao Bao, Satoru Matsushima, Tetsuya Murakami, and Tom Taylor. **RFC7597: Mapping of Address and Port with Encapsulation (MAP-E)**. July 2015. 10

[39] Remi Despres, Sheng Jiang, Reinaldo Penno, Yiu Lee, Gang Chen, and Maoke Chen. **RFC7600: IPv4 Residual Deployment via IPv6 - A Stateless Solution (4rd)**. July 2015. 10

[40] **Free peut attribuer la même adresse IP à plusieurs abonnés** [online, cited 28-Apr-2016 04:50 CEST]. 10

[41] Erik Nordmark and Robert E. Gilligan. **RFC4213: Basic Transition Mechanisms for IPv6 Hosts and Routers**. October 2005. 10

[42] Brian E. Carpenter and Cyndi Jung. **RFC2529: Transmission of IPv6 over IPv4 Domains without Explicit Tunnels**. March 1999. 11

[43] Fred L. Templin, Tim Gleeson, and Dave Thaler. **RFC5214: Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)**. March 2008. 11

[44] Brian Carpenter. **RFC6343: Advisory Guidelines for 6to4 Deployment**. August 2011. 11

[45] Brian E. Carpenter and Keith Moore. **RFC3056: Connection of IPv6 Domains via IPv4 Clouds**. February 2001. 11

[46] Christian Huitema. **RFC4380: Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)**. February 2006. 11

[47] **IPv6 rapid deployment: Current usage** [online, cited 28-Apr-2016 00:08 CEST]. 11

[48] Alexandre Cassen. **IPv6 Free; Native IPv6 to the User**. May 2009. 11

[49] **Free déploie l'IPv6** [online]. December 2007. 11

[50] Marius Georgescu, Hiroaki Hazeyama, Takeshi Okuda, Youki Kadobayashi, and Suguru Yamaguchi. **The STRIDE Towards IPv6: A Comprehensive Threat Model for IPv6 Transition Technologies**. April 2016. 13

[51] Guo liang Han, Cong xiao Bao, and Xing Li. **A scalable and efficient IPv4 address sharing approach in IPv6 transition scenarios**. May 2015. 14, 46

[52] Roberta Maglione, Wojciech Dec, Ida Leung, and Edwin Mallette. **Use cases for MAP-T**. October 14, 2015. 14

[53] Matt Mathis and John W. Heffner. **RFC4821: IPv6 Addressing of IPv4/IPv6 Translators**. March 2007. 14

[54] **Softwires Working Group** [online, cited 30-Apr-2016 07:02 CEST]. 15

[55] **IETF Softwire WG meeting, IETF 84, Thursday August 2, 2012** [online, cited 28-Apr-2016 04:45 CEST]. 15

[56] Behcet Sarikaya and Hui Ji. **Multicast Support for Mapping of Address and Port Protocol and Light Weight 4over6**. June 8, 2015. 15, 72

[57] Ole Troan and Ralph Droms. **RFC3633: IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6**. December 2003. 16, 24

[58] Susan Thomson, Thomas Narten, and Tatuya Jinmei. **RFC4862: IPv6 Stateless Address Autoconfiguration**. September 2007. 16

[59] Xing Li, Congxiao Bao, and Fred Baker. **RFC6145: IP/ICMP Translation Algorithm**. April 2011. 23

[60] Xing Li, Congxiao Bao, Dan Wing, Ramji Vaithianathan, and Geoff Huston. **RFC6791: Stateless Source Address Mapping for ICMPv6 Packets**. November 2012. 23

[61] Congxiao Bao, Christian Huitema, Marcelo Bagnulo, Mohamed Boucadair, and Xing Li. **RFC6052: IPv6 Addressing of IPv4/IPv6 Translators**. October 2010. 23

[62] Tomek Mrugalski, Ole Troan, Ian Farrer, Simon Perreault, Wojciech Dec, Congxiao Bao, Leaf Y. Yeh, and Xiaohong Deng. **RFC7598: DHCPv6 Options for Configuration of Softwire Address and Port-Mapped Clients**. July 2015. 24, 35

[63] **JANOG31: Softwire Working Group** [online, cited 29-Apr-2016 19:31 CEST]. 30, 32

[64] **JANOG Softwire Working Group: MAP-E** [online, cited 29-Apr-2016 19:31 CEST]. 30, 32

[65] **CERNET MAP: An open source CPE implementation of MAP-E/MAP-T** [online, cited 29-Apr-2016 19:31 CEST]. 30, 32

[66] **Tiny MAP-E implementation** [online, cited 29-Apr-2016 19:31 CEST]. 30

[67] **Cisco IOS XRv Router Installation and Configuration Guide** [online, cited 29-Apr-2016 19:31 CEST]. 30

[68] **Cisco Cloud Services Router 1000V Series** [online, cited 29-Apr-2016 19:31 CEST]. 30

[69] Masakazu Asama. **MAP supported Vyatta (as known as ASAMAP)** [online, cited 29-Apr-2016 19:31 CEST]. 30, 32

[70] **The Linux Foundation Forms Open Source Effort to Advance IO Services** [online, cited 29-Apr-2016 19:31 CEST]. 30, 72

[71] **VPP: Configure an MAP-E Terminator** [online, cited 29-Apr-2016 19:31 CEST]. 30

[72] Masanobu Saitoh and Hiroki Suenaga. **Developing CPE Routers based on NetBSD: Fifteen Years of SEIL**. March 2014. 32

[73] **SEIL, Internet Initiative Japan Inc** [online, cited 29-Apr-2016 19:31 CEST]. 32

[74] **OpenWrt** [online, cited 29-Apr-2016 19:31 CEST]. 32

[75] **SEIL, Technical Manual** [online, cited 29-Apr-2016 19:31 CEST]. 33

[76] **OpenWrt: Table of Hardware, full details** [online, cited 29-Apr-2016 19:31 CEST]. 33

[77] **TP-Link: Archer C7 AC1750** [online, cited 29-Apr-2016 19:31 CEST]. 33

[78] **OpenWrt: TP-Link Archer C7 AC1750** [online, cited 29-Apr-2016 19:31 CEST]. 33, 42, 55

[79] **OpenWrt: Recommended routers** [online, cited 29-Apr-2016 19:31 CEST]. 33

[80] **WiTi – An Open Router Platform Is Coming!** [online, cited 29-Apr-2016 19:31 CEST]. 34, 72

[81] **Raycore: RC-OE2ATR Series Unmanaged media converter** [online, cited 29-Apr-2016 19:31 CEST]. 34

[82] **The Open Source WRT54G Story** [online, cited 29-Apr-2016 19:31 CEST]. 35

[83] **Linksys Releases GPLed Code for WRT54G** [online, cited 29-Apr-2016 19:31 CEST]. 35

[84] Andrew Yourtchenko. **Run your next CGN on a $20 OpenWRT**, 2013. 35

[85] **openwrt-map** [online, cited 29-Apr-2016 19:31 CEST]. 35

[86] Tomasz Mrugalski, Ole Troan, Wojciech Dec, Congxiao Bao, and Leaf Y. Yeh end Xiaohong Deng. **draft-ietf-softwire-map-dhcp-03: DHCPv6 Options for Mapping of Address and Port**. February 25, 2013. 35

[87] **iPerf - The network bandwidth measurement tool** [online, cited 29-Apr-2016 19:31 CEST]. 36

[88] **iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool** [online, cited 29-Apr-2016 19:31 CEST]. 36

[89] **Netperf Homepage** [online, cited 29-Apr-2016 19:31 CEST]. 36

[90] **Flent: The FLExible Network Tester** [online, cited 29-Apr-2016 19:31 CEST]. 36

[91] **Flent: The FLExible Network Tester - Source** [online, cited 29-Apr-2016 19:31 CEST]. 36

[92] **Comparison of platform virtualization software** [online, cited 30-Apr-2016 23:33 CEST]. 36

[93] **Proxmox Virtual Environment** [online, cited 30-Apr-2016 23:31 CEST]. 36

[94] **LXC (Linux Containers)** [online, cited 30-Apr-2016 23:38 CEST]. 37

[95] **Debian "jessie" Release Information** [online, cited 30-Apr-2016 23:38 CEST]. 37

[96] **OpenWrt Ticket #21358: global error** [online, cited 01-May-2016 02:46 CEST]. 43

[97] **OpenWrt Routing: dependencies for map-t (kmod-ipv6)** [online, cited 01-May-2016 02:46 CEST]. 43, 45

[98] **OpenWrt's build system – About** [online, cited 01-May-2016 02:46 CEST]. 44

[99] **Cisco MAP Simulation Tool (beta)** [online, cited 01-May-2016 04:46 CEST]. 53

[100] Marius Georgescu, Hiroaki Hazeyama, Takeshi Okuda, Youki Kadobayashi, and Suguru Yamaguchi. **Benchmarking the Load Scalability of IPv6 Transition Technologies: a Black-Box Analysis**. July 2015. 55

[101] **Realtime Response Under Load (RRUL) Test** [online, cited 02-May-2016 07:46 CEST]. 65

[102] Edwin Cordeiro, Rodrigo Carnier, and Antonio Marcos Moreiras. **RFC7703: Experience with Testing of Mapping of Address and Port Using Translation (MAP-T)**. November 2015. 69

[103] **The Bufferbloat project** [online, cited 02-May-2016 16:33 CEST]. 72

[104] Jacni Qin, Mohamed Boucadair, Christian Jacquenet, Yiu L. Lee, and Qian Wang. **draft-ietf-softwire-dslite-multicast-11: Delivery of IPv4 Multicast Services to IPv4 Clients over an IPv6 Multicast Network**. February 26, 2016. 72

REFERENCES

[105] **IANA IPv4 Address Space Registry** [online, cited 24-Apr-2016 22:46 CEST].

[106] Akira Nakagawa. **Operational Experience of MAP-E**, July 24, 2015.

[107] Akira Nakagawa. **JPNE MAP-E Deployment**, Mar.25.2015.

[108] Luc De Ghein. **CCSI: Transition Solutions for IPv6**, 2014.

[109] **Mapping of Address and Port Using Translation** [online, cited 16-Apr-2016 08:20 CEST].

[110] **MAP-T configuration on OpenWRT** [online, cited 16-Apr-2016 08:20 CEST].

[111] **OpenWRT on kvm with MAP-T example** [online, cited 16-Apr-2016 08:20 CEST].

[112] Gabor Bajko, Mohamed Boucadair, Steven M. Bellovin, Randy Bush, Luca Cittadini, Olaf Maennel, Reinaldo Penno, Teemu Savolainen, and Jan Zorz. **RFC6346: The Address plus Port (A+P) Approach to the IPv4 Address Shortage**. August 2011.

[113] Gang Chen, Zhen Cao, Chongfeng Xie, and David Binet. **RFC7269: NAT64 Deployment Options and Experience**. June 2014.

[114] Cedric Aoun and Elwyn B. Davies. **RFC4966: Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status**. July 2007.

[115] George Tsirtsis and Pyda Srisuresh. **RFC2766: Network Address Translation - Protocol Translation (NAT-PT)**. February 2000.

[116] Xing Li, Congxiao Bao, Maoke Chen, Hong Zhang, and Jianping Wu. **RFC6219: The China Education and Research Network (CERNET) IVI Translation Design and Deployment for the IPv4/IPv6 Coexistence and Transition**. May 2011.

Appendix A

Scripts & config

```
#!/usr/bin/perl
use warnings;
use strict;
use POSIX;

my %stats;

while (my $line=<STDIN>){
  chomp($line);
  my ($sec, $mbps) = split(',', $line);

  $stats{$sec}{avg} += $mbps; # add total, which is used as avg
  $stats{$sec}{count}++; # how many tests has been done?

  if($stats{$sec}{max}){
    if($mbps > $stats{$sec}{max}){
      $stats{$sec}{max} = $mbps;
    }
  } else {
    $stats{$sec}{max} = $mbps;
  }

  if($stats{$sec}{min}){
    if($mbps < $stats{$sec}{min}){
      $stats{$sec}{min} = $mbps;
    }
  } else {
    $stats{$sec}{min} = $mbps;
  }
}
```

```
31
32 print "sec,avg,min,max\n";
33 foreach my $second (sort { $a <=> $b } keys %stats){
34   print "$second," . ceil($stats{$second}{avg} / $stats{$second}{count}) .
       ",$stats{$second}{min},$stats{$second}{max}\n";
35 }
```

Listing A.1: Make average numbers from CSV-output.

```
1 #!/usr/bin/perl
2 use warnings;
3 use strict;
4 use JSON -support_by_pp;
5 use POSIX;
6
7 my @json_content = <STDIN>;
8 my $json = new JSON;
9 my $json_text = $json->allow_nonref->utf8->relaxed->escape_slash->loose->
    allow_singlequote->allow_barekey->decode("@json_content");
10
11 sub sort_streams{
12   my $start = shift;
13   $start = floor($start);
14   return $start;
15 }
16
17 my $i = 1;
18 foreach my $interval (sort { sort_streams($a->{sum}->{start}) <=>
    sort_streams($b->{sum}->{start}) } @{$json_text->{intervals}}){
19   my $mbps = floor($interval->{sum}->{bits_per_second} / 1000 / 1000);
20   print "$i,$mbps\n";
21   $i++;
22 }
```

Listing A.2: Make CSV-output based on JSON from iperf3.

```
1 #!/usr/bin/env perl
2 use strict;
3 use warnings;
4 use Net::IP;
5 use NetAddr::IP;
6 use POSIX;
7
8 # static stuff
9 my $option_s46_rule = sprintf("%04x", 89);
```

```
10 my $option_s46_dmr = sprintf("%04x", 91);
11 my $option_s46_portparams = sprintf("%04x", 93);
12 my $option_s46_cont_mapt = sprintf("%04x", 95);
13
14 # dynamic stuff
15 my $ipv6_prefix = "2a02:4260:f800::/48";
16 my $ipv4_prefix = "185.54.31.128/28";
17 my $dmr_prefix = "2a02:4260:f000:0004::/64";
18 my $ea_length = 8;
19 my $psid_offset = 6;
20
21 sub packdhcp{
22   my ($dhcp_option, $hex_value) = @_;
23
24   my $length = floor(length($hex_value) / 2);
25
26   return ($dhcp_option . sprintf("%02x%02x",
27     floor($length/256), ($length % 256)) .
28     $hex_value);
29 }
30
31 sub create_bmr{
32   my $ipv4 = Net::IP->new($ipv4_prefix);
33   my $ipv6 = Net::IP->new($ipv6_prefix);
34
35   (my $ipv4_addr = $ipv4->hexip()) =~ s/://g;
36   $ipv4_addr =~ s/^0x//;
37   (my $ipv6_addr = $ipv6->short()) =~ s/://g;
38
39   my $bmr = sprintf("%02x%02x%02x%s%02x%s",
40     0,
41     $ea_length,
42     $ipv4->prefixlen(),
43     $ipv4_addr,
44     $ipv6->prefixlen(),
45     $ipv6_addr
46     );
47
48   return packdhcp($option_s46_rule, $bmr .
49     packdhcp($option_s46_portparams,
50     sprintf("%02x000000", $psid_offset)));
51 }
52
53 sub create_dmr{
54   my $dmr = Net::IP->new($dmr_prefix);
```

```
55
56    (my $dmr_addr = (split('/', $dmr_prefix))[0]) =~ s/://g;
57
58    return packdhcp($option_s46_dmr,
59      sprintf("%02x", $dmr->prefixlen()) . $dmr_addr);
60 }
61
62 my $dhcp_option = create_bmr();
63 $dhcp_option .= create_dmr();
64 $dhcp_option =~ s/..\K(?=.)/:/g;
65
66 print "option dhcp6.mapt-option $dhcp_option;\n";
```

Listing A.3: Make DHCPv6 BMR and DMR options for a MAP-T domain.

```
1  #!/bin/bash
2
3  mkdir graphs
4
5  # take all Flent test files in the current directory
6  ls -1 *.gz | while read file; do
7
8    # gather name of test
9    testname="`echo $file|perl -wple 's,^(.+?)\-2016.+,$1,'`"
10
11   # iterate through all the plots available for that test
12   flent $testname --list-plots 2>&1 | grep -v "^Available" | while read
         plot; do
13
14     # gather name of plot
15     plotname="`echo $plot|perl -ple 's,\s+,,g;s,^(.+?)\:.+,$1,'`";
16
17     # graph name
18     graphname="graphs/$testname-$plotname.png"
19
20     flent $testname -p $plotname -i $file -o $graphname
21   done
22 done
```

Listing A.4: Simple script to make all the available graphs of each plot type in Flent.

```
1 !
2 ip cef
3 ipv6 unicast-routing
4 ipv6 cef
```

```
!
interface GigabitEthernet0/0
 description lnk:dalek; eth6; internet
 ip address 185.54.31.1 255.255.255.254
 load-interval 30
 media-type gbic
 speed auto
 duplex auto
 no negotiation auto
 ipv6 address 2A02:4260:F000::1/127
 ipv6 enable
!
interface GigabitEthernet0/1
 description lnk:cybermat; Gi1/0/9; servers
 ip address 185.54.31.6 255.255.255.254
 load-interval 30
 media-type rj45
 speed auto
 duplex auto
 no negotiation auto
 ipv6 address 2A02:4260:F000:1::1/64
 ipv6 enable
 ipv6 nd prefix default no-advertise
!
interface GigabitEthernet0/2
 description lnk:cybermat; Gi1/0/11; Outside --> csr1000v; Gi3
 ip address 185.54.31.2 255.255.255.254
 ip ospf authentication null
 ip ospf 503 area 0
 load-interval 30
 media-type rj45
 speed auto
 duplex auto
 no negotiation auto
 ipv6 address 2A02:4260:F000::2/127
 ipv6 enable
 ipv6 ospf 503 area 0
!
router ospf 503
 router-id 185.54.31.2
 log-adjacency-changes
 auto-cost reference-bandwidth 10000
 redistribute connected subnets
 redistribute static subnets
 passive-interface default
```

```
 no passive-interface GigabitEthernet0/2
 default-information originate always
!
ip route 0.0.0.0 0.0.0.0 GigabitEthernet0/0 185.54.31.0
ipv6 route ::/0 GigabitEthernet0/0 2A02:4260:F000::
!
ipv6 router ospf 503
 router-id 185.54.31.2
 log-adjacency-changes
 auto-cost reference-bandwidth 10000
 default-information originate always
 passive-interface default
 no passive-interface GigabitEthernet0/2
 redistribute connected
 redistribute static
!
```

Listing A.5: Core router configuration

```
!
ip routing
ipv6 unicast-routing
!
ip dhcp excluded-address 185.54.31.8 185.54.31.10
ip dhcp pool openwrt
   network 185.54.31.8 255.255.255.248
   domain-name foobar.jocke.no
   default-router 185.54.31.9
   dns-server 8.8.8.8
!
interface GigabitEthernet0/1
 description lnk:openwrt
 no switchport
 ip address 185.54.31.9 255.255.255.248
 ip ospf authentication null
 ip ospf 503 area 0
 load-interval 30
 ipv6 address 2A02:4260:F000:2::1/64
 ipv6 enable
 ipv6 ospf 503 area 0
!
interface GigabitEthernet0/12
 description lnk:csr1000v; Gi4; Inside
 no switchport
 ip address 185.54.31.4 255.255.255.254
```

```
 ip ospf authentication null
 ip ospf 503 area 0
 load-interval 30
 ipv6 address 2A02:4260:F000::4/127
 ipv6 enable
 ipv6 ospf 503 area 0
!
router ospf 503
 router-id 185.54.31.4
 auto-cost reference-bandwidth 10000
 redistribute connected
 redistribute static
 passive-interface default
 no passive-interface GigabitEthernet0/12
!
ipv6 route 2A02:4260:F000:3::/64 GigabitEthernet0/1 2A02:4260:F000:2::2
ipv6 router ospf 503
 router-id 185.54.31.4
 auto-cost reference-bandwidth 10000
 passive-interface default
 no passive-interface GigabitEthernet0/12
 redistribute connected
 redistribute static
!
```

Listing A.6: Dual-stack: distribution switch configuration

```
!
ipv6 unicast-routing
!
interface GigabitEthernet1
 description net1
 load-interval 30
 negotiation auto
!
interface GigabitEthernet2
 description net0
 load-interval 30
 negotiation auto
!
interface GigabitEthernet3
 description net2; Outside; VLAN502
 ip address 185.54.31.3 255.255.255.254
 ip ospf authentication null
 ip ospf 503 area 0
```

```
 load-interval 30
 negotiation auto
 ipv6 address 2A02:4260:F000::3/127
 ipv6 enable
 ipv6 ospf authentication null
 ipv6 ospf 503 area 0
!
interface GigabitEthernet4
 description net3; Inside; VLAN503
 ip address 185.54.31.5 255.255.255.254
 ip ospf authentication null
 ip ospf 503 area 0
 load-interval 30
 negotiation auto
 ipv6 address 2A02:4260:F000::5/127
 ipv6 enable
 ipv6 ospf authentication null
 ipv6 ospf 503 area 0
!
router ospf 503
 router-id 185.54.31.5
 auto-cost reference-bandwidth 10000
 redistribute connected subnets
 redistribute static subnets
 passive-interface default
 no passive-interface GigabitEthernet3
 no passive-interface GigabitEthernet4
!
ipv6 router ospf 503
 router-id 185.54.31.5
 auto-cost reference-bandwidth 10000
 passive-interface default
 no passive-interface GigabitEthernet3
 no passive-interface GigabitEthernet4
 redistribute connected
 redistribute static
!
```

Listing A.7: Dual-stack: BR router configuration

```
config interface 'loopback'
  option ifname 'lo'
  option proto 'static'
  option ipaddr '127.0.0.1'
  option netmask '255.0.0.0'
```

```
6
7  config interface 'lan'
8    option type 'bridge'
9    option ifname 'eth1'
10   option proto 'static'
11   option ipaddr '10.13.37.1'
12   option netmask '255.255.255.0'
13   option ip6addr '2a02:4260:f000:3::1/64'
14   option mtu '1476'
15
16 config interface 'wan'
17   option ifname 'eth0'
18   option proto 'static'
19   option ipaddr '185.54.31.10'
20   option netmask '255.255.255.248'
21   option gateway '185.54.31.9'
22   option mtu '1476'
23
24 config interface 'wan6'
25   option ifname 'eth0'
26   option proto 'static'
27   option ip6addr '2A02:4260:F000:2::2/64'
28   option ip6gw '2A02:4260:F000:2::1'
29   option mtu '1476'
30
31 config switch
32   option name 'switch0'
33   option reset '1'
34   option enable_vlan '1'
35
36 config switch_vlan
37   option device 'switch0'
38   option vlan '1'
39   option ports '2 3 4 5 0'
40
41 config switch_vlan
42   option device 'switch0'
43   option vlan '2'
44   option ports '1 6'
```

Listing A.8: Dual-stack: CPE router configuration, /etc/config/network

```
1 !
2 ip routing
3 ipv6 unicast-routing
```

```
!
interface GigabitEthernet0/1
 description lnk:openwrt
 no switchport
 no ip address
 load-interval 30
 ipv6 address 2A02:4260:F000:2::1/64
 ipv6 enable
 ipv6 nd managed-config-flag
 ipv6 nd other-config-flag
 ipv6 dhcp relay destination 2A02:4260:F000:1::2
 ipv6 ospf 503 area 0
!
interface GigabitEthernet0/12
 description lnk:csr1000v; Gi4; Inside
 no switchport
 no ip address
 load-interval 30
 ipv6 address 2A02:4260:F000::4/127
 ipv6 enable
 ipv6 ospf 503 area 0
!
ipv6 router ospf 503
 router-id 185.54.31.4
 auto-cost reference-bandwidth 10000
 passive-interface default
 no passive-interface GigabitEthernet0/12
 redistribute connected
 redistribute static
!
```

Listing A.9: MAP-T: distribution switch configuration

```
!
ipv6 unicast-routing
!
interface GigabitEthernet1
 description net1
 load-interval 30
 negotiation auto
!
interface GigabitEthernet2
 description net0
 load-interval 30
 negotiation auto
```

```
!
interface GigabitEthernet3
 description net2; Outside; VLAN502
 ip address 185.54.31.3 255.255.255.254
 ip ospf authentication null
 ip ospf 503 area 0
 load-interval 30
 negotiation auto
 nat64 enable
 ipv6 address 2A02:4260:F000::3/127
 ipv6 enable
 ipv6 ospf authentication null
 ipv6 ospf 503 area 0
!
interface GigabitEthernet4
 description net3; Inside; VLAN503
 no ip address
 load-interval 30
 negotiation auto
 nat64 enable
 ipv6 address 2A02:4260:F000::5/127
 ipv6 enable
 ipv6 ospf authentication null
 ipv6 ospf 503 area 0
!
router ospf 503
 router-id 185.54.31.5
 auto-cost reference-bandwidth 10000
 redistribute connected subnets
 redistribute static subnets
 passive-interface default
 no passive-interface GigabitEthernet3
!
ipv6 router ospf 503
 router-id 185.54.31.5
 auto-cost reference-bandwidth 10000
 passive-interface default
 no passive-interface GigabitEthernet3
 no passive-interface GigabitEthernet4
 redistribute connected
 redistribute static
!
nat64 map-t domain 1
 default-mapping-rule 2A02:4260:F000:4::/64
 basic-mapping-rule
```

```
58   ipv6-prefix 2A02:4260:F800::/48
59   ipv4-prefix 185.54.31.128/28
60   port-parameters share-ratio 16 start-port 1024
61 !
```

Listing A.10: MAP-T: BR router configuration

```
1  config interface 'loopback'
2    option ifname 'lo'
3    option proto 'static'
4    option ipaddr '127.0.0.1'
5    option netmask '255.0.0.0'
6
7  config interface 'lan'
8    option type 'bridge'
9    option ifname 'eth1'
10   option proto 'static'
11   option ipaddr '10.13.37.1'
12   option netmask '255.255.255.0'
13   option ip6assign '64'
14   option mtu '1476'
15
16 config interface 'wan6'
17   option ifname 'eth0'
18   option proto 'dhcpv6'
19   option reqaddress 'try'
20   option reqprefix 'auto'
21   option mtu '1476'
22
23 config switch
24   option name 'switch0'
25   option reset '1'
26   option enable_vlan '1'
27
28 config switch_vlan
29   option device 'switch0'
30   option vlan '1'
31   option ports '2 3 4 5 0'
32
33 config switch_vlan
34   option device 'switch0'
35   option vlan '2'
36   option ports '1 6'
```

Listing A.11: MAP-T: CPE router configuration, /etc/config/network

```
subnet6 2a02:4260:f000:1::/64 {
}

option dhcp6.mapt-option code 95 = string;

subnet6 2a02:4260:f000:2::/64 {
  range6 2a02:4260:f000:2::100 2a02:4260:f000:2::1000;

  prefix6 2a02:4260:f800:: 2a02:4260:f800:ff00:: /56;

  option dhcp6.mapt-option 00:59:00:16:00:08:1c:b9:36:1f:80:30:2a
      :02:42:60:f8:00:00:5d:00:04:06:00:00:00:00:5b:00:09:40:2a:02:42:60:
      f0:00:00:04;
}
```

Listing A.12: map-server: DHCPv6 configuration, /etc/dhcp/dhcpd6.conf