A close examination of the public Teredo infrastructure

and its application in connectivity monitoring

Maarten Aertsen
maarten@rtsn.nl 4096R/14789500

March 9, 2012

Bachelor of Science thesis in Telematics
Committee: Giovane Moura, Aiko Pras, Pieter-Tjerk de Boer
Chair: Design and Analysis of Communication Systems (DACS)
Faculty: Electrical Engineering, Mathematics and Computer Science (EEMCS)
Institution: University of Twente, The Netherlands

Contents

1 Introduction **5**

1.1 Problem description 5

1.2 Research questions 6

1.3 Approach and document outline 6

2 Background **8**

2.1 Teredo 8

2.1.1 Purpose 8

2.1.2 Infrastructure entities 8

2.1.3 A sample connection 10

2.2 Connectivity monitoring 11

2.2.1 Why use connectivity monitoring as an example 12

2.2.2 Purpose 12

2.2.3 Connectivity monitor blueprint 13

3 Infrastructure survey **15**

3.1 Selection of candidate capabilities 15

3.1.1 Candidate capabilities of the client 15

3.1.2 Candidate capabilities of the server 16

3.1.3 Candidate capabilities of the relay 16

3.1.4 Summary 17

3.2 Infrastructure availability 17

3.2.1 Relay enumeration 17

3.2.2 Server enumeration 21

3.2.3 Summary 23

3.3 Conclusion 23

4 Proposed usage of the infrastructure **24**

4.1 Candidate suitability for connectivity monitoring 24

4.1.1 Requirements for connectivity monitoring 24

4.1.2 Examining candidate suitability for connectivity monitoring . . . 25
4.1.3 Summary . . . 26
4.2 Design of a connectivity monitor . . . 26
4.2.1 Connectivity monitoring using Teredo infrastructure . . . 27
4.2.2 Connectivity monitoring using Teredo servers . . . 27
4.2.3 Connectivity monitoring using Teredo relays . . . 29
4.2.4 Summary . . . 30
4.3 Conclusion . . . 31

5 Guidelines for validation **32**
5.1 Validation overview . . . 32
5.1.1 Measurements required for validation . . . 32
5.1.2 Measurement outcomes and respective meaning . . . 33
5.2 Guidelines for implementation of the monitor . . . 34
5.2.1 Implementation guidelines for server use . . . 34
5.2.2 Implementation guidelines for relay use . . . 35
5.3 Guidelines for lab-scale validation . . . 35
5.3.1 A sample lab scale network topology . . . 35
5.3.2 Entities involved in the sample lab scale network topology . . . 36
5.4 Guidelines for Internet-scale validation . . . 37
5.5 Conclusion . . . 37

6 Conclusions **39**
6.1 Future work . . . 39
6.2 Conclusion . . . 39
6.2.1 Recommendations . . . 40

A Raw results relay enumeration **43**
A.1 early 2010 . . . 43
A.2 early 2012 . . . 43

B Raw results server enumeration **45**
B.1 Collected from software analysis . . . 45

B.2 Collected from server logs 45
B.3 Collected from local client discovery 45

Abstract

This research focusses on the public infrastructure used by the Teredo protocol, a tunneling technology providing IPv6 connectivity to hosts on IPv4-only networks. The purpose is to determine whether the protocol is robust enough to prevent any unintended repurposing of its infrastructure. To judge the practicality of our findings, any room for misuse found is used to collect information on the connectivity of networks, answering the research question: *"Can the public Teredo infrastructure be used to monitor IP connectivity of an autonomous system?"*.

To answer this research question, we present: the concept of a connectivity monitor using the public Teredo infrastructure; the results of a close examination of the protocol specification to find capabilities potentially useful in the realisation of this monitor; methodologies to locate all public Teredo infrastructure on the Internet and the results of three surveys using these methodologies; the design of a connectivity monitor using the capabilities found; and guidelines to perform a full validation of this connectivity monitor.

Based on our research of the Teredo protocol specification, we believe that it is possible to use the public Teredo infrastructure to monitor IP connectivity of autonomous systems and perform other functions outside the intended scope of Teredo. Judging by the results of our survey, finding a large deployment base of public Teredo infrastructure, we believe that real world validation of this work to allow mitigation of potential room for misuse is highly advisable.

Chapter 1

Introduction

We live in an important year for the Internet. If someone were to start collecting a list of milestones of over forty years of Internet history, February 3 2011 would most definitely deserve an entry. It illustrates the popularity and growth of a worldwide network. A network originally envisioned to connect the available *interactive computers*[10]. In the course of this growth from research project to worldwide infrastructure, the Internet exhausted the resources required to expand further. February 3rd, 2011 was the day when the IANA[1] Unallocated Address Pool ran out of IP blocks to distribute[2].

Fortunately, the issue of address depletion was foreseen long time ago. For nearly twenty years the Internet community has worked on solutions, both by conserving address space and development of a successor to the Internet Protocol version 4 (IPv4) to create more headroom[5]. However, finding ways to slow down address exhaustion and inventing new protocols is only the first step in solving an operational problem. After invention comes implementation, transition and finally full adoption.

Transition from IPv4 to IPv6 has proven to be painful: although the base IPv6 standard was published in December 1998[3] and subsequently implemented by vendors, a study in april 2011 estimated IPv6 use still between 0.1 and 0.2 percent of the total traffic volume[9]. The biggest technical cause for this delay is the lack for (full) IPv6 support. The transition to dual connectivity over both IPv4 and IPv6 is not just the toggle of a switch labeled *enable IPv6*. Networks need to individually upgrade applications, hosts and network components and train staff. Adding to this intra-network complexity are the different transition periods of interconnected networks. Including market factors, the transition to a fully IPv6 capable world will take decades.

In the meantime, hosts using IPv4 and IPv6 will need to talk to each other. In absence of support from an upstream ISP, there are multiple existing solutions to accomplish this. Most common are: explicit tunneling using a tunnel broker[4], 6to4[2, 6] and isatap[14]. Less common, but deployed on all computers running Microsoft Windows starting from the Vista version, is Teredo[7]. These transition technologies use additional infrastructure to translate between the IPv4 and IPv6 world. Translation between two worlds which will be increasingly needed now February 3rd, 2011 is past us.

1.1 Problem description

Infrastructure for transition mechanisms is becoming ever more critical as the need for translation between IPv4 and IPv6 increases. Proper scrutiny of these solutions and their infrastructure is required to prevent misuse. This research focusses on the public infrastructure used by the Teredo protocol. The purpose is to determine whether the protocol is robust enough to prevent any unintended repurposing of its infrastructure.

To judge the practicality of our findings, any room for misuse found will be used to collect information on the connectivity of networks. In the real world, this connectivity information, when available from topographically dispersed points in the Internet, can be used to monitor connectivity of a chosen network and can provide information on its connectivity arrangements with other networks. This information is often considered sensitive.

[1] The Internet Assigned Numbers Authority (IANA) is responsible for the global coordination of the DNS Root, IP addressing, and other Internet protocol resources.

[2] IANA distributes large blocks of IP addresses (/8's) over the world's Regional Internet Registries (RIRs), which distribute them further down to regional ISPs. Originally, each IP address identified one unique node on the Internet.

1.2 Research questions

To get a clearly defined research question we specify that the networks we will be looking at are Autonomous Systems (AS), the smallest building blocks of the Internet. The main question then becomes:

> *Can the public Teredo infrastructure be used to monitor IP connectivity of an autonomous system?*

We view *IP connectivity* as an operational status of an IP network, comprising of the presence or absence two key components: network reachability[3] and host reachability[4]. In that perspective, IP connectivity monitoring is measuring change in this operational status.

Due to the low volume of research in the area of Teredo, it is required to do some groundwork before we are able to address the main question. These steps are split up as sub questions to the main research question.

1. What Teredo infrastructure is publicly available?
   (a) What capabilities does the infrastructure provide according to the specifications?
   (b) Are there any unintended capabilities introduced in the specifications?
   (c) How much infrastructure is publicly available?
2. Is Teredo infrastructure suitable for connectivity monitoring?
   (a) Is any of the specified behaviour suitable for connectivity monitoring?
   (b) Is any of the specified but unintended behaviour suitable for connectivity monitoring?
   (c) Do any of the real-world implementations introduce unspecified capabilities suitable for connectivity monitoring?

1.3 Approach and document outline

Following this introduction chapter, this report is built up as follows. Figure 1.1 shows a graphical summary of this section.

Chapter 2 provides a background in the Teredo protocol to introduce the reader to the subject matter. In addition, this chapter will explain the connectivity monitoring concept which will be used later to judge the practical value of any results.

Chapter 3 contains the methodology and results of a survey of the publicly available Teredo infrastructure. The first part of this chapter consists of a close examination of the specification to list all capabilities, both intended and unintended. The second part of the chapter is dedicated to the actual survey. Due to the limited availability of data on this subject we make an attempt to locate all public Teredo infrastructure on the Internet. This chapter covers sub question 1.

[3] i.e. operational status of routers, connectivity status with peer or upstream networks and announcements of address space in a routing protocol.

[4] i.e. host uptime, interface status, access policies.

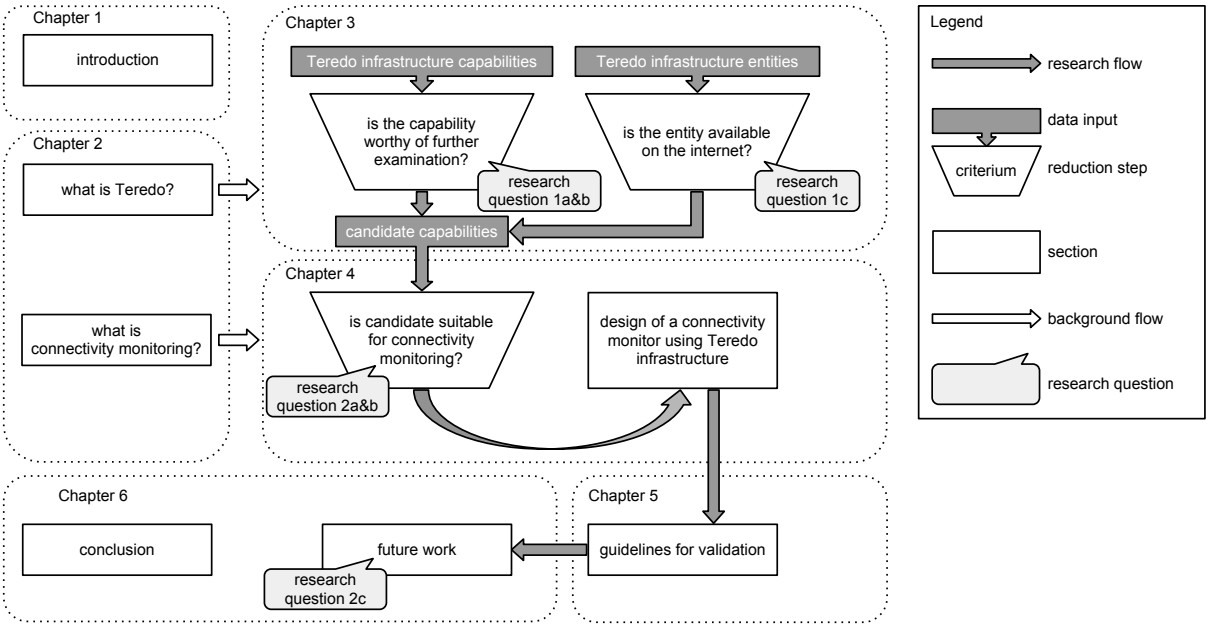


Figure 1.1: Graphical representation of the document outline

Chapter 4 continues with the set of capabilities found in chapter 3 and evaluates their usefulness using the connectivity monitor concept. Additional examination of production implementations of Teredo infrastructure to find more unintended capabilities is not attempted. The design of a prototype monitor is discussed using these capabilities. This addresses parts (a) and (b) of sub question 2, (c) is left as future work.

Chapter 5 combines the findings of Chapters 3 and 4 and provides guidelines for validation of our findings, by testing a prototype connectivity monitor on real world production implementations of the Teredo infrastructure. An actual validation is not included in this report due to time constraints. It is our hope that the guidelines provided will lead others to resume this direction of research.

Chapter 6 concludes this report and provides recommendations for future work.

Chapter 2

Background

This chapter provides the background required to read Chapters 3 and 4, following the flow depicted in Figure 1.1. First the reader is introduced to Teredo: we start with an overview, followed by a description of the different entities and an in depth examination of a connection setup. This knowledge is used in Chapter 3 to select infrastructure capabilities. In Chapter 4, the connection monitor concept introduced in the second part of this chapter, combined with the conclusions of Chapter 3, is used to design a connectivity monitor using Teredo infrastructure. Before introducing the concept, the motivation for choosing connectivity monitoring and its purposes are explained.

The reader is expected to have prior knowledge of basic networking topics (i.e. TCP/IP, IPv6, NAT, Internet routing).

2.1 Teredo

This section provides a background on Teredo. Having finished this section, the reader is expected to have a basic understanding of the Teredo protocol entities and their functioning. While explaining the protocol, quotes from its specification (RFC 4380 [7]) are used, which are printed in *italic*.

2.1.1 Purpose

> Teredo *enables nodes located behind one or more IPv4 Network Address Translations (NATs) to obtain IPv6 connectivity by tunneling packets over UDP*

Teredo is a protocol used in the transition period from IPv4 to IPv6. Its stated design goal is to *provide an "IPv6 provider of last resort"*. In other words: provide IPv6 connectivity to hosts which do not have access to any other form of IPv6 connectivity (e.g. static tunneling, 6to4 or native). This is usually caused by the presence of one or more devices which do not allow or do not support the traversal of packets using IP protocol 41 (6in4, IPv6 directly encapsulated in IPv4 [12]) in the network path. The most common cause of this problem is NAT, omnipresent in Customer Premises Equipment (CPE, e.g. wireless routers). Teredo was designed to overcome this problem by using UDP instead of raw IPv4 to tunnel IPv6 packets.

2.1.2 Infrastructure entities

To provide the Teredo service, a total of three entities are involved. In the following paragraphs respectively the client, server and relay are introduced. Together they enable a node without normal IPv6 connectivity to obtain IPv6 connectivity using Teredo. An actual connection setup will be illustrated in the next subsection.

Client

The Teredo client is *a node that has some access to the IPv4 Internet and wants to gain access to the IPv6 Internet*. Teredo is only supposed to be used when there is no better alternative to obtain connectivity. In the general case this means that the client is behind a NAT and uses RFC 1918 private address space, causing preferred transition mechanisms to fail.

In Teredo, each client is addressed within `2001:0::/32`, the *Global Teredo IPv6 Service Prefix* (hereafter: "Teredo prefix"). Encoded in this client address is the global IPv4 address of the client[1] and the UDP port used by the Teredo client software. In the next paragraph we show that additional information about the server is also encoded in the address.

One of the main differences between Teredo and other transition technologies is its use of UDP for encapsulation. Although the use of UDP increases the amount of CPE that knows how to deal with the packets it receives, it does not solve the problem NAT causes for the *end to end model* of networking: it is still not possible to directly address a host behind NAT. In practice this prevents hosts on the WAN-side of the NAT to set up connections with hosts behind it. This is the reason why Teredo, in addition its use of UDP, needs to provide a way for packets to "punch through" the NAT (in a STUN-like manner [13]). This is done by the set up and maintenance of mappings, which brings us to the server.

Server

The server is *a node that has access to the IPv4 Internet through a globally routable address, and is used as a helper to provide IPv6 connectivity to Teredo clients*. The most important role of the server is to deal with the NAT blocking the direct connectivity of the client by helping in the setup and maintenance of mappings. To set up these mappings, so called bubble packets are used, which are sent to and echoed by the client. This causes the NAT to learn that packets from that destination back to the client should be forwarded. This is called a mapping in the NAT.

The server is also responsible for the addressing of the client, by providing tunneled router advertisements in response to router solicitations from clients[2]. Supplementary to the information describing the client, each address handed out contains the server IPv4 address. This is used by relays to identify the server of a given client.

Although the server plays a central role in the initialization of the client, no actual data is carried. The server only transports bubbles and certain types of ICMP traffic. Transporting traffic from and to clients is handled by the relay.

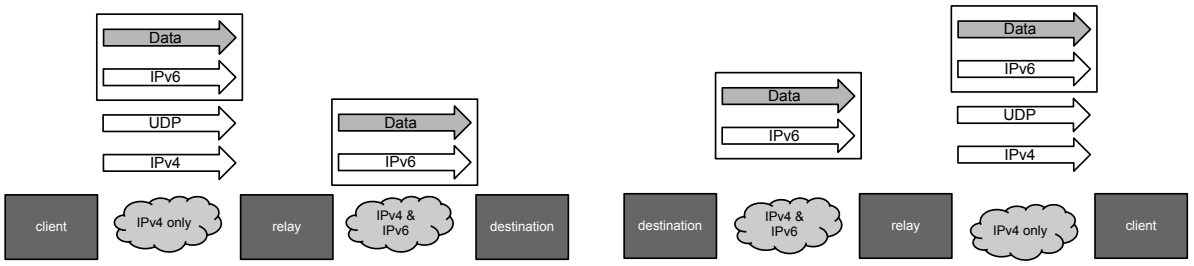
Relay

The relay is *an IPv6 router that can receive traffic destined to Teredo clients and forward it using the Teredo service*. To this end, the relay first contacts a client's server to set up mappings in NAT and then directly forwards traffic to the client. Traffic for clients transparently ends up at relays because the Teredo prefix is layer 3 anycasted by networks containing relays. The actual data from and to the client is encapsulated in UDP, as illustrated in Figure 2.1. IPv6 packets sent by the client first traverse an IPv4-only network encapsulated in UDP. The relay (acting as a protocol translator) removes this IPv4+UDP layer, releasing the encapsulated IPv6 packet onto the IPv6 Internet.

To minimize suboptimal routing caused by the tunneled path from client to relay, a client chooses a relay on a per destination basis. The next subsection explains this relay selection and parts of the setup

[1] More specifically the global IPv4 address of the WAN side of the NAT in front of the client.

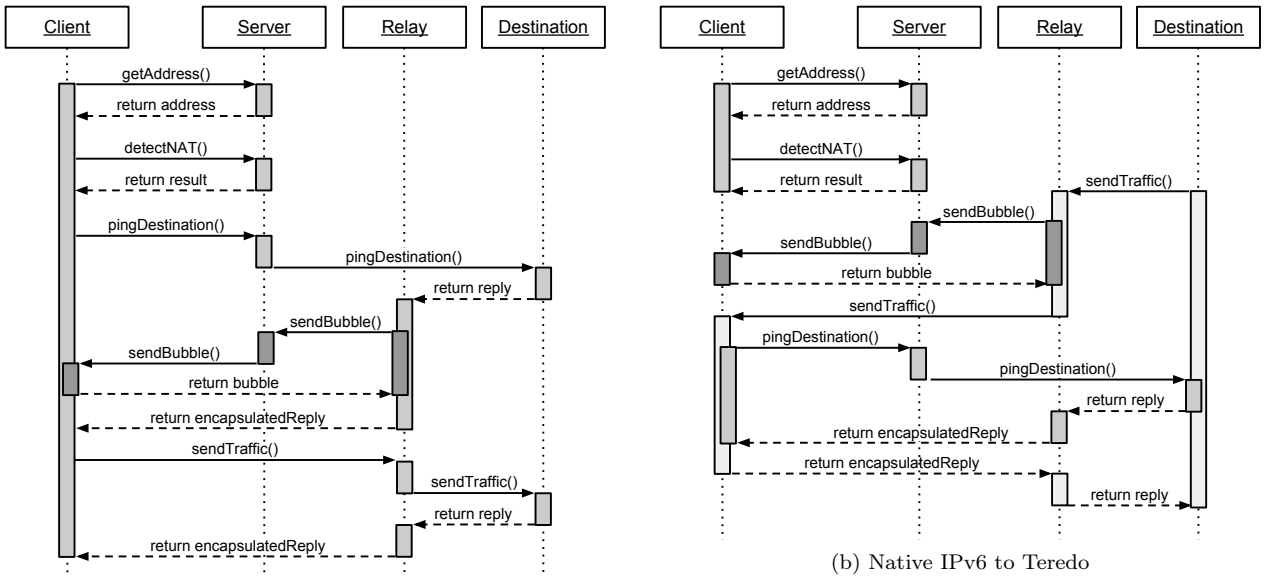
[2] The router advertisements sent by the server during the initialisation phase (called *qualification*) are mostly identical to the announcements specified in RFC 4861 [11] for native IPv6, with the notable exception that all packets are encapsulated in UDP. A secure alternative is available



(a) client to destination

(b) destination to client

Figure 2.1: The Teredo relay as UDP tunnel endpoint



(a) Teredo to native IPv6

(b) Native IPv6 to Teredo

Figure 2.2: sample Teredo connections to and from native IPv6 hosts

required by the client and server.

2.1.3 A sample connection

This section explains the connection setup required for traffic exchange between a Teredo client and a native IPv6 speaker. First a connection from a Teredo client to a native IPv6 destination is explained, then we show the reverse. Both connections are illustrated in Figure 2.2. Note that these illustrations have been simplified[3] and use call names which are not found in the specification, both done to increase readability. In the text, we will refer to the call names used in the figure.

Common to the two connection directions is the initial setup sequence by the Teredo client. It first asks for an address by sending router solicitations to the server (`getAddress()`). The server replies with a router advertisement, having determined the external IPv4 address and port after NAT by looking at the source address and port of the UDP packet containing the router solicitation. Next is a procedure to detect the NAT type (`detectNAT()`). We will not explain this part of the set up any further, both because the NAT type is not relevant for our purposes and because it has been deprecated in later updates of the protocol, where a particular type of NAT is always assumed[16].

[3]We exclude the full packet exchange in the initialisation phase (`getAddress()` and `detectNAT()`). Full details can be found in the specification.

Teredo to native IPv6

When a Teredo client wants to connect to an IPv6 destination, having first completed the initialisation sequence, its first priority is to locate a Teredo relay, as close as possible to its destination. To do this, it sends an ICMPv6 echo packet with its own Teredo address as the source to the destination. This IPv6 packet is sent over UDP to its server (`pingDestination()` in Figure 2.2, using the UDP encapsulation show in Figure 2.1). The server decapsulates this packet and sends it over native IPv6 to the destination (2nd `pingDestination()`). The destination (or if it is unreachable, the first upstream router) replies (`return reply`). This reply-packet, addressed to the Teredo IPv6 address of the client is then forwarded to nearest relay (due to the layer 3 anycasting of the Teredo prefix).

The relay, discovering a new *peer* by its destination address, first verifies whether the IPv4 host (our Teredo client) encapsulated in the destination IPv6 address really runs a client[4] and expects packets from the source address. This verification is done using a bubble packet, sent to the server (also determined from the Teredo address) encapsulated in UDP, with the inner packet addressed to the client. The server relays this tunneled packet to the client. When the relay receives a bubble *directly* from the client, all queued packets are directly sent over UDP. This bubble exchange not only serves as DoS protection, but also to open mappings in NAT enabling direct communication between relay and client.

The client, having discovered the relay closest to its destination, can now directly send traffic over UDP to the relay (`sendTraffic()`). Finding a relay and verifying your path to it is known as a *direct connectivity test*.

Native IPv6 to Teredo

A packet exchange initiated by a native IPv6 host looks very much like the reverse just described, albeit there are some differences in ordering. Packets sent towards the client (`sendTraffic()`) end up at the nearest relay, which encounters a new *peer* as their destination. This triggers the bubble exchange described previously.

Before the client can reply to the first packet, it needs to find a relay to reach to its destination. To do this, it will send an ICMPv6 echo packet to the destination as previously explained, with the exception that the server already knows the client on the return path. Having found the nearest relay, the Teredo client dequeues all traffic for the native IPv6 host.

Unsolicited connections from a host using native IPv6 to a Teredo client is not guaranteed to work. The specification warns clients that the traffic received from unknown IPv6 addresses may be spoofed and they are not required to accept that traffic. During our research we did not come across clients that drop unsolicited traffic from native IPv6 hosts.

This concludes our introduction of Teredo. Chapter three uses the background on Teredo covered here to answer research question 1.

2.2 Connectivity monitoring

In Section 1.2 we defined the term "IP connectivity monitoring" used in the main research question. This section will introduce the concept we will later refer to as the connectivity monitor. This concept is used to show that any room for alternative, non-intended use of the Teredo infrastructure found is not only theoretical. It serves as an example to illustrate such alternative use to a wider population than the academic community.

[4]The algorithm to convert from IPv4 address to Teredo address is very simple, so anyone can start sending packets to such an address. Without this detection Teredo relays could easily be used for DoS purposes

First we explain why connectivity monitoring was chosen as an example. Then we state the purpose of a connectivity monitor using Teredo infrastructure to collect its measurements. Finally we provide a blueprint of the actual monitor.

2.2.1 Why use connectivity monitoring as an example

This subsection explains why connectivity monitoring was chosen as an example for alternative use of the Teredo infrastructure. The choice of an example was based on four criteria, which we list below. For each of these, we explain why the chosen example satisfies the criterion. In the next section we continue with the purpose of connectivity monitoring.

Simple The example chosen should be simple to implement. The focus of this research is Teredo, not some alternative service. When alternative capabilities are demonstrated to exist, one can move from example to real-world implementation.

Connectivity monitoring can be as simple as determining whether a reply is received for each request sent. This is trivial to implement and requires minimal state. Extensions to measure loss, latency, path etc. are also trivial to implement.

Useful The example should demonstrate a capability that has use beyond the surprise of its mere existence. Ideally, it should provide a function or service which is an addition to the tools available to the Internet community, which increases its appeal to a wider population than the academic community.

Connectivity between networks on the Internet is essential for services to function beyond local networks. New tools to debug this inter network connectivity are always a welcome addition when they can provide additional depth. Using a connectivity monitor as an example to illustrate Teredo infrastructure capabilities serves to measure the network connectivity as experienced by this infrastructure, which currently falls outside the reach of the general operator community.

Universally understandable The example service should be well-known: network operators should be able to relate to the example without additional background reading.

Standard connectivity tools like ping and traceroute are universally known among Internet users and network operators. Implementing a ping or traceroute like service using Teredo infrastructure satisfies this criterion.

Non-obvious An example should use the Teredo infrastructure to do something outside the protocol. In other words, the effect of the illustrated capability should have a scope larger than the infrastructure itself. This is property is required to make the example appeal to a wider community than the operators of Teredo infrastructure.

Connectivity monitoring beyond the in-protocol reachability of clients and relays is beyond the scope of the Teredo protocol. Using the protocol or its infrastructure for unintended purposes such as general connectivity monitoring makes the example non-obvious.

2.2.2 Purpose

This section intends to clarify the purpose of connectivity monitoring using Teredo infrastructure. As shortly alluded to in the previous section, adding additional depth to the existing toolbox of connectivity

debugging tools can be valuable for network operators. IPv6 migration strategies have been known to cause (unexpected) breakage, which further substantiates this claim [1, 8]. Combining the criteria from the previous section with our intention to provide a new tool establishes our purpose, which for clarity is listed in full below:

> *By illustrating possible Teredo infrastructure capabilities using a connectivity monitor as an example service, we provide:*
>
> *a) the reader a simple, useful, universally understandable and non-obvious example to illustrate any discovered capability, and*
>
> *b) the network operator an additional tool to debug connectivity problems associated with Teredo.*

2.2.3 Connectivity monitor blueprint

With both background and purpose clear this section specifies a blueprint of the connectivity monitor. In the remainder of this research, an attempt will be made to fill in parts of this blueprint using Teredo capabilities.

We start by describing the general concept in three sentences:

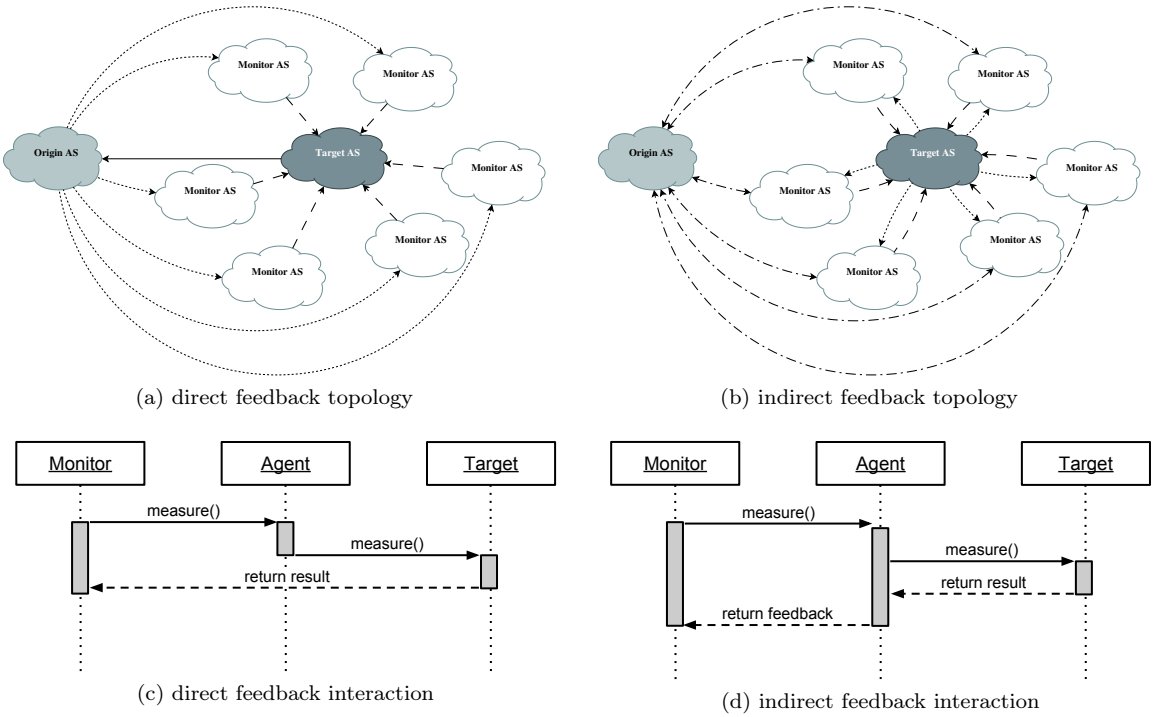
> *from* `<monitor AS>`;
>
> *using* Teredo infrastructure at {`<agent AS`$_1$`>`, `<agent AS`$_2$`>`, ..., `<agent AS`$_n$`>`};
>
> *monitor* `<target AS>`.

In the following paragraphs we will step by step analyse these three sentences.

collecting measurements We discern two possible ways of interaction between `<monitor AS>` and `<target AS>`, illustrated in 2.3. Using the first method, hereafter called *indirect feedback*, `<monitor AS>` requests information from each `<agent AS>`. In turn, each of these `<agent AS>`'s performs a measurement and subsequently returns information. Using the second method, the first part is the same, but the `<target AS>`'s response goes directly back to `<monitor AS>`. We call this method *direct feedback*, as the feedback does not pass back via the `<agent AS>`. Although there is a significant difference in result, we do not discern between direct and indirect feedback before examining the infrastructure capabilities.

using Teredo infrastructure Measurements conducted using the model in Figure 2.3 will yield information on connectivity as experienced by `<agent AS>`. At this point, we cannot specify sentence two in more detail, as the capabilities of Teredo infrastructure are still unknown. That will change as we progress through the next chapters: Chapter 3 will provide pointers to available capabilities, which will be examined for use in the context of sentence two in Chapter 4. Any results of this examination will be used to design a prototype for this sentence, thereby completing the description of our connectivity monitor.

actual monitoring This leaves the actual monitor step unspecified. We intend to collect information on two things: a) the connectivity status of `<target AS>` and b) properties of this connectivity. To be more specific, we want to replicate the output of the ping and traceroute tools. We will use ping to verify connectivity and traceroute to examine the path/hop count (as an example of a property). By providing host level monitoring capabilities and extrapolating multiple measurements, it becomes possible to measure IP connectivity of an AS.



(a) direct feedback topology

(b) indirect feedback topology

(c) direct feedback interaction

(d) indirect feedback interaction

Figure 2.3: Diagrams of connectivity monitor concepts

For completeness we specify what we see as the minimal ping and traceroute implementations. When referring to ping, we allude to the measurement of ICMP(v6) reply packets in response to an ICMP(v6) echo packets. When referring to traceroute, we mean measurement of ICMP(v6) TTL exceeded and/or reply packets, in response to to ICMP(v6) echo packets with increasing TTL values. In Chapter 4 we will return to the blueprint of the connectivity monitor and fill in the blanks.

Chapter 3

Infrastructure survey

As seen in Figure 1.1 (document outline), this chapter works towards the use of Teredo for connectivity monitoring by creating a list of capabilities which are potentially useful in our connectivity monitor. To select a list of these "candidates", we reduce the full list of capabilities as provided in the Teredo specification in two distinct ways.

The first reduction, which is the subject of Section 3.1, uses the specification to only select capabilities of a certain type. The second reduction, which is the subject of Section 3.2, uses the availability information of Teredo entities on the Internet to only select capabilities of actively deployed protocol entities.

This twofold survey of the Teredo infrastructure answers research question 1 and provides a foundation for chapter 4, where each candidate found in this chapter is further examined for its suitability in our connectivity monitor.

3.1 Selection of candidate capabilities

This section reduces the full list of Teredo capabilities as found in the Teredo specification to a list of candidates. We first define *capability* and *candidate capability*:

definition 1 (capability)**.** (The possibility of) action by a Teredo protocol entity, visible outside its computational environment.

definition 2 (candidate capability *or candidate*)**.** A capability that is a) externally triggered (i.e. of request/response nature) and is b) *possibly* serving more purpose than originally intended. This capability is therefore of possible use in the context of our connectivity monitor.

In the next subsections, an overview of all candidate capabilities found in a close examination of the specification is provided. For each intended capability of respectively the client, the server and the relay the original purpose is clarified. In addition, any capability introduced by the specification serving more purpose than strictly required for the functioning of Teredo is classified as an unintended capability. Where applicable, the relevant protocol primitives used in Figure 2.2 will be listed. Chapter 4 will build on this listing to determine the usefulness of these capabilities in the context of our connectivity monitor.

3.1.1 Candidate capabilities of the client

As protocol entity primarily intended to make use of capabilities instead of providing them, the amount of candidate capabilities offered by the client is limited.

client candidate 1: *A client will reply to unicast bubble packets*
By replying to bubble packets, a client establishes a mapping in a NAT-device, which can subsequently be used to contact it directly. (`return bubble` in Figure 2.2)

client candidate 2: *A client will advertise itself to other clients on a local multicast channel*
This is part of the (optional) *local client discovery procedure* and serves to identify other Teredo clients behind the same NAT. This improves routing to nearby clients.

3.1.2 Candidate capabilities of the server

The capabilities provided by the server facilitate direct connectivity between client and relay.

server candidate 1: *A server answers router solicitations of clients*
The server replies to UDP-tunneled router solicitations of clients as would a normal IPv6 router connected to a client subnet. Clients send router solicitations during initialization to obtain an address and learn their NAT status and subsequently to refresh the mappings in NAT. (`return address`, `return result` in Figure 2.2)

server candidate 2: *A server decapsulates and relays ICMPv6 request and reply traffic for its clients*
The relaying of ICMPv6 request packets is part of the relay discovery procedure of clients. In addition, servers are expected to relay ICMPv6 reply packets, but this capability is not used anywhere throughout the specification. The relaying of ICMPv6 reply packets can therefore be considered unintended. (second occurrence of `pingDestination()` in Figure 2.2)

server candidate 3: *A server relays bubbles to destinations with Teredo addresses to set up mappings in NAT*
These mappings need to be set up for each server to client and relay to client channel. Bubble relaying is done for its own clients and arbitrary relays. (second occurrence of `sendBubble()` in Figure 2.2)

3.1.3 Candidate capabilities of the relay

relay candidate 1: *A relay encapsulates IPv6 traffic to clients*
Although a relay can provide this capability to "trusted" destinations only, limiting relay services to a subset of all hosts is usually not feasible due to the limited authentication information present in such a packet. Limiting access based on topology information (e.g. IP-addresses) is possible, but not practical when serving multiple networks, as a relay is expected so serve the entire population which is able to reach it. (second occurrence of `sendTraffic()` in Figure 2.2(b))

relay candidate 2: *A relay sends bubbles to a client's server to set up mappings in NAT*
When a client is located behind certain types of NAT, the relay first has to set up mappings so it can transmit data directly to the client. For this purpose, the relay is able to send bubbles, which are relayed by the client's server. Actual data is sent once a reply to this bubble is received. (`sendBubble()` in Figure 2.2(b))

relay candidate 3: *A relay decapsulates and relays traffic from clients unless it matches a filtering rule*
Once a relay for a particular destination is found by a client, it will send traffic using this relay. A relay is not expected to relay data for every host: the specification mentions the option of filtering. The following is quoted from section 5.4 of the specification [7]:

> *Teredo relays should be able to receive packets sent over IPv4 and UDP by Teredo clients; they may apply filtering rules, e.g. only accept traffic from Teredo clients if they have previously sent traffic to these Teredo clients.*

Strangely enough, the specification does not *require* any filtering here, only an example is given. When filtering is not in place, or implemented as suggested in the example, the relay can be used as a generic packet relay, provided that its user does not trigger the filtering conditions. For the example given, avoiding filtering is trivial: a trivial direct connectivity test by a client satisfies the condition "if they have previously sent traffic". Functioning as a generic packet relay is an unintended capability. The absence of mandatory filtering is probably an omission in the specifications. (second occurrence of `sendTraffic()` in Figure 2.2)

3.1.4 Summary

In this first part of chapter 3, we presented an overview of the externally triggered capabilities in the Teredo protocol (candidates). This addressed research question 1a:

> *What capabilities does the infrastructure provide according to the specifications?*

In the examination of the Teredo specifications, we counted a total of seven candidate capabilities. Two of these capabilities belong to the client, three to the server and three to the relay.

Research question 1b took the examination one step further and required classification of the candidates found:

> *Are there any unintended capabilities introduced in the specifications?*

We found that two of the capabilities contained unintended components: the relaying of ICMPv6 reply packets by the server and functioning as a generic packet relay by a (reference implemented) relay.

3.2 Infrastructure availability

Capabilities, either intended or unintended, are only useful when the infrastructure providing them is actively deployed on the Internet. This section intends to identify the Teredo protocol entities that are, and use this information to select only the candidate capabilities found in the specification of those entities.

Of the three entities: the client, server and relay, only the server and relay are permanently reachable. The client is active only when the host running it requires access to the IPv6 Internet. On the premise that our connectivity monitor needs to replicate measurements over time, this variance in availability poses a problem. Therefore the capabilities provided by the client are no longer seen as candidates.

Due to lack of data on public deployments of Teredo servers and relays in literature, we had to take our own measurements. The intended result of these measurements is an accurate count of the available Teredo relays and servers. Both require a different approach and are treated in separate subsections, but conform to the same structure:

First, we consider possible approaches. After selection of an approach, we describe the involved entities and propose a methodology to provide measurements. A summary of results is given in this chapter, raw results can be found in the Appendices A and B.

3.2.1 Relay enumeration

The goal of relay enumeration is to identify as many operational relays on the Internet as possible. This subsection explores multiple approaches to relay enumeration. Then for the chosen approach, we explain the entities involved and provide our measurement methodology. Relay enumerations have been performed twice, in early 2010 and 2012. Results for both measurements are analysed and presented.

Possible approaches

To discover relays at large, it is useful to take a step back and reevaluate how clients end up using a particular relay. Figure 3.1 represents this in diagram form, while Figure 2.2 goes into more detail. The

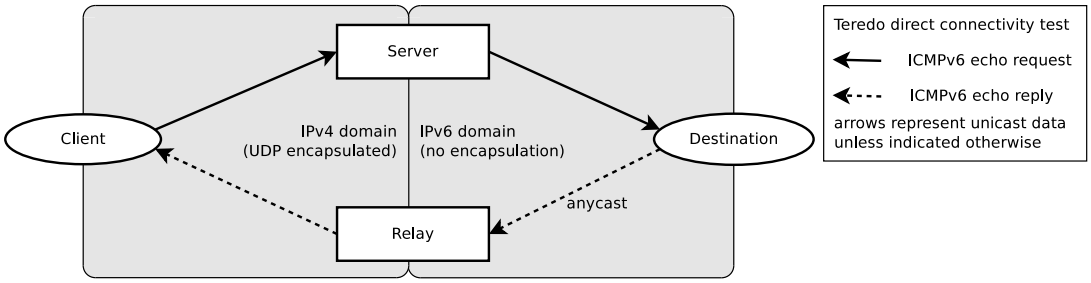


Figure 3.1: Teredo direct connectivity test

Teredo client uses a direct connectivity test by sending an ICMPv6 echo request package over UDP to the Teredo server. The server sends this packet over IPv6 to the destination. Subsequently the response packet of the destination IPv6 host is addressed to the client's Teredo source address. This response packet ends up at the nearest relay due to the layer 3 anycasting of the Teredo address prefix.

This brings us to the first way to detect networks containing relays: by monitoring the BGP announcements of the Teredo address prefix. It is important to note that this only covers a part of the total relay infrastructure, as (globally) announcing the prefix in BGP is optional[1]. In other words, it is not possible to find the full set of relays using this method, unless all relay operators decide to announce their relay service in BGP. The BGPmon project[2] provides information obtained in this way on its website.

A second, active approach to relay detection uses the source address of the relay delivering the return packet to the client over UDP (the arrow in the lower left corner of Figure 3.1). Instead of passively monitoring the collected BGP announcements, this approach completes the direct connectivity test described and illustrated earlier. By performing this test with arbitrary hosts on the Internet, our client can discover unique relays on the last step of the return path. When performed with hosts contained in networks known to provide a relay (for example by announcing the Teredo prefix in BGP), there is a high chance of discovering the exact relay location. This improves the granularity of the enumeration possible using the first approach.

To discover a larger set of relays, including those in networks not announcing the Teredo prefix, we measured responses to a mass direct connectivity test. In an ideal world, a test would be run for every native IPv6-host in existence. This approach would discover all relays in existence, even host-based ones[3]. Such tests are definitely not feasible, not only because of scale issues, but also because discovering all IPv6 hosts is (purposefully) a very difficult problem to solve on its own.

In our measurements, we chose the middle ground between passive BGP enhanced with direct connectivity tests and an exhaustive scan of every host in existence. We did perform direct connectivity tests on a massive scale, but by making some assumptions about the location of relays, our measurement size scaled much better than exhaustive search.

- First we assume that on average, networks contain several magnitudes more hosts than relays, and that every relay serves one or more hosts in the network. This allows us to find the relays in a network by connecting to hosts located therein.

- Secondly, we assume that networks share their relay services with downstream networks, even when they do not announce the Teredo prefixes (upstream) into the global routing tables. This is a direct consequence of the economics of peering and transit[4]. This allows measurement of a large network

[1] Note that a network publicly announcing the Teredo address prefix to upstream networks, without limiting its propagation scope, will provide its relay infrastructure as a service to the global Internet. This can be seen as good netizenship, but there is no direct (economic) benefit to doing this.

[2] `http://bgpmon.org/Teredo.php`

[3] a host based relay is a relay only providing service to the host on which they are active

[4] By providing relay services in a network, traffic to the Teredo prefix from downstream (usually customer) networks does not have to leave the network to reach a relay, decreasing the chance that a flow consequently uses a paid transit.

by measuring direct connectivity to hosts in smaller networks downstream of that network.

- Finally, we make no effort to discover host-based relays. These are both less interesting in a global infrastructure perspective and are much harder to detect due to reasons mentioned earlier.

With these assumptions, we trade off a (possible) decrease in completeness of the obtained result for a very large decrease in the required scale of the measurements.

Entities

This subsection explains the different entities involved in our chosen measurement approach.

Teredo client The client is under our control and is used as our measurement platform. It only has IPv4-connectivity, but contrary to the assumptions of the Teredo specifications we do not deploy a NAT in front of the client. This is done to decrease complexity of the setup, without influencing any of the measured behaviour. The Teredo client software running on the client is Miredo, *"an open-source Teredo IPv6 tunneling software, for Linux and the BSD operating systems"* [5].

Teredo server The server has no special role in the measurements and is used as specified in the protocol. However, we did choose to use our own server, to decrease the chance of being ratelimited/blocked by a server administrator due to abnormal traffic patterns. By running our own server, we decreased the chance of unnoticed administrative changes during the experiment. In addition, with the server under our control, it becomes possible to capture traffic traces at an additional location, which is useful for debugging purposes. Hosting a Teredo server requires global IPv4 and IPv6 addresses, and the ability to originate IPv6 traffic with source addresses in the Teredo address range.

Teredo relay Each of the relays to discover. These are not under our control. We detect a relay if we connect to an IPv6 destination in its service area.

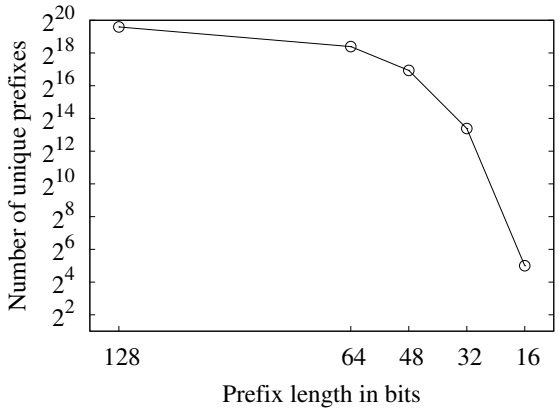
IPv6 destination These are the hosts which are one-by-one used for a direct connectivity test. The IPv6 destinations are not in any way set up for the relay enumeration. Except for a handful of a hosts, they are not under our control. We have no guarantee that a host is online, but this does not pose a problem. A "destination unreachable" response by a router directly upstream to the host also causes the involvement of the nearest relay, serving the same purpose as a response from the targeted host. Destinations dropping our ICMPv6 packets prevent us from discovering an upstream relay.

IPv6 address source We collected IPv6 addresses from a popular software mirror over the period of one year. This software mirror is, amongst other things, being used to distribute automatic software updates to a worldwide population, which makes it perfect to collect hosts in active IPv6 networks around the globe.

In total 790,253 unique IPv6-addresses were collected for use as IPv6 destination, spread out over 344,266 unique subnets (/64). Figure 3.2 summarizes some statistics on their distribution. The collected address set is widely distributed, which we show by cross-referencing the IPv6 addressing policies currently in effect[6].

[5] `http://www.remlab.net/miredo/`

[6] Addressing policy is determined by the Regional Internet Registries (RIRs) for their respective region. On a lower conceptual level, allocations to smaller networks are done by Local Internet Registries (LIRs) influenced by this policy. IANA keeps a high level administration of the IPv6 address space.



Prefix length in bits	Number of unique prefixes
128	790253
64	344266
48	125107
32	10676
16	32

Figure 3.2: Distribution statistics on the collected IPv6-addresses used in the relay enumeration

The number of unique /16's, /32's and /48's, listed in the table of Figure 3.2, give us an indication of the huge size of the dataset:

- The RIRs get allocations of size /23 from IANA. To this date these allocations have been from 32 unique /16's, which the dataset fully covers.
- A /32 is the minimal IPv6 allocation made by most RIRs to a LIR, which in turn only qualifies for such an address block if it has multiple larger downstream networks, each qualifying for a /48. Using the /32 and /48 as a lower and upper bound for the number of unique networks, we collected hosts in anywhere between 10,676 and 125,107 networks.

The size and distribution of the dataset are important, because they determine the "area" covered by our measurements. Given that our dataset covers most of the allocated address space to date, our measurements, while not conclusive, have a high chance of finding the greater part of the deployed Teredo relays.

Methodology

Preparation

1. Record traffic on the client: listen for traffic to and from the Teredo UDP port on the client[7].
2. Record traffic on the server: listen for traffic to and from the Teredo UDP port on the server[7].

Measurement

3. Send a single ICMPv6 packet to each collected IPv6 address[8]
4. This triggers a Teredo connection setup, as illustrated in Figure 2.2 and 3.1.

Analysis

5. Analyze the recorded traffic by filtering on return traffic from relays only. We are interested in the UDP-tunneled reply to our ICMPv6 echo by the relay.[9]
6. Count IPv4 endpoints. These are the Teredo relays closest to the collected IPv6 destinations.

[7] Traffic recording is done using the tcpdump binary, almost universally provided on *NIX systems

[8] In order not to trigger any rate limiting or cause interruptions during this experiment, care was taken to send more than five packets per second.

[9] For analysis the Wireshark filter `(ipv6.dst == <Teredo source addr>) && !(ipv6.src == <Teredo server addr>)` was used, where `<Teredo source addr>` and `<Teredo server addr>` are replaced by the IPv4 literals. It is important to filter out traffic with tunneled IPv6 traffic using a Teredo source address, as this traffic is sent directly by Teredo clients.

Results

Early 2010 On 28 and 29 January 2010, a first try at a large scale relay enumeration was attempted, albeit with a much smaller (and less distributed) set of 5000 IPv6-addresses. Measurements completed in little over 13 hours, resulting in the detection of 38 distinct relays. Appendix A.1 has the full list of collected relay IP addresses. Note that even this small measurement provides more results than BGP announcements alone. In comparison, at the time of the first attempt, the first approach (passive BGP monitoring) counted 8 networks publicly announcing the Teredo prefix in BGP. This emphasizes our previous point that not all networks containing relays announce the Teredo prefix publicly.

Early 2012 On January 1-3 2012 a second, large scale relay enumeration was performed, with the full set of 790,253 addresses. With these measurements, lasting 38.4 hours, 72 unique Teredo relays were detected, sending a total of 94,772 packets to the test client. Appendix A.2 has the full list of collected relay IP addresses.

Analysis of the reverse DNS for the collected relay IP addresses reveals a number of dynamic/DSL/cable/dial up hosts, which are probably host-based relays. For the purpose of a continuous monitoring service, using network relays is desirable. On a protocol level, there is no clear way to distinguish between a host-based relay and a relay providing service to a network, but the combination of WHOIS data and reverse DNS lookups will probably provide enough information to eliminate most of the host-based relays from the dataset.

Cumulative In total, during the two measurements performed 98 unique relays were found. The large amount of churn in the results can indicate two things:

- Networks hosting Teredo relays are doing this for short time periods only.
- Our results contain a high number of host-based relays.

More measurements are required to reach a definite conclusion on this topic. For our purposes, we have established that there is a high amount of relay infrastructure deployed on the Internet and classify additional measurements as future work.

3.2.2 Server enumeration

Like relay enumeration, the goal of server enumeration is to identify as many operating servers on the Internet as possible. In the following subsections, approaches to enumeration are explored and, where applicable, explained in detail.

Possible approaches

Unlike relays, Teredo servers are statically configured for each client. They are not dynamically discovered and clients usually connect to the server their software provides by default. This introduces the first way to enumerate servers: by examining documentation, source code or traffic traces of client software implementations.

The lack of dynamic discovery has direct consequences for other enumeration approaches, since the specifications provide no way for an entity to find a server. Nevertheless, some features of the protocol *do* aid in the discovery of the server. Most prominent is the format of the Teredo address prefix, which includes information on the server a client uses, as explained in Section 2.1.2. Examining the fourth to the eight byte of an Teredo address yields the server address.

The Teredo specification includes a special procedure to discover clients behind the same NAT. This optional *local client discovery procedure* aims to exchange address information with other clients, so they can avoid hairpin routing[10] at their local NAT when contacting each other. The local client discovery procedure utilizes a special multicast address[11] to realize a shared communication channel with all supporting local clients. By listening on this multicast address, bubble exchanges between local clients can be observed and additional addresses learned.

Entities

This subsection explains the different entities involved in our chosen approaches. Note that an entity can perform a different role in each approach.

Teredo client This entity has a different role in first and third of the three approaches listed above and does not participate in the second. In the first approach the client software, documentation or traffic traces are analysed to provide the server(s) it connects to by default. In the third approach, the client participates in the local client discovery, which is snooped by the multicast observer.

Multicast observer The multicast observer only plays a role in the third approach to server enumeration. It should have access to the same multicast domain as clients it intends to listen in on. Give that the Teredo multicast address in located in the "Local Network Control Block" (`224.0.0.0/24`) this effectively means the same L3 subnet.

IPv6 address source This entity is involved in the second approach. Its description is identical to the description for the "IPv6 address source" under "Relay enumeration" (section 3.2.1).

Results

Software analysis The only known vendors providing Teredo client implementations are Microsoft (for their Windows operating system) and Miredo (for *NIX). There are other existing projects implementing server and relay software (e.g. NICI-teredo and ng_teredo), but those do not provide client software and have no (demo) infrastructure running (anymore).

The client written by Microsoft uses `teredo.ipv6.microsoft.com`, which seems to be (geo-)loadbalanced over multiple IP addresses. This server is used as default for Windows XP, 2003 and Vista.

The client written by Remlab, Miredo, uses a server running their own server software, located at `teredo.remlab.net`. The Linux distribution Debian packages the Miredo server with a different default server: `teredo-debian.remlab.net`, also run by Remlab.

As part of the Internet search for client software, a number of other Teredo servers were found. These are mostly provided by companies providing network services.

In total this approach provided information on 6 unique Teredo servers, run by 4 different organisations. The raw addresses are provided in appendix B.1.

[10] e.g. routing a packet back through the interface where it was received, which not every CPE supports

[11] Allocated by IANA as 224.0.0.253, see also `http://www.iana.org/assignments/multicast-addresses/multicast-addresses.txt`

Log analysis Using the Teredo IPv6 addresses collected from the log files of a popular software mirror, we were able to identify seven unique Teredo servers, run by four different organisations. The raw addresses are provided in appendix B.2.

Local client discovery By listening in on the Teredo multicast address on a large subnet for a period of ten days, we detected the use of four unique Teredo servers, all run by Microsoft. The raw addresses are provided in appendix B.3.

Cumulative Adding the results of the three approaches and removing duplicates resulted in a total of eight unique Teredo servers, by five different organisations.

3.2.3 Summary

In the second part of chapter 3 we measured infrastructure availability by counting unique Teredo relays and servers deployed on the Internet. We presented various approaches, the methodology to make them reproduceable and analysed their results. These give us an answer to research question 1c:

How much infrastructure is publicly available?

In total, 98 unique relays and eight unique servers were found.

3.3 Conclusion

The purpose of this chapter was to answer research question 1:

What Teredo infrastructure is publicly available?

To answer this question, we presented two methods to reduce the set of all Teredo capabilities to a set of candidates (describing the "useful" capabilities of the public Teredo infrastructure)[12]. The first method, an examination of the specification, yielded a total of 8 candidate capabilities. The second method used data on the availability of public infrastructure to reduce this list of capabilities to contain only candidates for actively deployed entities.

The second reduction method removed the client capabilities from the set of candidates due to its low availability. Because the client was not designed to be continuously available, its capabilities cannot be used for measurements conducted over a longer period of time.

The results of the relay and server enumeration give reason to further examine the remaining six candidate capabilities in chapter 4. A comment must be made about the number of deployed servers found. Although continuously available, obtaining useful monitoring results using a low number of vantage points is very hard. This means the capabilities of the server can probably not be used as a single source of connectivity information.

[12] the terms *capability* and *candidate* are defined in Section 2

Chapter 4

Proposed usage of the infrastructure

In the previous chapter, a set of candidate capabilities were identified. In this chapter, we examine which of those capabilities are suitable to be employed in the connectivity monitor. This completes the multi-step reduction as depicted in the graphical document outline (Figure 1.1). Using the capabilities thus found, we design a connectivity monitor using Teredo infrastructure. The validation of this design is the subject of Chapter 5.

This chapter uses the terminology introduced in Section 2.2 to identify the different networks involved in connectivity monitoring. Defined in this terminology, the flow of this chapter is the following. First we identify capabilities which, when provided by Teredo infrastructure in a network, would make a network suitable as an agent. Subsequently we specify the required interaction between monitor, agent and target networks to obtain connectivity information. Finally we turn these required interactions into a minimal design of a monitor.

4.1 Candidate suitability for connectivity monitoring

In this section, we will further examine the candidate capabilities resulting from Chapter 3 and select those that fit in the blueprint introduced in Section 2.2.3. First we formulate requirements by deducing the criteria a candidate has to meet from the blueprint. Then we apply the criteria to the candidates, thereby completing the last reduction step. The resulting candidates will be used in the second part of this chapter.

4.1.1 Requirements for connectivity monitoring

Three way communication

The direct and indirect models for connectivity monitoring introduced in Section 2.2.3 both contain three types of participating entities: the monitor, the agent and the target. For successful measurement (as shown in Figure 2.3), the agent needs to receive a request for measurements (interacting with the monitor), do its measurements (interacting with the target) and optionally return feedback (again interacting with the monitor). This leads to our first criterion to evaluate the remaining candidates:

> *Capabilities provided by the agent need to involve both monitor and target.*

When focussing on the measurement (excluding the feedback), we can represent this interaction as `M->A->T`.

Endpoint requirements

Using the interaction representation introduced above, for a given measurement `M->A->T` we have two endpoints: `M` and `T`. `M` is under our control and can behave in any way we specify. This is not the case for `T`, because we stated in the problem description that "*[..] any room for misuse found will be used to collect information on network connectivity*". IP connectivity is later defined in very general terms

(Section 1.2). As a consequence, we cannot require `T` to contain Teredo entities. In turn, this precludes the use of the Teredo protocol for interaction `A->T`. To sum up, the second criterion to evaluate the remaining candidates is:

> *Capabilities provided by the agent to perform the actual measurements cannot use the Teredo protocol.*

Free agent selection

To find the third criterion, we quote once more from the problem statement:

> *"[..] connectivity information, when available from topographically dispersed points in the Internet, can be used to monitor connectivity of a chosen network [..]"*

From the adverbial of this sentence, two requirements can be distilled: measurements have to be performed by multiple agents *and* these agents need to be topologically dispersed. This requires the monitor to be able to address multiple distinct agents and do so on an individual (selected) basis. In terms of our representation: the selection of `A` in the interaction `M->A` should be fully controlled by `M`. When reflecting this requirement on the capability of the agent, we get the third criterion:

> *Capabilities provided by the agent should be freely accessible to the monitor*

Summarizing, we have three criteria a candidate needs to satisfy to make it a suitable agent capability.

4.1.2 Examining candidate suitability for connectivity monitoring

In this subsection we will examine each of the remaining candidates. A candidate will need to satisfy the three criteria formulated in the previous subsection to make it a suitable agent capability.

Server candidate 1: *A server answers router solicitations of clients*
The exchange of router solicitation and router advertisement messages uses a `M->T->M` interaction model. This fails the first criterion. In addition, this exchange is encapsulated in UDP, requiring Teredo protocol entities on both ends. This fails the second criterion. The third criterion is satisfied: free agent (server) selection is possible. With only one of three capabilities satisfied server candidate 1 is not a suitable agent capability.

Server candidate 2: *A server decapsulates and relays ICMPv6 request and reply traffic for its clients*
This capability satisfies the first and second property, with `M` being a client, `A` being the server and `T` an arbitrary host. Free agent (server) selection is also possible. This means server capability is suitable as an agent capability.

Server candidate 3: *A server relays bubbles to destinations with Teredo addresses to set up mappings in NAT*
A server relaying bubbles to a Teredo destination uses a `M->A->T` interaction model. This satisfies the first criterion. The second criterion is not satisfied, as bubble packets are encapsulated in UDP, which restricts `T` to Teredo clients. Finally, as seen in the previous two server candidates, free agent selection is possible. With only two out of three criteria satisfied, server candidate 3 is not a suitable agent capability.

Relay candidate 1: *A relay encapsulates IPv6 traffic to clients*
A relay encapsulating IPv6 traffic to clients uses a `M->A->T` interaction model, with `M` being a native IPv6 host, `A` being the relay and `T` a Teredo client. This satisfies the first criterion. In contrast, the second criterion is not satisfied, because the target is restricted to Teredo clients. In addition, free agent selection is also not possible, as the relay is addressed using the Teredo prefix. Due to the anycast routing of the Teredo client addresses using this prefix a monitor cannot select an agent of his choice.

Relay candidate 2: *A relay sends bubbles to a client's server to set up mappings in NAT*
A relay sends bubbles to a server when it has packets queued for one of its clients. Depending on whether you regard the native IPv6 host that sent those packets as part of the interaction model it becomes either `M->A->A->T` (IPv6 host, relay, server & client) or `M->A->T` (relay, server & client). Both of these models satisfy the first criterion. Interaction between agent and target uses Teredo, as seen in Server candidate 3. For the same reasons this fails criterion two. On top of that comes agent selection, which is not free due to the anycast routing of the Teredo prefix. Relay candidate 2 is not suitable as an agent capability.

Relay candidate 3: *A relay decapsulates and relays traffic from clients unless it matches a filtering rule*
A client sending encapsulated traffic to a relay, where it gets decapsulated and relayed to a native IPv6 host uses a `M->A->T` interaction model. This satisfies the first criterion. With the target being a native IPv6 host and the interaction `A->T` using native IPv6 packets, the second criterion is also met. The relay is also freely addressable, because the client uses IPv4 to send its encapsulated packets. Contrary to the two other relay capabilities listed, relay candidate 3 is not influenced by layer 3 anycast routing. It is suitable as agent capability.

4.1.3 Summary

This section completed the last step in reducing the set of all Teredo capabilities[1] to those that, when deployed in a network, make this network suitable as agent in our connectivity monitor.

Based on the blueprint of the connectivity monitor, three criteria were determined to judge the remaining candidate capabilities. Capabilities provided by the agent needed to involve both monitor and target, could not use the Teredo protocol and should be freely accessible to the monitor. Of the six candidate capabilities resulting from chapter 3, only two proved to be suitable as agent capabilities.

4.2 Design of a connectivity monitor

In this section we present the design of a connectivity monitor using the two agent capabilities identified in the first part of this chapter.

First we will describe steps common to measurements using both capabilities. Then, for each capability, we will give a short recapitulation, specify what we will be measuring and present our design.

The design is divided in several parts. The graphical representation of a working monitor is used as the basis of the overview. It contains all entities, the required setup and the actual measurement. Each of these subjects will subsequently be explained in separate paragraphs for each of the diagrams.

[1] as defined in Chapter 2

4.2.1 Connectivity monitoring using Teredo infrastructure

This section presents measurement steps common to both methodologies presented in the next two sections. This involves a description of the basic monitor behaviour, how to find agents and considerations for agent selection.

Basic Monitor behaviour

The monitor behaves as a client while using both capabilities. Although its behaviour is different from the client in the specifications, it does not send any illegal packets. The qualification procedure (obtain address, detect NAT) is performed, but not described here (see Chapter 2).

Agent selection

Before any measurements can be run, it is necessary to select a set of agents performing those measurements. In the ideal case, these agents are located in different parts of the network topology surrounding the target, to obtain the most complete results.

The first step of this selection is a list of all available agents. Such a list has been compiled in Chapter 3. The next step is to eliminate agents which provide duplicate results because of their similar geographical location. This is out of scope for this research, but the required data for deduplication can most likely be found in WHOIS databases or BGP AS-PATH data. Actual selection of agent networks based on topology information is left as an exercise to the reader.

4.2.2 Connectivity monitoring using Teredo servers

This section explains the use of a capability of the Teredo server to do measurements useful for connectivity monitoring. We start with a short summary of the capability we will be using.

Chapter 3 describes the capability as: "*A server decapsulates and relays ICMPv6 request and reply traffic for its clients*". This capability uses a `M->A->T` interaction model, as described in Section 4.1.2. In our (mis)use of this capability we only permit ourselves control over the client (`M`). The capability allows us to originate ICMPv6 request packets from the location of the server (`A`), by sending these packets encapsulated in UDP over IPv4.

Intended measurements

As specified in Section 2.2.3, we intend to verify the connectivity of a network and obtain properties of this connectivity. To this end we try to reproduce the output of the common ping and traceroute tools.

We can collect information on the connectivity of the target network as experienced by the network containing the server, because the server originates the ICMPv6 traffic we sent it. Single ICMPv6 request packets provide connectivity verification (ping), and ICMPv6 packets with small, increasing time-to-live (TTL) values on the underlying IPv6 packet allow hop by hop discovery of the path (traceroute).

Design

This section presents the design of a connectivity monitor using server candidate 2. A sequence diagram of this monitor is shown in Figure 4.1. The diagram is split up horizontally in three setup/measurement

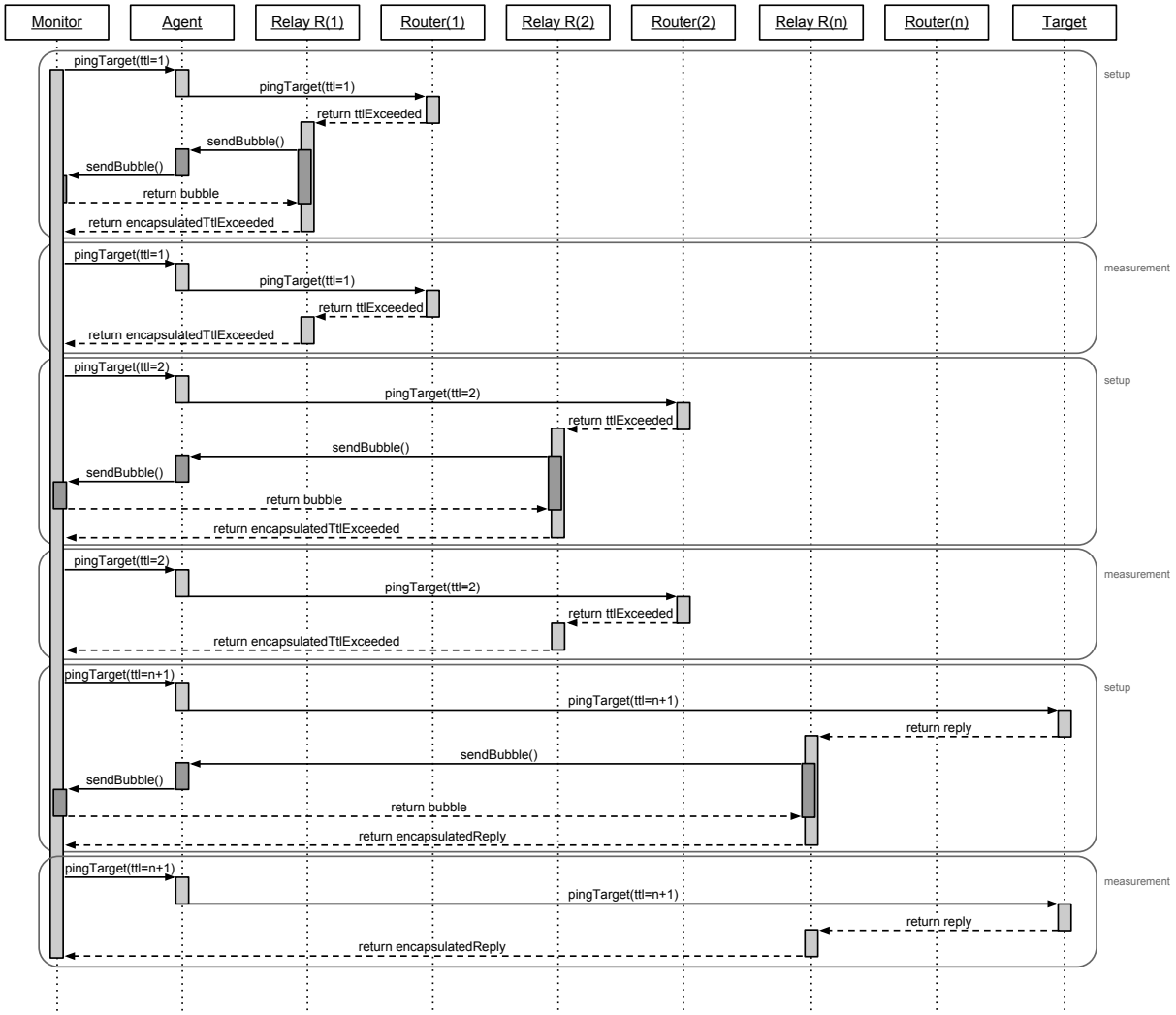


Figure 4.1: Teredo server as agent

pairs. The last pair illustrates "ping" measurement, while all three combined illustrate "traceroute". In the following paragraphs we will walk step by step through this diagram, starting with entities, then describing required setup and finishing with the actual measurements.

Entities The `Monitor`, `Agent` and `Target` entities have been explained in Chapter 2. In addition the connectivity monitor using Teredo servers uses all `Router` and corresponding `Relay` entities on the layer 3 path from the `Agent` to the `Target`. To discern entities on different hops, we number them from 1 to n. This number is the required TTL value of a IPv6 packet to let the `Router` discard the packet and send "TTL exceeded" to its nearest relay (which gets the same number). I.e. `Router(1)` is the first router on the path from `Agent` to `Target`. Before forwarding a packet, `Router(1)` decreases the TTL, in this case to zero. This causes the packet to be discarded and a "TTL exceeded" to be sent through `Relay R(1)`. This concludes our overview of the entities and the first three primitives of Figure 4.1.

Setup The ICMPv6 packets sent from `Agent` to `Target` use a source address selected by `Monitor`. The protocol requires this to be an address in the Teredo prefix and for the `Monitor` to use a particular server, it has to use an address supplied by that server. Qualification with each server is therefore required to obtain a "local" address.

Each relay used by the routers on the path from `Agent` to `Target` starts a bubble exchange when encountering the first "TTL exceeded" packet addressed to the monitor. This exchange has been previously described in Section 2.1.3. This completes our overview of the setup.

Measurement After the setup has completed for a given hop, the monitor sends another packet, this time directly forwarded by the relay. This packet is used for measurements and provides information on the router hop. The setup and measurement phases continue until we receive a ICMPv6 echo reply from the target, indicating that the TTL was high enough to reach it. In Figure 4.1, measurements using TTL={3,n} have been left out, but proceed in the same way as TTL={1,2}. When not interested in the path from agent to target, we can set the TTL high enough to skip those measurements, thereby performing a normal ping using the server.

4.2.3 Connectivity monitoring using Teredo relays

This section presents the use of capabilities of the Teredo relay to perform connectivity monitoring of a target network. It follows the same structure as the previous section, starting with a short summary of the relay capability we are using.

The capability is described as "*A relay decapsulates and relays traffic from clients unless it matches a filtering rule*" and uses a *M->A->T* interaction model. We only control the client (*M*). The capability allows us to originate arbitrary IPv6 packets from the location of the server, within the limitations of optional filtering, by sending those packets encapsulated in UDP over IPv4.

Intended measurements

Using relays as agents, we can collect information on the connectivity of a target network by originating packets from multiple locations on the internet and centrally collect the results at the monitor.

Although the capability allows the decapsulation and relaying of arbitrary IPv6 traffic and we are not limited to ICMP request traffic, we choose to leave the exploration of other measurement methods as feature work and stick to the "ping" and "traceroute" measurements described in Section 4.2.2.

Design

This section presents the design of a connectivity monitor using relay candidate 3. It proceeds analogue to the design of the monitor using a server capability: a sequence diagram of the monitor is shown in Figure 4.2. In the following paragraphs we will walk step by step through this diagram, starting with entities, then describing required setup and finishing with the actual measurements.

entities The `Monitor`, `Agent` and `Target` entities are as described in Chapter 2. The `Server` is used for qualification (not shown), relay discovery (`pingDestination()`) and bubble relaying (`sendBubble()`). `Dummy` is a IPv6 host in the service area of the `Agent` serving no other purpose than replying to the ICMPv6 echo request sent as part of the relay discovery. `Relay` is the relay used by `Target` (called `Relay R(n)` in the previous section).

setup Because we intend to use a relay to send traffic to a host outside its own service area, some setup is necessary. Although the specification does not require filtering by a relay, we assume that all relays implement the example filter. This requires a relay to check whether it has ever sent traffic *to* a client before functioning as a relay in the other direction. This filter is trivially evaded by doing a direct

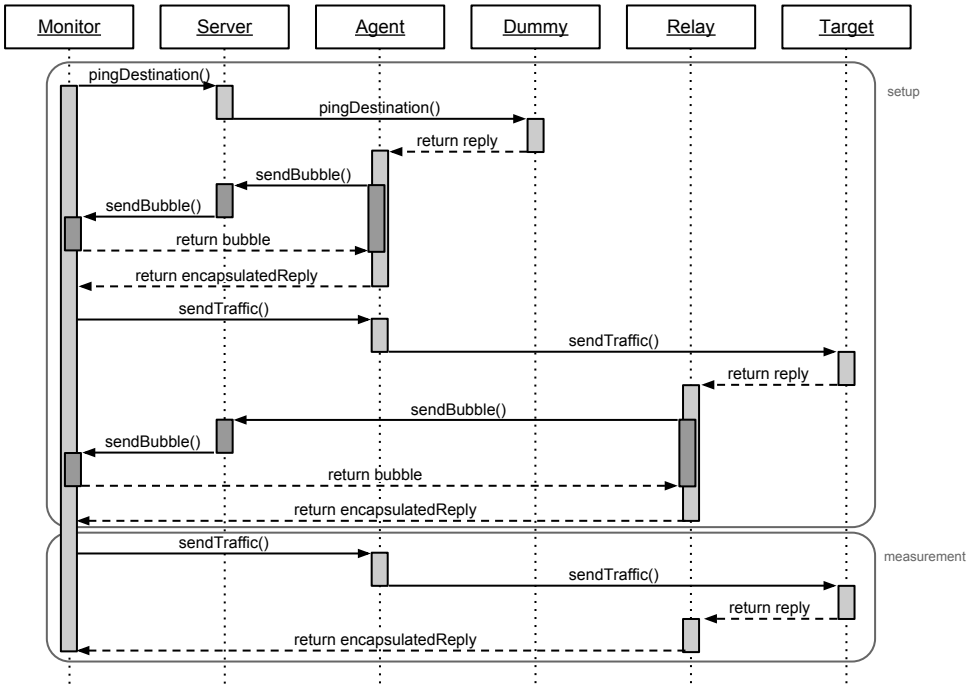


Figure 4.2: Teredo relay as agent

connectivity test with a `Dummy` in the service area of the `Agent` (the first half of the setup part of Figure 4.2). If the chosen `Agent` does not implement this filter, this step is not necessary.

The second part of the setup is getting on the list of trusted peers of the `Relay` closest to the `Target`. When it receives the first return packet destined for the `Monitor` a sequence of bubble packets is triggered. This bubble exchange has been explained previously in Section 2.1.3.

measurement The actual measurement is then done by using the `Agent` to originate a ICMPv6 request packet addressed to the `Target`. It is possible to use small, increasing TTL values on the underlying IPv6 packet to return traceroute-like results. This works similar to the previously demonstrated "traceroute" obtained using a Teredo server as agent. As the `Agent` and `Relay` have been setup previously, no additional bubble exchanges take place.

4.2.4 Summary

In the second part of Chapter 4 we presented the design of a capability monitor using two capabilities of the public Teredo infrastructure.

The first capability used was the relaying of ICMPv6 echo request packets by the Teredo server. This allows measurements to be conducted from the network of the server. We presented two uses of this capability producing results similar to the popular ping and traceroute tools.

The second capability used is the decapsulation and forwarding of Teredo client traffic by Teredo relays. (Mis)use of this capability turns the Teredo relay into a generic packet relay, allowing the "client" to originate IPv6 packets from the relay network. We described the same ping and traceroute-like measurement methods for this capability.

a note on latency measurements The path traversed by the ICMPv6 request packets is asymmetric (`A->T->M`). The latency obtained by measuring the delta between the transmission of a request and the receipt of a reply is therefore not easily compared to a conventional ping or traceroute run on the agent (`A->T->A`). Still, latency measurements conducted by our connectivity monitor can provide useful information, especially when combined with other measurements[2].

4.3 Conclusion

The purpose of this chapter was to answer research questions 2 and its subquestions:

> *Is Teredo infrastructure suitable for connectivity monitoring?*

To this end we reduced the list of candidate capabilities resulting from Chapter 3 to a list of actual agent capabilities of use in connectivity monitoring using the public Teredo infrastructure. We used the blueprint for our connectivity monitor to establish criteria. Each of the remaining candidates was evaluated, leaving two capabilities for the design phase.

In the design phase, the use of these two remaining capabilities to provide actual connectivity measurements was presented. Both capabilities provide a method to obtain ping and traceroute-like measurement data from the perspective of the network providing the infrastructure.

The results of this chapter demonstrate that in theory it is possible to repurpose Teredo infrastructure for connectivity monitoring. Clearly, this is not the intention of the protocol specification. The results obtained in this chapter answer research questions 2a and 2b.

Because we realize that there is a distinct difference between room in a specification and exploitation in practice, validation of these results is needed. In Chapter 5 we will provide guidelines for this validation, as time constraints prevented us from conducting it ourselves.

[2] e.g. by combining latency data obtained though the presented connectivity monitor with conventional latency data on the infrastructure used to conduct the monitoring as experience by the monitor.

Chapter 5

Guidelines for validation

Following the document outline depicted in 1.1, this chapter presents guidelines to validate the results obtained in Chapters 3 and 4. Validation is required to assess the feasibility of misuse for the room found in the protocol specification. It requires a move from theory to practice and test real world implementations of the Teredo protocol entities.

Due to time constraints, it was not possible to carry out a full validation as part of this research[1]. Because we attach great importance to this validation we provide guidelines for others to resume this line of research and in the mean time refrain from making claims on the real world applicability of the protocol issues found.

The chapter is organised as follows: after providing an overview of the validation to be performed, the chapter is split up in three parts, as illustrated in Figure 5.1. First we present guidelines for implementation of the monitor introduced in Chapter 4. Subsequently, we present guidelines for validation on lab scale. Thereafter we outline the staps required to move from lab to Internet scale validation. Finally, the chapter is summarized in a conclusion section.

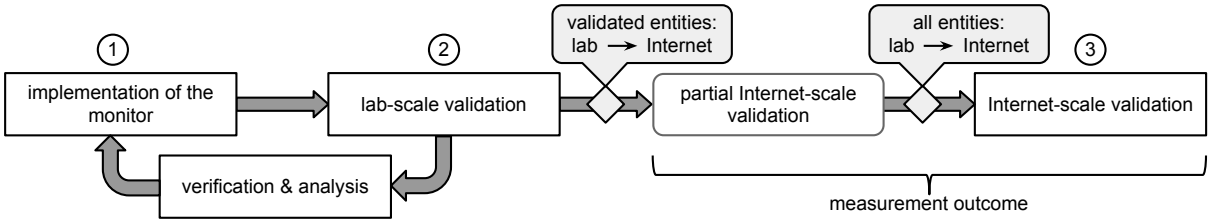


Figure 5.1: Graphical representation of the chapter outline

5.1 Validation overview

In this section we present a high level overview of the validation. We start by describing the measurements to be taken while performing a validation, followed by possible measurement outcomes and the their respective meaning. This section is followed by the actual guidelines for validation, starting with the implementation of the monitor.

5.1.1 Measurements required for validation

Measurements taken during the validation intend to verify whether connectivity monitoring using Teredo infrastructure on the Internet is feasible using the room for protocol misuse found in Chapters 3 and 4. The simplest form of measurement is to compare data gathered using our monitoring approach with alternate measurements (i.e. by using normal ping and traceroute tools from multiple locations). This is only possible if the monitoring proceeds successfully, producing meaningful results. Verifying this requires a controlled environment. Explaining deviating results is very difficult without information on the individual parts.

[1]Partial validation of selected protocol behaviour was performed to gain better understanding of protocol behaviour. The experience collected while performing these intermediate validations, combined with the wish to fully validate our results, led to guidelines presented in this chapter.

In the absence of (meaningful) results, we need an approach that enables debugging of the monitor implementation and localisation of problems in our monitor concept. In a controlled environment, the arrival, processing and departure of packets at entities can be verified with packet dumps and log files of the running software, which satisfies this requirement. We will refer to this controlled environment as our lab environment.

In the lab environment, packet traces should be taken for each entity in the setup. When using a Teredo server as agent (as depicted in Figure 4.1), that would mean packet capturing and log file analysis at the `Agent`, all relays (`Relay R(1)-Relay R(n)`), all routers (`Router(1)-Router(n)`) and `Target`. In general, more measurement data available means better localisation of problems, which is highly preferred while going through the debugging and analysis loop depicted in Figure 5.1.

Later, when moving up to Internet scale, the advantage of full control disappears step-by-step, as parts fully working in the lab get replaced by (services on) the Internet. This decreases the amount of intermediate measurements available, but increases realism. At the end of this exercise, all entities with the exception of the monitor are outside of our control and a result to the validation is obtained. We will take a further look at the guidelines for lab-scale and Internet-scale validation in Sections 5.3 and 5.4. The next subsection treats possible outcomes of the just described validation.

5.1.2 Measurement outcomes and respective meaning

Without a succesful direct comparison of two data sources proving that Teredo infrastructure can be used for connectivity monitoring[2], the outcome of a validation is ill-defined. In addition, the success of a validation does not explain the cause of this result. This section defines the meaning of different measurement results in order to classify the outcome of a validation in a finite number of conclusions. It describes the status of server and relay software used to provide the public Teredo infrastructure in terms of their implementation of the standard.

Deliberate stricter implementation of the standard Implementations are stricter than the standard and include specific measures to prevent the use of chosen TTL values (servers) and the relaying of packets to hosts outside their service area (relays). This outcome is reached when the proposed monitor concept does not work in practice due to these measures and a vendor enquiry produces confirmation of their deliberate stricter implementation.

Unintentional stricter implementation of the standard Implementations are stricter than the standard by accidental inclusion of measures to prevent the use of chosen TTL values (servers) and the relaying of packets to hosts outside their server area (relays). This outcome is reached when the proposed monitor concept does not work in practice due to these measures and a vendor enquiry produces confirmation of unintentional strict implementation.

Deliberate, communicated exact implementation of the standard Implementations implement exactly what is required according to the standard, but are aware that these problems exist and list them in their software or deployment documentation. This means operators can make a concious choice to allow the use of chosen TTL values (servers) and the relaying of packets to hosts outside their service area (relays). This outcome is reached when the proposed monitor concept works in practice, this fact is communicated to operators and a vendor enquiry produces confirmation of this deliberate exact implementation of the standard.

[2] As proposed in the first paragraph of Section 5.1.

Deliberate exact implementation of the standard Implementations implement exactly what is required according to the standard, but are aware that these problems exist. Operators unknowingly allow the use of chosen TTL values (servers) and the relaying of packets to hosts outside their service area (relays). This outcome is reached when the proposed monitor concept works in practice and a vendor enquiry produces confirmation of this deliberate exact implementation of the standard.

Exact implementation of the standard Implementations implement exactly what is required according to the standard, unaware that the standard allows room for alternate use of their infrastructure. Operators unknowingly allow the use of chosen TTL values (servers) and the relaying of packets to hosts outside their service area (relays). This outcome is reached when the proposed monitor concept works in practice and a vendor enquiry produces confirmation that the vendor is unaware of the issue.

5.2 Guidelines for implementation of the monitor

This section provides an implementor of the monitor with guidelines for implementation. We focus on the extent to which the monitor differs from the client as specified in the standard.

In general, the monitor should behave as a Teredo client. Behaviour not explicitly specified in this section should therefore be implemented as specified in the standards [7, 16], while keeping in mind that the monitor does not have to support its normal client function[3], only behave itself as such to stay protocol compliant. We start by listing functionality which a monitor should provide as a client. In the remainder of this section, we present a list of non-standard functionality required for validation. This list is split up in two parts: first we examine the behaviour of the monitor using a server as agent, then we do the same for the relay.

What follows is a non-exhaustive list of specified functionality the monitor should provide to make itself appear as a client. The corresponding primitives have been explained in previous chapters, for a refresher we refer to the relevant figures.

The client should support:

- Qualification of the client, with the purpose of obtaining server trust and a valid Teredo address (`getAddress()`, Figure 2.2).
- The ability to send tunneled ICMPv6 packets, as specified in the "direct connectivity test" (`getAddress()`, Figure 2.2).
- The ability to correlate the ICMPv6 packets sent and the replies received, as specified by the "direct connectivity test" (correlation of `pingDestination()` and `return encapsulatedReply`, Figure 2.2).
- Replying to the bubble-exchange initiated by Teredo relays (`return bubble`, Figure 2.2).

5.2.1 Implementation guidelines for server use

This section provides guidelines for the implementation of a monitor using Teredo servers as agents. Figure 4.1 provides an overview of all interaction required, which is used as the foundation of our listing of required functionality.

The following non-specified functionality is required for a working monitor:

[3] e.g. allow a host without native IPv6 access to access the IPv6 Internet.

- The ability to originate tunneled ICMPv6 packets with variable TTL values (`pingTarget(TTL=`x`)`, Figure 4.1).

- The ability to correlate the ICMPv6 packets sent and the tunneled TTL-exceeded packets sent by routers on the traffic path. The nonce in the ICMPv6 packet specified for the direct connectivity test in the Teredo protocol can be used for this purpose (`correlation of pingTarget(TTL=`x`)` and `return encapsulatedTtlExceeded`, Figure 4.1).

5.2.2 Implementation guidelines for relay use

Following the structure of the previous section, this section provides guidelines for the implementation of a monitor using Teredo relays as agents. Figure 4.2 provides an overview of all interaction required, which is used as a base for our listing of required functionality.

The following non-specified functionality is required to be implemented for a working monitor:

- Sending traffic (in our case ICMPv6 echo packets) through a relay which has not been discovered for its destination (`sendTraffic()`, Figure 4.2).

- Accepting unsolicited[4] traffic from relays (which transport the reply to the traffic previously sent) (`return encapsulatedReply` from `Relay` to `Monitor`, Figure 4.2).

5.3 Guidelines for lab-scale validation

Following the guidelines for the implementation of a monitor, this section presents guidelines for a lab-scale validation. We define lab-scale as a fully controlled environment, where every entity involved can be examined at will and downtime for replacement or reconfiguration is possible. This section starts with some general observations and a look at a sample lab topology. We will proceed by listing the entities interacting with the monitor in the topology and give suggestions to set them up. The actual description of the topology builds on the terminology introduced in Chapter 4.

First some observations about a suitable lab topology. The lab setup should support validation of both the server-as-agent and relay-as-agent approaches. In addition, it should be as realistic as possible and in order to move to a full internet scale validation, it should be relatively simple to replace parts of the topology by (services on) the Internet.

5.3.1 A sample lab scale network topology

Figure 5.2 illustrates the sample lab topology used in the remainder of this chapter. Networks `A`, `B`, `E`, `F` and `T` (for Target) have dual stack (IPv4+IPv6) connectivity, contain various Teredo entities and have a route to the nearest Teredo relay (shown as an arrow). Networks `C` and `D` only have IPv4 connectivity and do not contain any Teredo infrastructure. Network `A` contains our monitor. The router icons each represent the border routers for the AS they are connecting[5]. Each AS has its own IP-space[6] and ideally these blocks are routed on the Internet so the internal lab entities can be reached from the Internet as soon as parts get replaced.

[4] Unsolicited in the sense that (in Teredo) reply traffic from a destination most often uses the relay nearest to that destination, as it is discovered using this return-path from destination to source.

[5] For simplicity, we use one router to connect two AS. Although different from reality, this should not influence results.

[6] Note that these blocks can be very small, with the biggest networks (`A` and `T`) still fitting in a `/29`. Seven such blocks, one for each AS, and an additional equally sized block for an Internet uplink for each router (see Section 5.4) requires a `/26` in total.

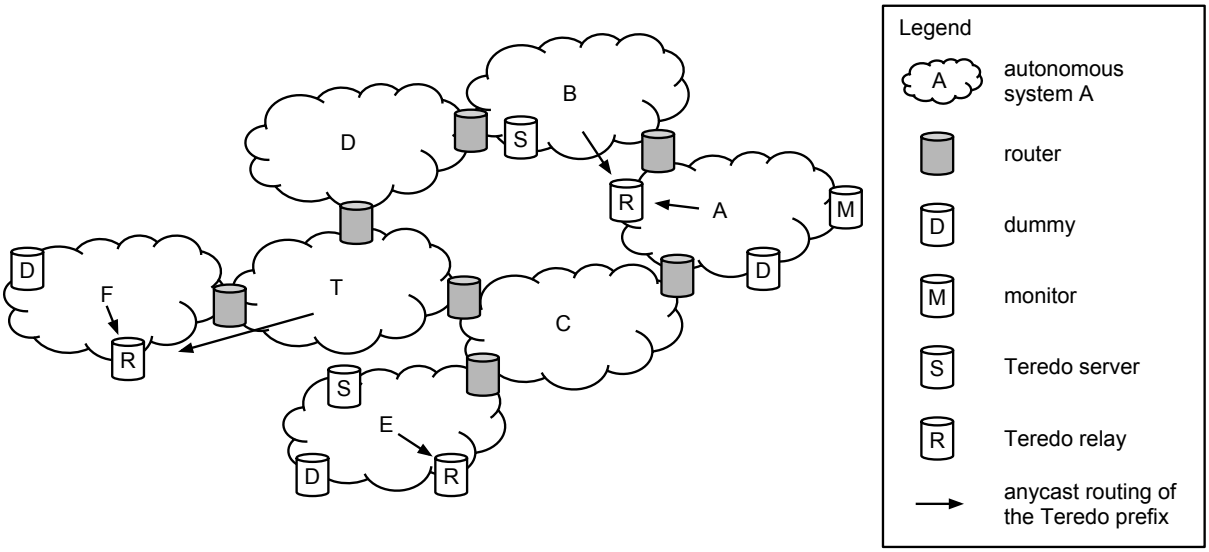


Figure 5.2: Example lab topology

Given the scale of the topology, it is advisable to use virtualisation technology to run all the required entities. In the remainder of this section, descriptions will follow on this premise.

5.3.2 Entities involved in the sample lab scale network topology

In this subsection, guidelines for the simulation of each of the entities in the sample network topology are provided.

Links When working with virtual machines for all entities, virtual links or bridges can be used to link them together. This provides the additional advantage that it becomes possible to listen "on-link", without depending on the packet dump functionality of an entity.

Routers To emulate a real network, multiple routers are required to separate the different entities in distinct L3 domains. Additionally, routers are required in order to test the server as an agent: they simulate the hops on the path from agent to target. Note that only one router on the path is required to test this behaviour, as the behaviour we use is consistent on all hops[7]. The routers need either static routes to all other AS or are required to run some routing protocol (e.g. BGP). It should be noted that one of the routers in an AS serves as the default route for the entities contained therein.

Servers The server entity is ideally located in a separate L3 domain from the monitor and runs a Teredo server daemon. We advice starting with Miredo, because it is both widely deployed and open-source. Having access to the source makes it easier to understand the results and/or debug problems.

Relays At least two relays are needed: one upstream to the monitor and one (distinct) relay upstream of the target. For reasons similar to the choice of server software we advice to start with Miredo.

[7]This is due to the use of standard IP mechanisms to signal TTL expiry.

Dummies To test the Teredo relay operating as an agent an additional dummy host is required. It only has to respond to ICMPv6 echo and should be located downstream from the chosen relay (agent).

Target network The target network does not contain any Teredo infrastructure. In order to measure its connectivity one or more IP-addresses in this network should reply to ICMPv6 echo. The simplest approach would be to bind parts of the AS IP-space to an internal interface at one of the border routers. Alternatively, an additional entity is run instead to provide this service. The objective of the lab validation is to monitor these IP-addresses from the perspective of the available Teredo infrastructure.

5.4 Guidelines for Internet-scale validation

In this section we present guidelines to move from the lab scale validation of the previous section to full validation on the public Internet. We advise to do this in a controlled fashion, in which the lab situation is step by step transformed to a partial Internet-scale validation before fully moving to the public Internet (see also 5.1). By moving in small steps, the validator keeps control over most of the infrastructure, retaining full analysis capabilities for as long as possible. Note that these individual steps can be completed in fast succession and do not necessarily increase complexity.

First we describe the required preparation for Internet validation. Then we present the actual steps for transformation from lab to Internet. We describe the required changes for each of these steps.

In order to be able to replace parts of the lab setup by (services on) the Internet, we need to give every router an uplink to an upstream router connected to the Internet. This upstream router routes the return traffic to the correct AS in the lab setup, so each AS is directly connected to the Internet.

1. A good first candidate to move out of the lab would be the server that the monitor uses to qualify and obtain an address. This server, when reachable from all the lab AS, can transparently replace the server used in the lab condition.
2. Two second candidates suitable for promotion out of the lab are the AS `C` and `D`. By eliminating them from the topology, instead of relying on the Internet connectivity of the remaining AS the links in the lab can be validated to behave no differently from Internet connectivity between AS.
3. A third candidate for replacement by real infrastructure would be all the Teredo servers. Although replacement by public Internet infrastructure duplicates large parts of the observed path between agent and target (when using servers as agents), this allows validation of real world server setup as opposed to our lab situation.
4. Final candidates for graduation from the lab are the relays and accompanying dummy hosts. After this step the remaining lab setup is not used anymore: only the monitor (and part of the internet uplink) is under our control. This is comparable to the intended setup for the connectivity monitor. By using public Teredo relays, we can validate their role as agents for connectivity monitoring.

5.5 Conclusion

After an introduction of the background on Teredo, a survey of the protocol and the available infrastructure and a proposal to use the room found in the protocol specification for the purpose of connectivity monitoring, this chapter made the case for proper validation of our findings.

Due to time limits, full validation was not performed as part of this research. Instead, the partial validation performed during our research to increase insight into the protocol was used to provide guidelines for validation, to be conducted as future research.

In this chapter, we presented an overview of the three steps required to validate the outcome of Chapters 3 and 4 on the Internet. First, we presented guidelines for implementation of the monitor. Subsequently, we presented guidelines for validation on a lab scale. These two parts paved the way for validation of our findings on full internet scale. To support the three steps, an overview of the required measurements and the possible outcomes was given. The outcome of a future validation can thus be classified in one of the five listed outcomes described in this chapter.

Chapter 6

Conclusions

This chapter concludes our examination of the public Teredo infrastructure. We start by listing future work identified throughout this research, subsequently presenting our conclusions and a concise summary of all chapters leading there. We end the chapter with recommendations for the improvement of the Teredo protocol specification.

6.1 Future work

The most prominent lead to future work is a full Internet scale validation of this work, as described in Chapter 5. To lower the barrier to such continuation, the chapter includes guidelines for implementation of the monitor, guidelines for validation on a lab scale and guidelines to move the validation to Internet scale.

A second direction worthy of pursuit would be a another survey of public Teredo infrastructure by doing traceroutes with a Teredo source address. This is can either be done using the normal traceroute binary from a host without egress-filtering of the Teredo prefix or, if validation of our work proves Teredo servers vulnerable to such misuse, use the existing Teredo servers to originate the traceroute from multiple locations. This work is worthy of pursuit because it enables relay enumeration on an even bigger scale than the one we performed. By tracerouting to one host in every allocated IPv6 address block (which is certainly within the realm of possibility) relays on all traversed paths can be discovered. This would improve the public availability of statistics on Teredo infrastructure deployment.

6.2 Conclusion

> *Can the public Teredo infrastructure be used to monitor IP connectivity of an autonomous system?*

Based on our research of the Teredo protocol specification, we believe that it is possible to use the public Teredo infrastructure to monitor IP connectivity of autonomous systems and perform other functions outside the intended scope of Teredo. Judging by the results of our survey, finding a large deployment base of public Teredo infrastructure, we believe that real world validation of this work to allow mitigation of potential room for misuse is highly advisable.

Summary

Chapter 2 provided a background on Teredo, defined connectivity monitoring and introduced a concept for such monitoring using Teredo infrastructure.

Chapter 3 analysed the capabilities of the Teredo infrastructure to find a set of candidate capabilities of potential use in this concept. Six candidates were found, with two of them classified as unintended: the relaying of ICMPv6 reply packets by the server and functioning as a generic packet relay by a (reference

implemented) relay. In the second part of the chapter, a survey was conducted to obtain full deployment statistics of Teredo servers and relays on the Internet. This resulted in the detection of 8 unique servers and 98 unique relays, evidence of widespread deployment of public Teredo infrastructure.

Chapter 4 presented the use of this infrastructure for connectivity monitoring, using the concept introduced in Chapter 2. To this end, the candidate capabilities identified in Chapter 3 were examined specifically for their suitability in our connectivity monitoring concept. Two of the candidate capabilities proved suitable, which were subsequently integrated in two monitoring approaches:

- The first approach uses Teredo servers as agents to monitor the connectivity of an AS, by using its ICMPv6 decapsulating and relaying capability, misusing the lack of TTL sanitation in the specification of the Teredo server. We use this capability to provide ping and traceroute measurements from the perspective of the server.
- The second approach misappropriates different room in the specification to use Teredo relays as generic packet relays. Because the specification does not mandate a form of filtering on relays enforcing a client-destination relation, a relay can be used for destinations it normally does not provide service to. We used this relay capability to provide ping and traceroute measurements from the perspective of the relay.

Chapter 5 started with the notion that our approach is theoretical and that actual vulnerability depends on the implementation of the specification, requiring validation on the public Internet. In the chapter, we provide guidelines for implementation of our monitor concept, guidelines for validation on lab scale and step by step instructions to scale up this validation to Internet scale. Although this research was time-limited precluding a full validation, we are confident that others can perform such validation based on our guidelines as future work. In the mean time we refrain from making claims on the real world applicability of the protocol issues found.

6.2.1 Recommendations

We recommend multiple changes to the Teredo protocol specification[7] or its updates [15, 16] to address the room for misuse found. This serves to inform current and future implementors of the specification of potential issues.

- Update section 5.3 of RFC 4380 to remove the obligation to relay ICMPv6 reply traffic.
- Update section 5.4 of RFC 4380 to add mandatory filtering of non-ICMPv6 traffic to destinations which have not previously sent traffic to the client through this relay.
- Update the security considerations of either RFC 4380 or one of its updates to include the threat of repurposing the ICMPv6 packets used for signalling. Teredo protocol entities should sanitize all such control traffic received and relayed accordingly (e.g. to preclude the free use of TTL values found in this research).

Acknowledgements

It has been quite a ride. From start in South Korea on December 1, 2009 to finish in Enschede on March 9, 2012, completing this thesis spanned 27 months. Three months of full-time research in South Korea, 21 months spent pursuing other (non-thesis) objectives, followed by three months to structure, re-experiment and write. I learned a lot. In a very broad context. I learned about conducting structured research in networking and security (which you obviously shouldn't stall for nearly two years) and about the difference between "stamppot" and "kimchi". I learned the internals of BGP and how you should behave among Korean peers. I learned too much to recount here.

All of this wouldn't have been possible without the assistance and advice from a group of people, some of which I choose to mention here.

First, I want to express gratitude to my supervisor Giovane for his help in structuring my findings and getting me on track with this thesis. Without his guidance, my work on Teredo would've never reached its current printed form.

Likewise, I would like to thank Prof. James Won-Ki Hong and the members of the DPNM lab for the tremendous hospitality during my stay in their lab. The warm welcome and friendly atmosphere in Pohang made me feel at home from day one, enabling me to connect with Korea and encouraging me to invest lots of energy in this thesis. I'm very grateful to Aiko for bringing South Korea and POSTECH in the picture and for his help in getting me in touch with both Giovane and DPNM.

Special mention is also deserved for Lennard Klein, who repeatedly challenged my writing and thinking and connected my findings to produce the server tracerouting capability as described in Chapter 4. Also appreciated are the comments by Sjoerd, who gave valuable input on the first draft (and its title). Finally, I would like to thank the people at SNT, ICTS (especially Jeroen van Ingen) and SURFnet. Without their consent and support several of my experiments would not have been possible.

Bibliography

[1] E. Aben. 6to4 - how bad is it really? https://labs.ripe.net/Members/emileaben/6to4-how-bad-is-it-really, December 2010.

[2] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (Proposed Standard), February 2001.

[3] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 1883 (Proposed Standard), December 1995. Obsoleted by RFC 2460.

[4] A. Durand, P. Fasano, I. Guardini, and D. Lento. IPv6 Tunnel Broker. RFC 3053 (Informational), January 2001.

[5] P. Gross and P. Almquist. IESG Deliberations on Routing and Addressing. RFC 1380 (Informational), November 1992.

[6] C. Huitema. An Anycast Prefix for 6to4 Relay Routers. RFC 3068 (Proposed Standard), June 2001.

[7] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (Proposed Standard), February 2006. Updated by RFCs 5991, 6081.

[8] G. Huston. Flailing ipv6. http://www.potaroo.net/ispcol/2010-12/6to4fail.html, December 2010.

[9] C. Labovitz. Six months, six providers and ipv6. http://asert.arbornetworks.com/2011/04/six-months-six-providers-and-ipv6/, April 2011.

[10] J.C.R. Licklider and R.W. Taylor. The computer as a communication device. *Science and technology*, 76(2):2, 1968.

[11] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), September 2007. Updated by RFC 5942.

[12] E. Nordmark and R. Gilligan. Basic Transition Mechanisms for IPv6 Hosts and Routers. RFC 4213 (Proposed Standard), October 2005.

[13] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), October 2008.

[14] F. Templin, T. Gleeson, and D. Thaler. Intra-Site Automatic Tunnel Addressing Protocol (ISATAP). RFC 5214 (Informational), March 2008.

[15] D. Thaler. Teredo Extensions. RFC 6081 (Proposed Standard), January 2011.

[16] D. Thaler, S. Krishnan, and J. Hoagland. Teredo Security Updates. RFC 5991 (Proposed Standard), September 2010.

Appendix A

Raw results relay enumeration

A.1 early 2010

```
65.19.157.214
66.220.18.42            tserv15.lax1.ipv6.he.net.
74.82.46.6              tserv22.tyo1.ipv6.he.net.
80.84.224.224           teredo-relay.proserve.nl.
83.170.1.40             miredo.teleport-iabg.de.
85.94.192.205           horus.seeweb.it.
87.251.43.68            teredo.bit.nl.
89.235.48.21            user21-48-235-89.dalunet.cz.
93.186.177.144          dedi0262.oxilion.nl.
93.32.145.156           93-32-145-156.ip33.fastwebnet.it.
128.214.231.107
129.187.10.29
130.244.6.54            kst-teredo-1.tele2.net.
130.89.162.183          reviresco.student.utwente.nl.
145.220.0.46            145.220.EARLY-REGISTRATION.of.SURFnet.invalid.
193.225.13.90           teredo.niif.hu.
194.210.30.212
195.140.195.68          temp.nat64.trex.fi.
195.222.111.211         miredo.airwire.ie.
203.178.8.1             kotemachi03.tokyo6to4.net.
203.98.50.11
209.51.161.14           tserv4.nyc4.ipv6.he.net.
209.51.161.58           tserv12.mia1.ipv6.he.net.
209.51.181.2            tserv9.chi1.ipv6.he.net.
213.154.235.74
213.172.48.141          ns1.euro6ix.com. # 141.48.172.213.in-addr.arpa. 3108 IN
                                         # CNAME 141.136-29.48.172.213.in-addr.arpa.
213.197.30.87           teredo-relay.concepts-ict.net.
216.218.221.6           tserv20.hkg1.ipv6.he.net.
216.218.224.42          tserv8.dal1.ipv6.he.net.
216.66.38.58            tserv21.tor1.ipv6.he.net.
216.66.80.26            tserv5.lon1.ipv6.he.net.
216.66.80.30            tserv6.fra1.ipv6.he.net.
216.66.80.90            tserv24.sto1.ipv6.he.net.
216.66.80.98            tserv23.zrh1.ipv6.he.net.
216.66.84.42            tserv10.par1.ipv6.he.net.
216.66.84.46            tserv11.ams1.ipv6.he.net.
216.93.240.34           bos7-ibn3981.mis.towardex.com.
```

A.2 early 2012

```
2.103.153.134           host-2-103-153-134.as13285.net.
41.73.40.135            41-73-40-135.neoinx.net.
62.106.11.41            host-62-106-11-41.phpoint.net.
65.19.157.214
74.82.46.22
74.82.46.26
78.227.21.21            dhl45-1-78-227-21-21.fbx.proxad.net.
80.98.15.143            catv-80-98-15-143.catv.broadband.hu.
81.201.60.207           teredo.pilsfree.net.
82.139.244.167          uag-da1.hego-it.com.
82.197.196.170
82.66.8.182             bdn33-1-82-66-8-182.fbx.proxad.net.
```

83.141.117.122	forkboy.irishbroadband.ie.
83.170.1.40	miredo.teleport-iabg.de.
84.24.70.98	54184662.cm-5-1b.dynamic.ziggo.nl.
84.44.148.116	xdsl-84-44-148-116.netcologne.de.
85.225.52.75	c-4b34e155.442-1-64736c11.cust.bredbandsbolaget.se.
85.91.7.151	yolan.magnet.ie.
87.251.43.68	teredo.bit.nl.
89.179.29.206	
89.235.48.21	static21-48-235-89.dalunet.cz.
92.233.96.48	cpc1-craw4-0-0-cust47.croy.cable.virginmedia.com.
94.79.45.23	94-79-45-23.broadband.progtech.ru.
115.156.196.36	
129.187.254.248	teredo-relay2.lrz.de.
130.244.6.54	kst-teredo-1.tele2.net.
130.89.149.144	dropsleutel.snt.utwente.nl.
138.246.2.248	host248-2.natpool.mwn.de.
139.18.17.123	teredo.rz.uni-leipzig.de.
145.220.0.46	145.220.EARLY-REGISTRATION.of.SURFnet.invalid.
158.36.183.230	msuagpub.hive.no.
173.19.122.182	173-19-122-182.client.mchsi.com.
178.206.13.139	
190.213.127.129	
193.167.245.121	hexpack.csc.fi.
193.219.168.112	kg.su.lt.
193.219.61.57	teredo.litnet.lt.
193.225.13.90	teredo.niif.hu.
194.210.30.212	
194.255.205.20	0xc2ffcd14.linknet.dk.telia.net.
195.140.195.140	teredo.bb.trex.fi.
195.22.205.162	orazio.seabone.net.
195.222.111.211	miredo.airwire.ie.
195.242.156.211	brunel.lhr.uk.as44980.net.
200.148.169.130	200-148-169-130.customer.tdatabrasil.net.br.
202.78.240.37	santayana.catalyst.net.nz.
203.178.17.1	
203.178.8.1	kotemachi03.tokyo6to4.net.
209.51.161.58	tserv12.mia1.ipv6.he.net.
209.51.172.18	
212.118.32.72	
212.188.8.39	mustique.mtu.ru.
212.227.34.209	teredo.gw-tunnel6to4-a.bs.kae.de.oneandone.net.
212.25.17.162	max.0x1b.ch.
212.76.39.67	teredo.ipv6.aster.pl.
213.141.64.6	gnstat.gavlenet.com.
213.206.99.60	ipv6-relay.widexs.nl.
216.218.142.110	
216.218.217.190	
216.218.217.234	
216.218.221.26	6to4.hkg1.he.net.
216.218.224.42	tserv8.dal1.ipv6.he.net.
216.66.38.58	tserv21.tor1.ipv6.he.net.
216.66.64.138	
216.66.77.226	
216.66.80.238	
216.66.80.250	6to4.lon1.he.net.
216.66.80.98	tserv23.zrh1.ipv6.he.net.
216.66.84.158	6to4.ams1.he.net.
216.66.84.170	
216.66.84.182	
217.153.128.58	miredo.ipartners.pl.
217.31.204.152	
217.31.204.153	
217.31.204.154	

Appendix B

Raw results server enumeration

B.1 Collected from software analysis

```
teredo.ipv6.microsoft.com
teredo.remlab.net
teredo-debian.remlab.net
softwareutotrans.consulintel.com
teredo.managemydedi.com
teredo.trex.fi
```

B.2 Collected from server logs

```
65.55.158.118           NXDOMAIN
74.63.116.220           74-63-116-220.reverse.managemydedi.com.
83.170.6.76             miredo.svr02.mucip.net.
94.245.115.184          NXDOMAIN
94.245.121.251          NXDOMAIN
94.245.121.253          NXDOMAIN
195.140.195.140         teredo.bb.trex.fi
```

B.3 Collected from local client discovery

```
94.245.115.184          NXDOMAIN
94.245.121.253          NXDOMAIN
94.245.121.251          NXDOMAIN
65.55.158.118           NXDOMAIN
```