Virtual Environment for IPv6 Analysis

Ricardo Alexander Saladrigas

Advisor: Dr. Anna Calveras

Barcelona
DEDICATION

To my parents, for giving me opportunities of immeasurable value and supporting me and my farfetched ideas.

To my near family, for their accumulated efforts of improving our collective life.

And to Maria Alexandra Siso, Robert Baumgartner, Alyssa Juday and Marc Ramirez for keeping me sane.
I extend my gratitude to everyone that has made my work possible.

I express my thanks to the communities of VirtualBox, StackOverflow, ServerFault and Ubuntu Help as well as the Reddit communities for Linux and Networking for answering all my technical questions in detail and without prejudice.

I would like to thank Dr Anna Calveras for her guidance and patience.
RESUMEN

Nuestro objetivo fue la creación de una red compuesta de máquinas virtuales conectadas de forma específica a través de interfaces virtuales y con una serie de protocolos pre configurados que permiten la fácil creación de túneles IPv6 y traductores IPv6 a IPv4. Esta red les permitirá a profesores y estudiantes analizar y observar tráfico IPv6 real sin la necesidad de una red física.
La red está compuesta de múltiples Máquinas Virtuales Ubuntu y una Máquina Virtual Windows 7. La red puede ser fácilmente instalada en un ordenador corriendo Ubuntu o una distribución basada en Ubuntu. Un USB arrancable fue desarrollado para usar la red en un ordenador sin la necesidad de una instalación o de un sistema operativo específico. Todas las máquinas virtuales Linux pueden fácilmente ser controladas a través de una terminal sin necesidad de clave utilizando una serie de scripts. Todas las máquinas virtuales Linux son capaces de realizar capturas de paquetes en cualquier de sus interfaces. Los paquetes capturados son enviados por SSH a la maquina física y mostrados en una instancia local de Wireshark. Los paquetes capturados no son guardados en la máquina virtual. Una vez en Wireshark, las capturas pueden ser guardadas y abiertas en cualquier ordenador para su estudio.

Las Máquinas Virtuales fueron probadas para el estudio de las siguientes funciones y protocolos relacionados con IPv6:

El proceso de autoconfiguración fue probado tanto para direcciones locales como globales en Linux (Ubuntu) y Windows. Observaciones notables incluyen el uso de DAD optimista en Windows (se asume que no habrá duplicados para las direcciones aleatorias) y como Linux revisa la existencia de duplicados para todas sus direcciones pero por defecto ignora cualquier duplicado detectado.
Protocolos como HTTP funcionan de forma transparente en IPv6 mientras que protocolos como ICMP y FTP requieren nuevas versiones o modificaciones para funcionamiento.
Los túneles manuales y los túneles automáticos ISATAP son muy similares a nivel de cada paquete pero tiene requerimientos de configuración y procesos de creación muy distintos. No existe un protocolo unificado para un Túnel Broker, pero un servicio de Túnel Broker fue creado utilizando openVPN y openSSL para ofrecer túnel cifrado automático con autenticación de usuario.
NAT64 utiliza un grupo de reglas para traducir IPv6 and IPv4 y viceversa. Requiere una configuración larga y complicada pero les permite a clientes IPv6 acceder a direcciones IPv4 de forma transparente.
Todos los protocolos han sido pre configurados y pueden ser iniciados y detenidos fácilmente por el usuario, permitiendo el enfoque en el estudio de los detalles de los paquetes.
ABSTRACT

The objective was to build a network composed of virtual machines connected via virtual interfaces with a set of preconfigured protocols that allow easy creation of IPv6 tunnels and IPv6 to IPv4 translators. This network allows teachers and students to analyze and observe real IPv6 traffic without the need of a real physical network.

The network is composed of multiple Ubuntu Virtual Machines and a Windows 7 Virtual Machine. It can be easily installed on most Ubuntu based distributions. A bootable USB drive was also developed to run the network without the need of an installation or a specific Operating System. All Linux machines can be accessed and controlled via terminal with a set of scripts and without the need of a password. All Linux machines allow doing packet capture on any interface of the virtual network. Captured packets are sent through SSH to the physical machine and shown on Wireshark. No captured packets are kept on the Virtual Machines. Once saved, the Wireshark files can be opened on any computer.

The Virtual Network was used to study the following features and protocols related to IPv6 in order to learn their details and inner workings:

**Autoconfiguration** was tested for local and global addresses for both Linux (Ubuntu) and Windows. Notable observations include how Windows utilizes optimistic DAD (assumes no duplicates for its random addresses) and how Linux checks for duplicates of all its addresses but has a default behavior of ignoring duplicates.

Protocols such as **HTTP** work transparently over IPv6 while protocols like **ICMP** and **FTP** require new versions or modifications in order to function.

**Manual Tunnels** and Automatic **ISATAP tunnels** look very similar on the packet level but have different configuration requirements and creation processes. There are no unified protocols for **Tunnel Brokers**, but a Tunnel Broker server was created using openVPN and openSSL to offer automatic encrypted tunneling with user authentication.

**NAT64** uses a set of rules to translate IPv6 to IPv4 and vice versa. It requires a lengthy configuration in order to work but allows IPv6 clients to access IPv4 addresses transparently.

All protocols are preconfigured and can be started and stopped easily by users, allowing focusing on the study of its packet details.
El nostre objectiu era la creació d'una xarxa composta de màquines virtuals connectades de forma específica a través d'interfícies virtuals i amb un seguit de protocols pre configurats que permeten fàcilment la creació de túnels IPv6 i traductors IPv6 a IPv4. Aquesta xarxa els permetrà a professors i estudiants analitzar i observar trànsit IPv6 real sense la necessitat d'una xarxa física. La xarxa està composta de múltiples Màquines Virtuals Ubuntu i una Màquina Virtual Windows 7. La xarxa pot ser fàcilment instal·lada en un ordinador corrent Ubuntu o una distribució basada en Ubuntu. Un USB arrancable va ser desenvolupat per utilitzar la xarxa en un ordinador sense la necessitat d'una instal·lació o d'un sistema operatiu específic. Totes les màquines virtuals Linux poden fàcilment ser controlades a través d'una terminal sense necessitat de clau utilitzant una sèrie d'scripts. Totes les màquines virtuals Linux són capaces de realitzar captures de paquets en qualsevol de les seves interfícies. Els paquets capturats són enviats per SSH a la màquina física i mostrats en una instància local de Wireshark. Els paquets capturats no són emmagatzemats a la màquina virtual. Un cop a Wireshark, les captures poden ser guardades i obertes a qualsevol ordinador per al seu estudi.

Les Màquines Virtuals van ser provades per a l'estudi de les funcions i protocols relacionats amb IPv6:

El procés d'autoconfiguració va ser provat tant per adreces locals com globals en Linux (Ubuntu) i Windows. Observacions notables inclouen l'ús de DAD optimista a Windows (s'assumeix que no hi haurà duplicats per a les adreces aleatòries) i Linux revisa l'existència de duplicats per a totes les adreces però per defecte ignora qualsevol duplicat detectat. Protocols com HTTP funcions transparent en IPv6 mentre que protocols com ICMP i FTP requereixen noves versions o modificacions per a funcionament. Els túnels manuals i els túnels autèntics ISATAP són molt similars a nivell de cada paquet però tens requeriments de configuració i processos de creació molt diferents. No hi ha un protocol unificat per a un Túnel Broker, però es va crear un servei de Túnel Broker utilitzant OpenVPN i OpenSSL per oferir túnel xifrat automàtic amb autenticació d'usuari.

NAT64 utilitza un grup de regles per traduir IPv6 and IPv4 i viceversa. Requereix una configuració llarga i complicada però permet a clients IPv6 accedir a adreces IPv4 de manera transparent.

Tots els protocols han estat pre configurats i poden ser iniciats i detinguts fàcilment per l'usuari, ademés de permetre l'enfocament en l'estudi dels detalls dels paquets.
# GENERAL INDEX

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEDICATION</td>
<td>i</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>RESUMEN</td>
<td>iii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>iv</td>
</tr>
<tr>
<td>RESUM</td>
<td>v</td>
</tr>
<tr>
<td>GENERAL INDEX</td>
<td>vi</td>
</tr>
<tr>
<td>FIGURE INDEX</td>
<td>xi</td>
</tr>
<tr>
<td>TABLE INDEX</td>
<td>xvii</td>
</tr>
<tr>
<td>1. motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2. Objectives</td>
<td>1</td>
</tr>
<tr>
<td>2. INTRODUCTION</td>
<td>2</td>
</tr>
<tr>
<td>2.1. IP version 6</td>
<td>2</td>
</tr>
<tr>
<td>2.1.1. Motivation</td>
<td>2</td>
</tr>
<tr>
<td>2.1.2. Differences</td>
<td>2</td>
</tr>
<tr>
<td>2.1.3. Header</td>
<td>3</td>
</tr>
<tr>
<td>2.1.4. Multicast Addresses</td>
<td>4</td>
</tr>
<tr>
<td>2.1.5. Present State</td>
<td>5</td>
</tr>
<tr>
<td>2.1.6. Autoconfiguration</td>
<td>6</td>
</tr>
<tr>
<td>2.1.7. Transition Mechanisms</td>
<td>8</td>
</tr>
<tr>
<td>2.2. Virtualization technologies</td>
<td>10</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>2.2.1.</td>
<td>Bochs</td>
</tr>
<tr>
<td>2.2.2.</td>
<td>Cooperative Linux</td>
</tr>
<tr>
<td>2.2.3.</td>
<td>Linux-VServer</td>
</tr>
<tr>
<td>2.2.4.</td>
<td>OpenVZ</td>
</tr>
<tr>
<td>2.2.5.</td>
<td>QEMU</td>
</tr>
<tr>
<td>2.2.6.</td>
<td>User-mode Linux</td>
</tr>
<tr>
<td>2.2.7.</td>
<td>VirtualBox</td>
</tr>
<tr>
<td>2.2.8.</td>
<td>Xen</td>
</tr>
<tr>
<td>2.2.9.</td>
<td>Selected System</td>
</tr>
<tr>
<td>3.</td>
<td>WORKING VIRTUAL ENVIRONMENT</td>
</tr>
<tr>
<td>3.1.</td>
<td>Client Network</td>
</tr>
<tr>
<td>3.2.</td>
<td>IPv6 network</td>
</tr>
<tr>
<td>3.3.</td>
<td>IPv4 network</td>
</tr>
<tr>
<td>3.4.</td>
<td>vboxnet0 interface</td>
</tr>
<tr>
<td>3.5.</td>
<td>Diagram of the network</td>
</tr>
<tr>
<td>3.6.</td>
<td>Interfaces</td>
</tr>
<tr>
<td>3.7.</td>
<td>Software installed</td>
</tr>
<tr>
<td>4.</td>
<td>INSTALLATION, CONFIGURATION AND USAGE</td>
</tr>
<tr>
<td>4.1.</td>
<td>Autoconfiguration</td>
</tr>
<tr>
<td>4.1.1.</td>
<td>Software: Radvd</td>
</tr>
<tr>
<td>4.2.</td>
<td>Ping</td>
</tr>
<tr>
<td>4.2.1.</td>
<td>Software: ping6</td>
</tr>
<tr>
<td>4.3.</td>
<td>Webpage service over IPv6</td>
</tr>
<tr>
<td>4.3.1.</td>
<td>Software: apache</td>
</tr>
</tbody>
</table>
4.4. DNS name server ................................................................. 24
  4.4.1. Software: dnsmasq ....................................................... 24
4.5. FTP server ........................................................................ 26
  4.5.1. Software: vsftpd .......................................................... 26
4.6. Manual tunnel .................................................................... 27
  4.6.1. Software: ip ................................................................. 27
  4.6.2. Software: route ........................................................... 30
  4.6.3. Configuration .............................................................. 30
4.7. ISATAP tunnel .................................................................... 31
  4.7.1. Software: isatapd ........................................................ 31
4.8. Tunnel Broker ................................................................... 32
  4.8.1. Software: OpenVPN ................................................... 32
  4.8.2. Software: OpenSSL .................................................... 41
4.9. NAT64 .............................................................................. 42
  4.9.1. Software: tayga ........................................................... 42
4.10. Kernel variables ............................................................... 43
  4.10.1. Software: sysctl ........................................................ 43
5. TRAFFIC ANALYSIS ................................................................ 45
  5.1. Autoconfiguration ............................................................ 45
    5.1.1. Necessary machines: ................................................ 45
    5.1.2. Autoconfiguration without router ................................. 45
    5.1.3. Autoconfiguration with router .................................... 52
    5.1.4. Autoconfiguration on Window with router .................... 62
    5.1.6 Autoconfiguration with static Global address .................. 72
5.2. Ping .......................................................................................................................... 78
  5.2.1. Necessary machines .......................................................................................... 79
  5.2.2. Diagram of the connection .............................................................................. 79
  5.2.3. Between hosts using local address ................................................................. 80
  5.2.4. Between host and external machine using global address .............................. 87

5.3. HTTP request over IPv6 .......................................................................................... 92
  5.3.1. Necessary machines ........................................................................................ 92
  5.3.2. Diagram of the connection ............................................................................ 92
  5.3.3. HTTP request using DNS ipv6 ........................................................................ 93

5.4. FTP over IPv6 .......................................................................................................... 106
  5.4.1. Necessary machines ....................................................................................... 106
  5.4.2. Diagram of the connection ............................................................................ 107
  5.4.3. FTP connection and file transfer over IPv6 ................................................... 108

5.5. Manual Tunnel ........................................................................................................ 120
  5.5.1. Necessary machines: ...................................................................................... 120
  5.5.2. Diagram of the connection ............................................................................ 121
  5.5.3. Ping request over Manual Tunnel ................................................................. 122

5.6. ISATAP tunnel ......................................................................................................... 132
  5.6.1. Necessary machines ....................................................................................... 132
  5.6.2. Diagram of the connection ............................................................................ 133
  5.6.3. Pings over ISATAP tunnel .............................................................................. 133

5.7. Tunnel Broker ......................................................................................................... 143
  5.7.1. Necessary machines ....................................................................................... 143
  5.7.2. Diagram of the connection ............................................................................ 144
5.7.3. Key exchange, connection and ping over Tunnel Broker tunnel .................. 144

5.8. NAT64 ............................................................................................................. 153

5.8.1. Necessary machines .................................................................................. 153

5.8.2. Diagram of the connection ....................................................................... 154

5.8.3. IPv6 only ping to an IPv4 guests ............................................................. 155

5.9. Kernel variables .......................................................................................... 166

5.9.1. net.ipv6.conf.eth1.router_solicitations .............................................. 166

5.9.2. net.ipv6.conf.eth1.router_solicitation_interval .................................. 167

5.9.3. net.ipv6.conf.eth1.router_solicitation_delay ....................................... 167

5.9.4. net.ipv6.conf.eth1.dad_transmits ......................................................... 168

5.9.5. net.ipv6.conf.eth1.accept_dad ............................................................... 168

5.9.6. net.ipv6.conf.eth1.accept_ra ................................................................. 169

5.9.7. net.ipv6.conf.eth1.hop_limit ................................................................. 171

6. CONCLUSION AND FUTURE RECOMMENDATIONS ................................. 173

7. BIBLIOGRAPHY ............................................................................................. 176

8. Glossary ........................................................................................................... 182

9. ATTACHMENTS ............................................................................................. 184
Figure 1: IPv6 Header diagram .............................................................................................................. 3
Figure 2: Prefix and Identifier ............................................................................................................... 7
Figure 3: Diagram of the virtual Network .............................................................................................. 17
Figure 4: Diagram of Autoconfiguration Capture (No Router) ............................................................... 46
Figure 5: First package of autoconfiguration, multicast message ........................................................... 47
Figure 6: DAD package on autoconfiguration ....................................................................................... 48
Figure 7: Unresponded router solicitations ............................................................................................. 49
Figure 8: 5th packet of the autoconfiguration (no router) ...................................................................... 50
Figure 9: Routing table, Ubuntu-2, no router ......................................................................................... 51
Figure 10: Diagram of Autoconfiguration Capture (With Router) .......................................................... 52
Figure 11: First packet of autoconfiguration with router (End of Router DAD) ..................................... 54
Figure 12: First packets of autoconfiguration with router ...................................................................... 54
Figure 13: Router Advertisement .......................................................................................................... 55
Figure 14: Multicast Listener Report for Global Addresses (1) .............................................................. 57
Figure 15: Random Global Address DAD ............................................................................................. 58
Figure 16: MAC generated Global Address DAD .................................................................................. 59
Figure 17: Final Multicast Listener Report ............................................................................................. 60
Figure 18: Routing table, Ubuntu-2, with router ................................................................................... 61
Figure 19: Window's Addresses for current session ................................................................................ 62
Figure 20: Diagram of Autoconfiguration Capture for Windows (With Router) ..................................... 63
Figure 21: First Windows DAD packet .................................................................................................. 64
Figure 22: Windows Router Solicitation ............................................................................................... 64
Figure 23: Windows Multicast Listener Report 1 .................................................................................. 65
Figure 24: Router Advertisement in Windows Capture .......................................................................... 67
Figure 25: Windows Multicast Listener Report 2 .................................................................................. 68
Figure 26: First Global DAD of Windows ............................................................................................ 69
Figure 27: Second Global DAD of windows ......................................................................................... 69
Figure 58: Machines needed for HTTP request ................................................................. 92
Figure 59: Diagram of HTTP capture .............................................................................. 93
Figure 60: DNS query ...................................................................................................... 94
Figure 61: DNS query response ....................................................................................... 96
Figure 62: TCP connection being established ................................................................. 97
Figure 63: HTTP GET ....................................................................................................... 98
Figure 64: HTTP GET TCP ACK ...................................................................................... 99
Figure 65: HTTP OK ......................................................................................................... 100
Figure 66: HTTP OK ACK .............................................................................................. 101
Figure 67: Second DNS query (For Favicon) ................................................................. 102
Figure 68: Second DNS reply (For Favicon) ................................................................ 103
Figure 69: Second TCP connection ............................................................................... 103
Figure 70: favicon HTTP GET ....................................................................................... 104
Figure 71: HTTP not found response .............................................................................. 105
Figure 72: End of TCP connection .................................................................................. 105
Figure 73: Machines needed for FTP ............................................................................... 107
Figure 74: Diagram of FTP capture ................................................................................ 109
Figure 75: TCP connection established for FTP ............................................................ 109
Figure 76: FTP response, welcome packet ................................................................. 110
Figure 77: FTP welcome, TCP ACK .............................................................................. 110
Figure 78: FTP User request .......................................................................................... 110
Figure 79: FTP user request, TCP ACK ........................................................................ 111
Figure 80: FTP response, needs password .................................................................... 111
Figure 81: FTP response, needs password, TCP ACK ............................................... 111
Figure 82: FTP request, password ................................................................................ 112
Figure 83: FTP response, login successful .................................................................... 112
Figure 84: FTP response, login successful, TCP ACK .............................................. 112
Figure 85: FTP request, SYST ...................................................................................... 113
Figure 86: FTP response, SYST .................................................................................... 113
Figure 87: FTP response, SYST, TCP ACK ................................................................. 113
Figure 147: accept_dad set to 0 from the accept_dad_0 capture .................................................. 169
Figure 148: accept_dad set to 2 from the accept_dad_2 capture .................................................. 169
Figure 149: IPv6 disabled from interface ......................................................................................... 169
Figure 150: accept_ra set on 1, ip forwarding disabled ................................................................. 170
Figure 151: accept_ra set on 1, ip forwarding enabled ................................................................. 170
Figure 152: accept_ra set on 0, ip forwarding disabled ................................................................. 171
Figure 153: Default hop limit ......................................................................................................... 172
Figure 154: Modified (50) hop limit ............................................................................................ 172
Table 1: Scope parameter of Ipv6 header ................................................................. 5
Table 2: Comparison of Virtualization Environments ............................................... 11
Table 3: Interfaces of the Network ........................................................................... 20
Table 4: Software installed ....................................................................................... 21
Table 5: ip command objects .................................................................................. 28
Table 6: Header comparison for NAT64 .................................................................. 165
1. MOTIVATION

1.1. Motivation

Due to the exhaustion of IPv4 addresses almost all internet operator have started using IPv6 in one way or the other. While most of the users around the world are still not directly connected with a direct IPv6 address, many autonomous systems on the network infrastructure of many countries use IPv6 in one way or the other.

According to (Cisco) several developed countries have surpassed 30% of IPv6 deployment, such as the United States (31.9%), Panama (30.1%), Belgium (41.04%), Germany (37.7%), Norway (36.08%), and France (30%). Countries such as Colombia, Venezuela, Spain, Japan, Australia, Saudi Arabia and India are around 20% in IPv6 deployment. Most countries are closer to 10% of the deployment. The African Continent has low levels of deployment with some exceptions such as South Africa, Sudan and Tunisia. A more detailed view on some of these deployment percentages is detailed in section 2.1.5.

All the mentioned countries have a ratio of IPv6 Transits Autonomous Systems to IPv4 Transits Autonomous Systems that is over 50%, with cases such as Norway, Germany and Japan having a ratio of over 80%.

However, given how virtualization tools are often used to solve many computers problems and how publically available they are, the decision was reached to build an entire simulated network of virtual machines working in both IPv6 and IPv4 that could simulate and visualize internet traffic and that could be installed on a mid-level computer on a classroom in order to physically visualize the new features of IPv6 and its interactions without the need on an external connection to the IPv6 network.

1.2. Objectives

- To create a network made out of connected virtual machines
- To simulate different scenarios pertaining IPv6 and its differences and new features.
- To study the IPv6 protocol and its inner workings.
- To study several of the transition strategies used between IPv6 and IPv4
- To analyze traffic in every scenario and see the protocol implementation in action.
2. INTRODUCTION

2.1. IP version 6

2.1.1. Motivation

The reasons for the creation of a new IP protocol where:

“

- Exhaustion of IP Class B Address Space.
- Exhaustion of IP Address Space in General.
- Non-hierarchical nature of address allocation leading to flat routing space.
“ (IETF, 1993)

While Classless InterDomain Routing made class B addresses obsolete and fixed the non-hierarchical structure (to a level), the reorganization of addresses was only a temporal fix to the IPv4 exhaustion problem.

RFC1454 summarized the candidates for the next IP protocol. Among them was simple IP (SIP), which was basically IPv4 with a bigger address space. Its successor, Simple IP Plus (SIPP), was chosen to officially be IPv6. However, SIPP originally used 64 bits for the addresses which was still considered insufficient. This lead to an adaptation to an address space of 128 bits which allows for $340,282,366,920,938,463,463,374,607,431,768,211,456$ addresses.

The opportunity was taken to add some important improvements and features on the protocol in order to decrease overhead and simplify processing.

2.1.2. Differences

- 128 bit addresses are used over the 32 bit addresses in IPv4, creating a pool of $3.4 \times 10^{38}$ addresses, a pool big enough to simplify assignation, avoid the complex classed routing used on IPv4 and avoid exhaustion on the conceivable future.
- Multicasting is introduced. It allows sending one packet to a group of destinations.
- Stateless address autoconfiguration is included. This allows interfaces to gain addresses automatically using neighbor discovery for communications inside the local network. A global address is assigned by receiving a router solicitation through multicast from which the interface can get parameters for the global address. This mechanism can be turn off in preference of DHCPv6 or a static addresses if necessary.
- IPsec (Packet encryption) is integrated into the protocol as an option.
- The header has been simplified, removing many of the rarely used fields.
- Routers do not fragment packets.
- Packet header is not passed through checksum. Packet integrity is assumed by checks in layers 2 and 4.
- Time to live is now Hop Limit
- The variable length header was replaced by a 40 byte header with optional extensions.

2.1.3. **Header**

![IPv6 Header diagram](image)

**Version**: 4 bits, the version of the IP protocol used.

**Traffic class**: can be used as identification for different types of traffic. This allows QOS (Quality of Service) by prioritizing some type of traffic over another. Differentiated Services Code Point is supported on IPv6 for this same purpose, where the first 6 bits are used to determine Per Hop Behavior as defined in RFC2474.

**Flow label**: The 20-bit Flow Label field in the IPv6 header may be used by a source to label sequences of packets for which it requests special handling by the IPv6 routers, such as non-default quality of service or "real-time" service. This aspect of IPv6 is, at the time of writing, still experimental and subject to change as the requirements for flow support in the Internet become clearer. Hosts or routers that do not support the functions of the Flow Label field are required to set the field to zero when originating a packet, pass the field on unchanged when forwarding a packet, and ignore the field when receiving a packet.
**Payload length:** length of the payload in octets, the substitute for the packed length used in IPv4. Since the IPv6 header has a fixed length, only the payload varies.

**Next Header:** Identifies the type of header that follows the IPv6 header, in other words, the protocol contained inside the packet.

**Hop Limit:** Replacement for time to live, the number is decremented with each hop. Packet is discarded if it reaches zero.

**Source Address:** the 128 address associated with the origin interface.

**Destination Address:** the 128 address associated the destination interface.

### 2.1.4. Multicast Addresses

As explained in RFC 4291, the structure of an IPv6 Multicast address is as follows:

```
| 8  | 4  | 4  | 112 bits |
+----+----+-------+-----------------------------+
|11111111|flgs|scop| group ID |
+--------+-----------------------------+
```

(IETF, 2006)

The first bit of the flags indicates whenever it as well known (permanently assigned) address (0) or a temporal address (1), and the scope indicates the limit of the scope of the multicast group:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Interface-Local scope</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Link-Local scope</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Admin-Local scope</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Site-Local scope</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>(unassigned)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>(unassigned)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Organization-Local scope</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>(unassigned)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>(unassigned)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>(unassigned)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>(unassigned)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Therefore, the address \textit{FF01:0:0:0:0:0:0:1} identifies all IPv6 nodes in interface-local scope and the address \textit{FF02:0:0:0:0:0:0:1} identifies all IPv6 nodes in the link local scope.

Changing the last bit will change the address to all router addresses, so the address \textit{FF01:0:0:0:0:0:0:2} identifies all IPv6 routers in interface-local scope and the address \textit{FF02:0:0:0:0:0:0:2} identifies IPv6 routers in the link local scope.

Every IPv6 multicast address has a corresponding MAC address for layer 2 routing derived by the four low-order octets \textit{OR’ed with the MAC 33:33:00:00:00:00}. An IPv6 multicast address of \textit{ff02:aaaa:bbbb::1:3} would have a corresponding MAC address of \textit{33:33:00:00:01:03}.

\subsection{2.1.4.1. Solicited-Node multicast addresses}

A solicited-node address is an IPv6 multicast address valid within the local-link. Solicited-Node multicast addresses are used in Neighbor Discovery Protocol for obtaining the layer 2 link-layer addresses of other nodes.

The solicited-node address is computed from the MAC address by taking the low-order 24 bits of the MAC and appending them to the prefix \textit{FF02:0:0:0:1:FF00::/104}.

So if we have a MAC address of \textit{08:00:27:95:f4:0c}, its Solicited-node address would be \textit{ff02::1:ff95:f40c}.

\subsection{2.1.5. Present State}

According to (Cisco) leaders on the American Continent on IPv6 deployment are the United States (31.9\%) and Panama (30.1\%). These percentages are calculated from the number of accessible IPv6 prefixes allocated to the country, the ratio of Transits Autonomous Systems that are IPv6 over IPv4, the ratio of IPv6 enabled websites and the ratio of users using IPv6. According to this data the United States has 37.79\% of its IPv6 prefixes accessible, a ratio of 61.53\% of IPv6 Transit Autonomous Systems, 46.87\% of its websites are IPv6 enabled and 10.4\% of its user have IPv6 addresses.
On Europe Belgium and Germany have the highest levels of IPv6 deployment with 46.13% and 37.76% respectively. On Belgium 41.04% of the allocated IPv6 prefixes are accessible; the ratio of IPv6 Transit Autonomous Systems is 77.11%, 48.72% of the websites are IPv6 enabled and 26.3% of the users have IPv6 addresses. On Germany 52.46% of the allocated IPv6 prefixes are accessible; the ratio of IPv6 Transit Autonomous Systems is 82.48%, 45.44% of the websites are IPv6 enabled and 11.5% of the users have IPv6 addresses.

On Asia Japan and Malaysia have the highest levels of IPv6 deployment with 29.82% and 28.7% respectively. On Japan 48.13% of the allocated IPv6 prefixes are accessible; the ratio of IPv6 Transit Autonomous Systems is 82.87%, 27.04% of the websites are IPv6 enabled and 5.38% of the users have IPv6 addresses. On Malaysia 36.08% of the allocated IPv6 prefixes are accessible; the ratio of IPv6 Transit Autonomous Systems is 65.5%, 52.87% of the websites are IPv6 enabled and 5.11% of the users have IPv6 addresses.

On Oceania New Zealand and Australia have the highest levels of IPv6 deployment with 23.29% and 22.49% respectively. On New Zealand 36.77% of the allocated IPv6 prefixes are accessible; the ratio of IPv6 Transit Autonomous Systems is 75.79%, 47.27% of the websites are IPv6 enabled and only 0.71% of the users have IPv6 addresses. On Australia 27.46% of the allocated IPv6 prefixes are accessible; the ratio of IPv6 Transit Autonomous Systems is 68.08%, 49.71% of the websites are IPv6 enabled and 1.07% of the users have IPv6 addresses.

African countries have low level of IPv6 deployment with some notable exceptions like Tunisia with 22.82% and South Africa with 20.87%. On Tunisia 50% of the allocated IPv6 prefixes are accessible; the ratio of IPv6 Transit Autonomous Systems is 91.28%. Regardless of these high numbers the percentage of deployment on Tunisia goes down because no websites or users are registered as using IPv6. On South Africa 39.52% of the allocated IPv6 prefixes are accessible; the ratio of IPv6 Transit Autonomous Systems is 73.84%, 49.17% of the websites are IPv6 enabled but only 0.21% of the users having IPv6 addresses.

2.1.6. Autoconfiguration

DHCP was introduced as a protocol for automatically allocating address and other information (such as DNS servers) to clients in a successful attempt to simplify configuration for certain networks. A successor exists for IPv6, DHCPv6. However, both protocols are stateful, meaning that the centralized server keeps a table of registered clients. IPv6 includes a system for stateless (decentralized) autoconfiguration.

Each interface in IPv6 is capable of having several addresses; a feature that was very rare on IPv4 unless multihoming was being used. However, in IPv6 each address has its own defined scope (global, site-local and link-local) and is unique (if unicast) for its scope. Since the uniqueness is only guaranteed for the scope, routers are not allowed to forward packets outside of their scope.

```
fe80::20c:29ff:fec2:52ff
```

Prefix Network identifier
128 bit IPv6 addresses have two parts, a network prefix and an identifier. While the network prefix is dependent on the network the identifier is uniquely generated, the method of generation depending on the link layer protocol used.

In the case of Ethernet (and its variations), the address is generated from the MAC address that is unique to the NIC. The process of creating the address is as follows:

For link local addresses the identifier is 64 bits long, so the 48 bit MAC address is stretched by adding 0xFFFE (16 bits) at the 24th bit of the MAC. For example, if we have a starting MAC of:

00-0C-29-C2-52-FF

The extension leads to an address of:

000C:29FF:FEC2:52FF

Furthermore, the bit at the 6th position of the first octet is inverted. So the final identifier is:

020c:29ff:fec2:52ff

Finally, the prefix for local link addresses and netmask are added, fe80::/64

fe80::20c:29ff:fec2:52ff

The process is similar for global address generation, but in many cases the identifier is generated randomly to hide the user’s MAC from external users.

However, while the network prefix for link local is static, the network prefix for global is not. In order to determine its global address the closest router must implement Router Advertisements, a mechanism where ICMPv6 packets are sent with information about the network such as its prefix and netmask. Clients are able to send out Router Solicitations to solicit this information from the router. Once received they can use it to generate their global addresses.
2.1.6.1.1. Duplicate Address Detection

Before an address is assigned the system checks for the uniqueness of the address using a process called Duplicate Address Detection (DAD). ICMPv6 packets are sent with the destination set to the tentative address and with the origin set to an empty origin address (::). These packets are known as Neighbor Solicitations and any other interface on the same local scope with the target address responds with a Neighbor Advertisement. If a response is detected the address cannot be assigned (since there is a duplicate) and an error is thrown. If no interface answers, it is assumed the tentative address is unique and assigned to the interface.

2.1.7. Transition Mechanisms

The transition to IPv6 is expected to last very long. While the rhythm of growth is increasing, many services are dependent on IPv4. Therefore a smooth “overnight” transition is not possible. What is likely to happen is the creation of IPv6 islands among the IPv4 network. These islands will grow in size until they will compromise the majority of the network. During the process of transition both protocols will exists simultaneously, even though they are not intercompatible. For this reason, several protocols have been created for moving IPv6 traffic through IPv4 only areas (and vice versa) that will help manage the interaction during the transitional process. Some of the most important ones are:

2.1.7.1. Dual Stack

Less of an actual mechanism and more of an implementation of devices that work in both IPv4 and IPv6, dual stack is the industry name given to systems where the stack from which packets are processed is separate for each version of the protocol. Therefore, a stack exists for the processing of IPv4 packets and an entirely different stack exists for the processing of IPv6 packets. This means that a tool used for network management or an application or service which was designed for IPv4 may not immediately work with IPv6. It also means that a single machine can work in both protocols simultaneously, even though it will not be able to pass packets from one protocol to the other seamlessly.

Dual-stack is defined in RFC 4213. Most modern operating systems and network hardware already have it implemented.

2.1.7.2. Tunneling

In some situations it may be impossible to reach an IPv6 destination (or island) without going through anIPv4 only route. In those cases a tunnel is usually the best solution.

A tunnel is a method where IPv6 packets are “encapsulated” inside IPv4 packets, like an additional layer of the OSI model, where the two layers are processed by their separate stacks.
This include an extra level of processing overhead, but it allows IPv6 only traffic to go through IPv4 network.

However, tunnels require configuration in order to work properly. Such configuration can be done manually, but requires access to both endpoints of the tunnel. This is what is known as a manual tunnel and once properly set up on both ends (and if the routing table is properly configured) the devices will be able to send IPv4 packages from one end to the other like they were a single hop on the IPv6 stack. The system will take care of encapsulating the information and transporting it over IPv4 per the specification.

Several protocols exist to simplify or automate tunnel creation. These are known as automatic tunnels. Two of the most widely used are:

2.1.7.2.1. ISATAP

ISATAP (Intra-Site Automatic Tunnel Addressing Protocol) is an automatic tunneling protocol. This protocol requires client software and a preconfigured centralized server reachable through IPv4 and connected to the IPv6 network.

The client, which is connected to an IPv4 network, generates a virtual IPv6 interface with a link local address based on the physical interface’s current IPv4 address by prepending fe80::0200:5efe:... for globally (unique) addresses, or fe80::0000:5efe:... for private addresses, followed by the 32 bits of the IPv4 address. As an example, if the IPv4 interface has an address of 192.10.10.150

ISATAP would use as its link-local IPv6 address:

fe80::0200:5efe:192.10.10.150

The centralized ISATAP server needs to be able to send router advertisements (with a network prefix). When the ISATAP client tries to connect it creates a temporal tunnel with its link local address and sends a router solicitation. When the router advertisement is received a global address is generated for the tunnel. After the process is done a tunnel is created between the client and server using the global addresses.
2.1.7.2.2. Tunnel Broker

Tunnel broker is the name given to a service or server which controls access to a tunnel. While the exact role of a tunnel broker can vary a lot, most tunnel brokers must fulfill the followings tasks:

“

• provide IPv6 connectivity to a subscribed client
• manage a set of X.509 certificates and keys and a certification authority (CA)
• check authorization of a client
• assign a fixed IPv6 prefix to each client (either /64 or /128 for a single address)
• adjust routing according to prefix-/address-assignment
• on subscription of a new client, create client configuration for server and as archive file for client
• handle subscription information

” (Strauf)

2.1.7.3. NAT64

NAT64 is a protocol that allows IPv6 only clients to connect with IPv4 clients without the need of a tunnel at the cost of additional configuration and overhead for the router doing the translation.

The protocol puts IPv4 addresses inside IPv6 addresses by reserving a configurableIPv6 network prefix and using the last 32 bits to contain the IPv4 address. When the router receives a packet with the especial prefix it converts it to an IPv4 address and routes it, and then does the inverse conversion for the response. If the machines on IPv6 are configured to do all IPv4 request as IPv6 request with the correct header (or if the DNS server is also configured if needed), the process is transparent on both sides and depends entirely on the router doing the conversion.

2.2. Virtualization technologies

On our current context, virtualization means simulating an entire computer and its Operating System on another operating system via software. This allows creating an entire network of devices with their simulated network interfaces connected in any way we want without the need of the physical hardware.

In order to create the necessary virtual network a number of virtual environments where considered.
## Table 2: Comparison of Virtualization Environments

<table>
<thead>
<tr>
<th>Name</th>
<th>Compatible Hosts</th>
<th>Compatible Guests</th>
<th>Supports Networking functions</th>
<th>Supports IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bochs</td>
<td>Windows, Linux, Unix, Mac OS...</td>
<td>Windows, Linux...</td>
<td>Partial</td>
<td>No</td>
</tr>
<tr>
<td>Cooperative Linux</td>
<td>Windows</td>
<td>Linux</td>
<td>Partial</td>
<td>No</td>
</tr>
<tr>
<td>Linux - Vserver</td>
<td>Linux</td>
<td>Linux</td>
<td>Yes</td>
<td>Yes (additional patch)</td>
</tr>
<tr>
<td>OpenVZ</td>
<td>Linux</td>
<td>Linux</td>
<td>Yes</td>
<td>On development</td>
</tr>
<tr>
<td>QEMU</td>
<td>Windows, Linux, Unix, Mac OS...</td>
<td>Windows, Linux, Unix, Mac OS...</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>User-mode Linux</td>
<td>Linux</td>
<td>Linux</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>VirtualBox</td>
<td>Windows, Linux, Unix, Mac OS...</td>
<td>Windows, Linux, Unix, Mac OS...</td>
<td>Yes</td>
<td>Yes (Partially)</td>
</tr>
<tr>
<td>Xen</td>
<td>Linux, Windows Xp</td>
<td>Linux, Windows Xp</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>VMware</td>
<td>Windows, Linux</td>
<td>Windows, Linux</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 2.2.1. Bochs

Bochs is a portable x86-64 emulator with a wide variety of options for hardware emulation, often used among OS developer as a debug tool or for running obsolete programs in incompatible Operating Systems.

While it has a wide variety of hosts Operating Systems Bochs is focused on development and lacks a lot of important network options such as a network connection between hosts and guests. It only includes an option to emulate a generic network card. There is also a lack of IPv6 support.
2.2.2. Cooperative Linux

Cooperative Linux was born from a project aimed at fixing the performance problems of most emulation systems. It allows the simultaneous execution of the Windows and Linux kernel while sharing the same hardware resources.

Cooperative Linux uses a modified version of the Linux kernel and a variety of customized hardware drives. It lacks the portability and ease of install of other environments, but allows for much higher performance.

2.2.3. Linux-VServer

Linux-VServer is a virtualization environment aimed towards running several Linux instances in the same hardware, allowing a single physical server to run several virtual servers. There are options to create network connections between the virtual serves with additional IPv6 support through a patch. It requires the installation of a customized kernel.

2.2.4. OpenVZ

OpenVZ utilizes a modified Linux Kernel to allow the physical layer to execute several instances of Linux simultaneously in containers. Only the layers over the kernel are virtualized, so all running instances share the same kernel. This limits the variety of distributions that could be run simultaneously but allows an increase in performance.

OpenVZ allows for wide variety of network connections. IPV6 supports, however, is still in development.

2.2.5. QEMU

QEMU is a system designed for CPU emulation. A program designed for a different CPU architecture can be emulated on User mode and an entire OS can be emulated using Native mode. It is compatible with a number of peripheries including network cards. Different operating systems can be installed using compatible images kept by the community. File structure can be stored in an especial format that only occupies the space currently used by the virtual hard drive and that can easily be translated from one physical computer to the other.

QEMU allows for network configuration of each individual virtual machine and interconnection between them through a backend network as explained on its documentation (QEMU Developers). It also includes IPv6 support.
2.2.6. User-mode Linux

User-mode Linux allows multiple Linux instances to run in the user space of a host Linux system. It is meant to emulate different versions of Linux over the same kernel and not to emulate entire networked systems.

2.2.7. VirtualBox

VirtualBox is one of the most popular virtual environments. It supports a longs list of operating systems both as guests and hosts, and can simulate many hardware peripheries.

VirtualBox can be installed with relative ease on most operating systems. On execution, the guest system is run as a window inside the host. Guest systems can also be run with no graphical interface.

Installed virtual machines and their files can be stored in images that can be moved among physical machines easily and that only occupy the space currently used by the virtual hard drive.

Network supports is also very wide and configurable. Virtual machines can be interconnected in private networks, linked to the host or connected to the outside network through NAT. IPv6 is fully compatible.

2.2.8. Xen

Xen is a virtual environment that runs in privileged mode. Its structure allows it to execute a modified Linux as a main system where several virtual machines (with conventional Operational Systems) can be executed.

2.2.9. Selected System

After consideration and trial of several systems, the decision to use VirtualBox was reached for the following reasons:

- Easy of install and compatibility. One of the objectives of the project is to be able to install the Virtual Network on several physical machines for testing. All the virtual machines can be stored in files and saved.
- VirtualBox is well documented; both by the publisher and the community, so most issues can be resolved easily.
- It can be configured both as a graphical interface and a through terminal, which allows creating several scripts that take care of the configuration for the installation of the virtual environment.
- Networking options are easy to configure and allow creating a custom network easily.
3. WORKING VIRTUAL ENVIRONMENT

The following virtual environment has been built in order to test the capabilities of IPv6 both on its own and while interacting with an IPv4 network.

The virtual network is composed of three sub networks.

3.1. Client Network

The client network is composed of:

- Ubuntu-1
- Ubuntu-2
- Windows-1
- UbuntuRouter
- Ubuntu-3

The client network composes all machines representing an end user, as well as the router which acts as the gateway for most of the clients to the outside network.

Ubuntu-1 and Ubuntu-2 represents two IPv6 only clients running a Linux based OS connected to the outside network by UbuntuRouter. Their function is to test how IPv6 only clients interact with the IPv6 network both directly and through tunnels, how they interact with the IPv4 networks through transitional protocols, as well as auto configuration using link local addresses and global addresses by interacting with UbuntuRouter.

Windows-1 is also a client connected to UbuntuRouter. Its role is to check if the autoconfiguration is in any way different on Windows than it is on Ubuntu.

Ubuntu-3 is the only client that is particularly different. It is not connected to UbuntuRouter but rather directly to the IPv4 network. It is an IPv4 only client that we will use to study how IPv4 clients can connect to the IPv6 network.

3.2. IPv6 network

The IPv6 network is composed of:

- UbuntuRouter-ipv6-gateway
- UbuntuRouter-ipv6-1
- Ubuntu-ipv6-webserver-DNS
The IPv6 network is composed of machines that simulate a purely IPv6 network with a single
webserver/DNS/FTP server. IPv6 clients can easily and directly access the DNS and webserver
without any configuration. It will allow us to test and check if normal web operations or FTP
connections are different when handled over IPv6.

UbuntuRouter-ipv6-gateway and UbuntuRouter-ipv6-1 represent trunk routers that move traffic
from the clients to the server and vice versa.

Ubuntu-ipv6-webserver-DNS is an Ubuntu server running an Apache2 instance, a DNS daemon
and a FTP daemon. It can only be accessed via IPv6.

3.3. IPv4 network

The IPv4 network is composed of:

- UbuntuRouter-ipv4-gateway
- UbuntuRouter-ipv4-1
- Ubuntu-ipv4-webserver-DNS
- Ubuntu-tunnelbroker
- Ubuntu-ipv6-webserver-tunnel

The IPv4 network is composed of machines that simulate a standard IPv4 network and an IPv6
island isolated from the rest of the IPv6 network with its own webserver.

UbuntuRouter-ipv4-gateway and UbuntuRouter-ipv4-1 act as trunk routers that move traffic
around the IPv4 network.

Ubuntu-ipv4-webserver-DNS is an Ubuntu server running both and Apache2 instance and a DNS
daemon. It can only be accessed via IPv4.

Ubuntu-tunnelbroker and Ubuntu-ipv6-webserver-tunnel are machines on an IPv6 island inside
the IPv4 Network. The Ubuntu-tunnelbroker has one interface in each network. The idea behind
this is to study tunnels and their use connecting to IPv6 only addresses that can only be reached
through an IPv4 only area. The Ubuntu-ipv6-webserver-tunnel is simply a webserver inside the
island.
3.4. vboxnet0 interface

Most virtual machines in the network are connected to the vboxnet0 virtual interface using IPv4. The vboxnet0 interface is used to connect the virtual machines to the physical host machine using SSH for configuration and wireshark capture. Routing is disabled for this interface and it is never used for anything beyond configuration or passing wireshark captures back into the host.
3.5. Diagram of the network

Figure 3: Diagram of the virtual Network
## 3.6. Interfaces

<table>
<thead>
<tr>
<th>Name</th>
<th>ip ver</th>
<th>ip address</th>
<th>Connected network</th>
<th>Emulated interface</th>
<th>MAC Address</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu-1-eth0</td>
<td>ipv4</td>
<td>192.168.56.11</td>
<td>vboxnet0</td>
<td>AMD PCNet FAST III</td>
<td>0800273DC6B1</td>
<td>Meant for ssh remote scripting</td>
</tr>
<tr>
<td>Ubuntu-1-eth1</td>
<td>ipv6</td>
<td>Static</td>
<td>local-network</td>
<td>AMD PCNet FAST III</td>
<td>08002795F40C</td>
<td></td>
</tr>
<tr>
<td>Ubuntu-2-eth0</td>
<td>ipv4</td>
<td>192.168.56.12</td>
<td>vboxnet0</td>
<td>AMD PCNet FAST III</td>
<td>08002733FCBF</td>
<td>Meant for ssh remote scripting</td>
</tr>
<tr>
<td>Ubuntu-2-eth1</td>
<td>ipv6</td>
<td>Dynamic</td>
<td>local-network</td>
<td>AMD PCNet FAST III</td>
<td>0800270B4677</td>
<td></td>
</tr>
<tr>
<td>Ubuntu-3-eth0</td>
<td>ipv4</td>
<td>192.168.56.23</td>
<td>vboxnet0</td>
<td>AMD PCNet FAST III</td>
<td>080027DD834F</td>
<td>Meant for ssh remote scripting</td>
</tr>
<tr>
<td>Ubuntu-3-eth0</td>
<td>ipv4</td>
<td>16.0.0.1</td>
<td>ipv4-2</td>
<td>AMD PCNet FAST III</td>
<td>08002766A5D4</td>
<td>Direct ipv4 host</td>
</tr>
<tr>
<td>windows-1-eth0</td>
<td>ipv6</td>
<td>Dynamic</td>
<td>local-network</td>
<td>AMD PCNet FAST III</td>
<td>080027125D4C</td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter-eth0</td>
<td>ipv4</td>
<td>192.168.56.13</td>
<td>vboxnet0</td>
<td>AMD PCNet FAST III</td>
<td>08002788AC8B</td>
<td>Meant for ssh remote scripting</td>
</tr>
<tr>
<td>UbuntuRouter-eth1</td>
<td>ipv1</td>
<td>2001:db8:0:1::1</td>
<td>local-network</td>
<td>AMD PCNet FAST III</td>
<td>0800278C66B6</td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter-eth2</td>
<td>ipv4</td>
<td>10.0.0.3</td>
<td>ipv4-1</td>
<td>AMD PCNet FAST III</td>
<td>080027606087</td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter-eth3</td>
<td>ipv6</td>
<td>2001:db8:0:2::2</td>
<td>ipv6-1</td>
<td>AMD PCNet FAST III</td>
<td>0800270F6A30</td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter-ipv4-gateway-eth0</td>
<td>ipv4</td>
<td>10.0.0.2</td>
<td>ipv4-1</td>
<td>AMD PCNet FAST III</td>
<td>080027A0595C</td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter-ipv4-gateway-eth1</td>
<td>ipv4</td>
<td>11.0.0.1</td>
<td>ipv4-3</td>
<td>AMD PCNet FAST III</td>
<td>0800272FCDAC</td>
<td></td>
</tr>
<tr>
<td><strong>UbuntuRouter-ipv4-gateway-eth2</strong></td>
<td>ipv4</td>
<td>192.168.56.14</td>
<td>vboxnet0</td>
<td>AMD PCNet FAST III</td>
<td>080027606087</td>
<td>Meant for ssh remote scripting</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------</td>
<td>----------------</td>
<td>--------</td>
<td>-------------------</td>
<td>---------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td><strong>UbuntuRouter-ipv4-gateway-eth3</strong></td>
<td>ipv4</td>
<td>16.0.0.2</td>
<td>ipv4-2</td>
<td>AMD PCNet FAST III</td>
<td>0800270F6A30</td>
<td></td>
</tr>
<tr>
<td><strong>UbuntuRouter-ipv4-1-eth0</strong></td>
<td>ipv4</td>
<td>11.0.0.2</td>
<td>ipv4-3</td>
<td>AMD PCNet FAST III</td>
<td>080027427239</td>
<td></td>
</tr>
<tr>
<td><strong>UbuntuRouter-ipv4-1-eth1</strong></td>
<td>ipv4</td>
<td>12.0.0.1</td>
<td>ipv4-4</td>
<td>AMD PCNet FAST III</td>
<td>0800271A8AA0</td>
<td></td>
</tr>
<tr>
<td><strong>UbuntuRouter-ipv4-1-eth2</strong></td>
<td>ipv4</td>
<td>192.168.56.15</td>
<td>vboxnet0</td>
<td>AMD PCNet FAST III</td>
<td>080027FF05EE</td>
<td>Meant for ssh remote scripting</td>
</tr>
<tr>
<td><strong>Ubuntu-ipv6-webserver-tunnel-eth0</strong></td>
<td>ipv6</td>
<td>2001:db8:0:8::3</td>
<td>ipv6-island</td>
<td>AMD PCNet FAST III</td>
<td>0800276A8FAE</td>
<td></td>
</tr>
<tr>
<td><strong>Ubuntu-ipv6-webserver-tunnel-eth1</strong></td>
<td>ipv4</td>
<td>192.168.56.22</td>
<td>vboxnet0</td>
<td>AMD PCNet FAST III</td>
<td>0800279866DE</td>
<td>Meant for ssh remote scripting</td>
</tr>
<tr>
<td><strong>Ubuntu-ipv4-webserver-DNS-eth0</strong></td>
<td>ipv4</td>
<td>12.0.0.2</td>
<td>ipv4-4</td>
<td>AMD PCNet FAST III</td>
<td>080027B5FE16</td>
<td></td>
</tr>
<tr>
<td><strong>Ubuntu-ipv4-webserver-DNS-eth1</strong></td>
<td>ipv4</td>
<td>192.168.56.16</td>
<td>vboxnet0</td>
<td>AMD PCNet FAST III</td>
<td>08002752A17F</td>
<td>Meant for ssh remote scripting</td>
</tr>
<tr>
<td><strong>Ubuntu-tunnel broker-eth0</strong></td>
<td>ipv4</td>
<td>192.168.56.23</td>
<td>vboxnet0</td>
<td>AMD PCNet FAST III</td>
<td>080027A13DAD</td>
<td></td>
</tr>
<tr>
<td><strong>Ubuntu-tunnel broker-eth1</strong></td>
<td>ipv4</td>
<td>12.0.0.3</td>
<td>ipv4-4</td>
<td>AMD PCNet FAST III</td>
<td>080027A94C97</td>
<td>Meant for ssh remote scripting</td>
</tr>
<tr>
<td><strong>Ubuntu-tunnel broker-eth2</strong></td>
<td>ipv6</td>
<td>2001:db8:0:8::1</td>
<td>ipv6-island</td>
<td>AMD PCNet FAST III</td>
<td>080027424CB3</td>
<td></td>
</tr>
<tr>
<td><strong>Ubuntu-ipv6-webserver-tunnel-auto-eth0</strong></td>
<td>ipv6</td>
<td>2001:db8:0:8::2</td>
<td>ipv6-island</td>
<td>AMD PCNet FAST III</td>
<td>080027830008</td>
<td></td>
</tr>
<tr>
<td><strong>Ubuntu-ipv6-webserver-tunnel-auto-eth1</strong></td>
<td>ipv4</td>
<td>192.168.56.25</td>
<td>vboxnet0</td>
<td>AMD PCNet FAST III</td>
<td>080027CDDF8D</td>
<td>Meant for ssh remote scripting</td>
</tr>
</tbody>
</table>
Table 3: Interfaces of the Network

<table>
<thead>
<tr>
<th>Machine</th>
<th>IPv6 gateway eth0</th>
<th>IPv6 gateway eth1</th>
<th>IPv6 gateway eth2</th>
<th>IPv4 gateway 1</th>
<th>IPv4 gateway 2</th>
<th>IPv4 webserver-DNS-eth0</th>
<th>IPv4 webserver-DNS-eth1</th>
</tr>
</thead>
<tbody>
<tr>
<td>UbuntuRouter</td>
<td>ipv6</td>
<td>ipv6</td>
<td>ipv4</td>
<td>ipv6</td>
<td>ipv6</td>
<td>ipv4</td>
<td>ipv6</td>
</tr>
<tr>
<td></td>
<td>2001:db8:0:2::3</td>
<td>2001:db8:0:3::1</td>
<td>192.168.56.17</td>
<td>2001:db8:0:3::2</td>
<td>192.168.56.18</td>
<td>2001:db8:0:4::2</td>
<td>2001:db8:0:4::2</td>
</tr>
<tr>
<td></td>
<td>ipv6-1</td>
<td>ipv6-2</td>
<td>vboxnet0</td>
<td>ipv6-4</td>
<td>vboxnet0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AMD PCNet FAST III</td>
<td>AMD PCNet FAST III</td>
<td>AMD PCNet FAST III</td>
<td>AMD PCNet FAST III</td>
<td>AMD PCNet FAST III</td>
<td>Meant for ssh remote scripting</td>
<td>Meant for ssh remote scripting</td>
</tr>
<tr>
<td></td>
<td>0800276938BA</td>
<td>0800270A0692</td>
<td>080027FB4B3F</td>
<td>080027A7CE18</td>
<td>080027FDEB71</td>
<td>0800273B8927</td>
<td>080027207794</td>
</tr>
</tbody>
</table>

3.7. Software installed

<table>
<thead>
<tr>
<th>Machine</th>
<th>SSH server</th>
<th>radvd</th>
<th>Apache web server</th>
<th>dnsmasq</th>
<th>tayga</th>
<th>OpenVPN</th>
<th>VSFTPD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu-1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ubuntu-2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ubuntu-3</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Windows-1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter-ipv4-gateway</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter-ipv4-1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter-ipv4-2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ubuntu-ipv4-webserver-DNS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Installed</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ubuntu-ipv4-webserver-tunnel</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ubuntu-tunnelbroker</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ubuntu-ipv4-webserver</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter-ipv6-gateway</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter-ipv6-1</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UbuntuRouter-ipv6-2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ubuntu-ipv6-webserver-DNS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Software installed
4. INSTALLATION, CONFIGURATION AND USAGE

4.1. Autoconfiguration

4.1.1. Software: Radvd

The radvd package is used by the routers to generate router advertisements with a specific prefix for global scope autoconfiguration.

4.1.1.1. Installation

With one of the interfaces connected to an external NAT for internet connection, run in the web server terminal:

```
apt-get install radvd
```

4.1.1.2. Configuration

Ip forwarding is enabled by going into the file located in:

```
/etc/sysctl.conf
```

Then, uncommenting the line:

```
net.ipv6.conf.default.forwarding=1
```

The configuration file is at `/etc/radvd.conf`. The file used for UbuntuRouter is the following:

```
interface eth1
{
  # If this option is on, the router advertisement will be sent
  AdvSendAdvert on;
  # The prefix to be advertised
  prefix 2001:db8:0:1::/64
  {
    # Indicates that this prefix can be used for on-link determination.
```
AdvOnLink on;
# Indicates that this prefix can be used for autonomous address
configuration as specified in RFC 4862.
    AdvAutonomous on;
};
;
radvd is restarted by using

/etc/init.d/radvd restart

4.2. Ping

4.2.1. Software: ping6

Ping6 is included in modern Linux distributions. It is a separate program for ping, which only
supports IPv4. The raw socket API for each of the versions is different so the entire program had
to be re-implemented.

4.2.1.1. Usage

For link local addresses the interface that is to be used must be explicitly specified.

    ping6 -I eth0 fe80::2e0:18ff:fe90:9205

For global addresses it works like regular ping.

    ping6 2001:db8:0:1::1

4.3. Webpage service over IPv6

4.3.1. Software: apache

Apache is a software application that is used by more than half of the http servers around the
world (Netcraft). It fully supports IPv6 without any needed change in the configuration.

4.3.1.1. Installation

With one of the interfaces connected to an external NAT for internet connection, run in the web
server terminal:

    apt-get install apache2
Apache version 1 and 2 are kept in separate repositories as there are not inter-compatible. Apache 1.3 is still used in commercial environments. We have chosen to use the newest version in order to use the updated documentation.

4.3.1.2. Configuration

Most of the default configurations will work on our case. A small change is necessary in the following file:

/etc/apache2/ports.conf

This will allow listening on all IP addresses. This could be a significant security risk but our network is an isolated simulation. Security is not a concern.

The listen parameter has been modified to only work on a specific port and allow any address:

NameVirtualHost *:80
Listen 80

The default html file to show on request is stored at /var/www/index.html. For the sake of simplicity we use a template to identify which page we are requesting.

<html><body><h1>Hello!</h1><p>You are looking at the page hosted on the ipv6 webserver, 2001:db8:0:4::2</p></body></html>

4.4. DNS name server

4.4.1. Software: dnsmasq

Many software packages exist for creating industrial level DNS solutions. However, for our current network we require something that is easier to install and configure; simple address resolution for a small network.

Dnsmasq is simple DNS software. It listens to DNS requests and resolves them according to a two column txt file, where the first column is the address to be returned and the second is the name to be resolved.
4.4.1.1. Installation

With one of the interfaces connected to an external NAT for internet connection, run in the web server terminal:

```
apt-get install dnsmasq-base
```

4.4.1.2. Configuration

The configuration file is hosted at:

```
/etc/dnsmasq.conf
```

We will use the following file, which redirects the dns configuration to another file for simplicity.

```
no-hosts
#this redirects all DNS requests to the /etc/hosts.dns files
addn-hosts=/etc/hosts.dns
```

We host the actual DNS resolution at a different file:

```
/etc/hosts.dns
```

On this file we can see the two columns of addresses and names used for the resolution:

```
2001:db8:0:4::2 www.webserver6.com
2001:db8:0:7::2 www.webserververtunnel.com
2001:db8:1:ffff::12.0.0.2 www.webserver4.com
```

On the client side we need to configure the DNS server to be used. This can be achieved by modifying the following file:

```
/etc/resolv.conf file.
```

We add the following lines to the files:

```
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
# DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
#set the DNS server
nameserver 2001:db8:0:4::2
```
As the file very well warns, the changes in this file are overwritten on reboot. We can fix this by adding a line to:

```
/etc/rc.local
```

The following line is added:

```
#write the DNS address at boot
echo "nameserver 2001:db8:0:4::2" >> /etc/resolv.conf
```

4.5. FTP server

4.5.1. Software: vsftpd

“VSFTPD stands for "Very Secure FTP Daemon" is a GPL licensed FTP server for UNIX systems (...)vsftpd is the default FTP server in the Ubuntu, CentOS, Fedora, NimbleX, Slackware and RHEL Linux distributions. It is secure and extremely fast. It is stable.” (Canonical)

4.5.1.1.1. Installation

With one of the interfaces connected to an external NAT for internet connection, run in the web server terminal:

```
apt-get install vsftp
```

4.5.1.1.2. Configuration

The configuration file is hosted at:

```
/etc/vsftp.conf
```

```
# Run standalone with IPv6
# Listen on an IPv6 socket instead of an IPv4 one. This parameter and the listen parameter are mutually exclusive.
listen_ipv6=YES
#
# Allow anonymous FTP. This is a security risk on a public server but security is not a concern in our network.
anonymous_enable=YES
#
```
# Activate directory messages - messages given to remote users when they go into a certain directory.
dirmessage_enable=YES
#
# vsftpd will display directory listings with the time in the local time zone. The default is to display GMT.
use_localtime=YES
#
# Activate logging of uploads/downloads.
xferlog_enable=YES
#
# Make sure PORT transfer connections originate from port 20(ftp-data).
connect_from_port_20=YES
#
# What the login banner string will display
ftpd_banner=Welcome the PFC test
#
# This string is the name of the PAM service vsftpd will use.
pam_service_name=vsftpd
#
# The default directory for anonymous connections
anon_root=/home/ricardo

4.6. Manual tunnel

The latest version of the Linux kernel (3.13) includes functionality that allows easily making different types of tunnels. A manual tunnel can easily be made with the included ip command and added as a virtual interface to which we can reroute IPv6 traffic using the route command.

4.6.1. Software: ip

“The ip command is used to assign an address to a network interface and/or configure network interface parameters on Linux operating systems.”
(NixCraft Community)

4.6.1.1. Usage

The ip command can be very complex and its full usage is beyond the scope of this work. However, its usage for some specific cases can be explained. For a full reference, check the manual source (NixCraft Community).
The `ip` command is followed by the object on which the command is going to act or get information about. The following objects can be used according to (Brown):

<table>
<thead>
<tr>
<th>Object</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>network device</td>
</tr>
<tr>
<td>address</td>
<td>protocol (IP or IPv6) address on a device</td>
</tr>
<tr>
<td>neighbour</td>
<td>ARP or NDISC cache entry</td>
</tr>
<tr>
<td>route</td>
<td>routing table entry</td>
</tr>
<tr>
<td>rule</td>
<td>rule in routing policy database</td>
</tr>
<tr>
<td>maddress</td>
<td>multicast address</td>
</tr>
<tr>
<td>mroute</td>
<td>multicast routing cache entry</td>
</tr>
<tr>
<td>tunnel</td>
<td>tunnel over IP</td>
</tr>
</tbody>
</table>

Table 5: `ip` command objects

### 4.6.1.1. Tunnel creation

To create or manage a tunnel we use the `tunnel` object, so the first part of our command is:

```
  ip tunnel
```

Next comes the action we are going to take with a tunnel. We can *add, change* and *delete* a tunnel using these keywords. In our case we want to add a tunnel, so we put the `add` keyword followed by the tunnels name:

```
  ip tunnel add NAME
```

Then, the type of tunnel needs to be specified from 4 options:

- **GRE**, for IPv4 or IPv6 encapsulated over IPv4 using an especial GRE header.
- **ISATAP**, for ISATAP tunnels (which will be used on section 4.7).
- **SIT**, for is IPv6 over IPv4.

For a more detailed explanation of IPIP and GRE tunnels, check source (Stretch).

So the command now looks like this:

```
  ip tunnel add NAME mode MODE
```
Using the `local` and `remote` keywords we need to specify the endpoints of the tunnel, with local being the address associated with the local interface and remote being the address of the remote tunnel endpoint.

```
ip tunnel add NAME mode MODE local ADDRESS remote ADDRESS
```

Finally, we can use the `dev` keyword too route all the tunnel’s packets through a specific interface by specifying the interface’s name.

```
ip tunnel add NAME mode MODE local ADDRESS remote ADDRESS dev INTERFACENAME
```

### 4.6.1.1.2. Network device management

To manage devices the `link` keyword is utilized.

```
ip link
```

By adding the `set` keyword after `link` we can change a device’s attributes.

```
ip link set
```

Using the `dev` keyword we can specify the device to be modified by name.

```
ip link set dev NAME
```

And finally, using the `up/down` keywords we can put the interface up or down.

```
ip link set dev NAME up
```

### 4.6.1.1.3. Address management

To manage addresses we use the `address` or `addr` keywords.

```
ipaddr
```

By using the `add` keyword we can specify an address to add.

```
ipaddr add ADDRESS
```

With the `dev` keyword we can specify the device to associate with the address by name.

```
ipaddr add ADDRESS dev NAME
```
4.6.2. Software: route

Router is a command that allows for the modification of the internal routing table of the device.

4.6.2.1. Usage

The route command can be very complex and its full usage is beyond the scope of this work. However, its usage for some specific cases can be explained. For a full reference, check the manual source (Haas).

After the command the \texttt{-A} option can be used to specify which protocol will be used. In our case, we specify \texttt{inet6} which means we are going to modify or request information regarding the ipv6 tables. IPv4 and IPv6 tablets are separated by the double stack.

\texttt{route -A inet6}

The \texttt{add/del} keywords can be used to modify the table. If not present the route command will simply show the table. After the add keyword and address (network) and netmask can be directly specified.

\texttt{route -A inet6 add ADDRESS}

The \texttt{dev} keyword can be used to specify the name of an interface where packets directed to the specified network will be routed to.

\texttt{route -A inet6 add ADDRESS dev NAME}

4.6.3. Configuration

All commands necessary for the manual tunnel are condensed on a script that is run on both ends of the tunnel (changing the necessary addresses). The script can be added at \texttt{/etc/rc.local} for it to be run on startup.

\texttt{
# create the tunnel device interface called v6 and specify the local and remote endpoints of the tunnel
ip tunnel add v6 mode sit local 10.0.0.3 remote 13.0.0.2 dev eth2
# put the interface up
ip link set dev v6 up
# give the tunnel interface an IPv6 address
ipaddr add 2001:db8:0:9::1/64 dev v6
}
#add a route into the routing table, so all traffic to the ipv6island goes through the tunnel

route -A inet6 add 2001:db8:0:8::/64 dev v6

4.7.   ISATAP tunnel

4.7.1.   Software: isatap

Isatap support was added to the Linux kernel at version 2.6.25 which allows different commands to create and manage automatic isatap tunnels. Isatapd is software for Linux created to facilitate the creation of said tunnels from the client side with a single command.

4.7.1.1.   Installation

With one of the interfaces connected to an external NAT for internet connection, run in the web server terminal:

```
apt-get install isatapd
```

4.7.1.2.   Configuration

At the client side only a command is needed with the name of the virtual interface and the ipv4 address of the tunnel’s endpoint. The isatapd protocol takes care of the rest of the tunnel’s creation.

```
#12.0.0.3 is the address of the isatap server
isatapd -n isatap -r 12.0.0.3 -v
```

At the server side we can create the endpoint (which we are using the Ubuntu-tunnelbroker for, in order to save up space) using the ip command in isatap mode and giving it an unused ipv6 address. The commands necessary have been added in a script at:

```
/root/startSATAP
```

```
#create the tunnel device for the isatap server, with 12.0.0.3 as the local address
ip tunnel add name is0 mode isatap local 12.0.0.3
#put up the new interface we have created
ip link set is0 up
#set an ipv6 address to the ISATAP interface
ip -6 addradd 2001:db8:1234:5:0:5efe:a00:1/64 dev is0
```

The isatap client assigns the global address using autoconfiguration, so we need to make sure the endpoint is running a properly configured radvd and sending router solicitations. The prefix
included in router solicitations must be the same as the prefix of the IPv6 address the isatap endpoint is using. The file used in the example is:

/etc/radvd.conf

interface is0
{
    AdvSendAdvert on;
    UnicastOnly on;
    #this is the prefix that will determine the autoconfigured address at the client endpoint
    prefix 2001:db8:1234:5::/64
    {
        # Indicates that this prefix can be used for on-link determination.
        AdvOnLink on;
        # Indicates that this prefix can be used for autonomous address configuration as specified in RFC 4862.
        AdvAutonomous on;
    }
};

4.8. Tunnel Broker

4.8.1. Software: OpenVPN

OpenVPN is software that allows the creation of a Private Network between two or more machines along a public unencrypted network and controls access to said network using signed certificates, which matches the requirements needed for our tunnel broker experiment. It has a high level of customization which allows the creation of a tunnel on the VPN.

4.8.1.1. Installation

With one of the interfaces connected to an external NAT for internet connection, run in the web server terminal:

    sudo apt-get install openvpn

4.8.1.2. Configuration – Server

The openVPN configuration files are stored:

    /etc/openvpn
If a server.conf file is present openVPN will automatically run in server mode. The server.conf file reads:

# Local IP address for the server to listen on.
local 15.0.0.2

#openVPN requires a security level to execute, where 0 allows no calling of external programs, 1 allows the call of executable, 2 allows user defined scripts and 3 allows to pass passwords to scripts using environmental variables. We set in 2 in order to execute external scripts for connection and disconnection.

script-security 2

# the port to listen on
port 56789

# udp is the recommended transport method to use
protoudp

# the name of the virtual device to mount
devtun

# openVPN requires a series of keys in order to authenticate and encrypt correctly, which are specified here.

#openVPN requires a certificate file from the certificate authority, which it will use to certify the signature on the keys. This is the path to that file

cert /etc/openvpn/easy-rsa/2.0/keys/ca.crt

cert /etc/openvpn/easy-rsa/2.0/keys/server.crt

#As a server, the machine requires a server certificate signed by the trusted Certificate Authority. This seems redundant in this case, but often Certificate Authorities and VPN servers are separate entities

cert /etc/openvpn/easy-rsa/2.0/keys/server.crt

#This is the locally generated server secret key. This file is kept secret.
key /etc/openvpn/easy-rsa/2.0/keys/server.

#openVPN also requires parameters for the Diffie Hellman key exchange. Even though we do not exchange keys using the procedure (Because key security is beyond the scope of this project), openVPN requires this parameters in order to run

dh /etc/openvpn/easy-rsa/2.0/keys/dh2048.pem

# the ip range that OpenVPN will use for its static ipv4 addresses. This space was selected to be different from any other address space used by other devices in the network to avoid problems.

server 10.18.0.0 255.255.255.0

#use Lempel-Viev-Oberhumer compression

comp-lzo

#set up the maximum number of clients connected to the tunnel broker to 10

max-clients 10

# The keepalive directive causes ping-like messages to be sent back and forth over the link so that each side knows when the other side has gone down. Ping every 18 seconds and assume that the remote peer is down if no ping received during an 83 second time period.

keepalive 18 83

#If the connection is restarted (Because of lack of activity for example) Do not reread the keys from the hard drive.

persist-key

#Do not run the up/down scripts and do not reopen any tunnels when the connection is restarted.

persist-tun
#write status of openVPN to the specified file after several seconds.

status openvpn-status.log

#Select a middle level of verbosity. The server will output data for every packet sent for monitoring.

verb 5

# The scripts that will run every time an authorized client is connected or disconnected

client-connect /etc/openvpn/client-connect.sh  
client-disconnect /etc/openvpn/client-disconnect.sh

The script that is run every time a client connects and that creates the automatic tunnel is at /etc/openvpn/client-connect.sh.

# This is a script that is run each time a remote client connects to this openvpn server. It will setup the IPv6 tunnel depending on the ip address that was given to the client

#A variable stores the base network prefix for the IPv6 addresses

BASERANGE="2001:f5a:53"

# We pipe the contents of a variable created by openVPN that stores the addresses of the clients and check how many fields (connected clients) it has using the awk command and store the result in a variable.

V6NET=$(echo ${ifconfig_pool_remote_ip} | awk -F. '{print $NF}')

#We use this variable to name each tunnel differently

SITID="sit${V6NET}"

# The ip command is used to setup the tunnel, using the openVPN variables to get the local and remote addresses and our new variable for the name of the tunnel. Afterwards, the interface is put up
This script is run every time a client disconnects. It takes down the automatic tunnel and is located at /etc/openvpn/client_disconnect.sh.

# This is a script that is run each time a remote client connects to this openvpn server. It will take down the IPv6 tunnel depending on the ip address that was given to the client

# First, a variable stores the base network prefix for the IPv6 addresses

BASERANGE="2001:f5a:53"

# Then, we pipe the contents of a variable created by openVPN that stores the addresses of the clients and check how many fields (connected clients) it has using the awk command and store the result in a variable.

V6NET=$(echo ${ifconfig_pool_remote_ip} | awk -F. '{print $NF}')

# We use this variable to get the name of the current tunnel

SITID="sit${V6NET}"
# Using the ip command we remove the ip address (generated by the previous script) from the virtual interface used by the tunnel

```bash
sudo ip -6 addr del ${BASERANGE}:${V6NET}::1/64 dev ${SITID}
```

# Using the ip command we take down the interface and remove the tunnel, using the variables to get the name in the same way as the previous script did

```bash
sudo ip link set dev ${SITID} down
sudo ip tunnel del ${SITID} mode sit ttl 255 remote
${ifconfig_pool_remote_ip} local ${ifconfig_local}
```

# Finally, save the openVPN variables in a log

```bash
echo "${script_type} client_ip:${trusted_ip} \ 
common_name:${common_name} local_ip:${ifconfig_local} \ 
remote_ip:${ifconfig_pool_remote_ip} sit:${SITID} \ 
ipv6net:${V6NET} duration:${time_duration} \ 
received:${bytes_received} sent:${bytes_sent}" | /usr/bin/logger -t ovpn
```

4.8.1.3. Configuration – Client

If a `client.conf` file is present openVPN will automatically run in client mode. The `client.conf` file reads:

```bash
# Specify that we are a client and that we will be pulling certain config file directives from the server.
client

# the name of the virtual device to mount
devtun

# udp is the recommended transport method to use
protoudp

# openVPN requires a security level to execute, where 0 allows no calling of external programs, 1 allows the call of executable, 2 allows user defined scripts and 3 allows to pass passwords to scripts using environmental variables. We set in 2 in order to execute external scripts for connection and disconnection.
```
script-security 2

#remote address and port to attempt a connection
remote 15.0.0.2 56789

# Most clients don't need to bind to a specific local port number.
nobind

# The keepalive directive causes ping-like messages to be sent back and forth over the link so that each side knows when the other side has gone down. Ping every 27 seconds and assume that the remote peer is down if no ping received during a 79 second time period.
keepalive 27 79

#If the connection is restarted (Because of lack of activity for example) Do not reread the keys from the hard drive.
persist-key

#Do not run the up/down scripts and do not reopen any tunnels when the connection is restarted.
persist-tun

#openVPN requires a certificate file from the certificate authority, which it will use to certify the signature on the keys. This is the path to that file
cert /etc/openvpn/ca.crt

directorio de servidor /etc/openvpn/client1.crt

directorio de clave /etc/openvpn/client1.key
#use Lempel-Viev-Oberhumer compression

comp-lzo

#Set a low level of verbosity

verb 3

# Path to the scripts that will run every time the tunnel is put up or down

up /etc/openvpn/up.sh
down /etc/openvpn/down.sh

# When client disconnects it will tell the server to remove the ipv6 tunnel the client was using

explicit-exit-notify

The script that is run every time this client connects and that creates the automatic tunnel is at /etc/openvpn/up.sh.

#!/bin/bash

# script that is run on the client when it creates a tunnel to the remote OpenVPN server

#A variable stores the base network prefix for the IPv6 addresses

IPV6BASE=2001:f5a:53

# We pipe the contents of a variable created by openVPN that stores the addresses of the clients and check how many fields (connected clients) it has using the awk command and store the result in a variable.

V6NET=$(echo ${ifconfig_local} | awk -F. '{print $NF}')

# We create a tunnel called sit1 to the (manually set) remote address of the tunnel broker. The local address is generated from the openVPN variable.

ip tunnel add sit1 mode sit ttl 255 remote 10.18.0.1 local ${ifconfig_local}
#Put up the interface

ip link set dev sit1 up

#Generate an address based on the variables and assign it to the tunnel interface

ip -6 addr add ${IPV6BASE}:${V6NET}::2/64 dev sit1

#Finally we route all traffic created for the tunnel to the interface

ip route add ::/0 via ${IPV6BASE}:${V6NET}::1

#We exit with no errors

exit 0

This script is run every time this client disconnects and it takes down the tunnel. It is at /etc/openvpn/down.sh.

#!/bin/bash

#A variable stores the base network prefix for the IPv6 addresses

IPV6BASE=2001:f5a:53

# We pipe the contents of a variable created by openVPN containing the addresses of the clients and check how many fields (connected clients) it has using the awk command, storing the result in a variable.

V6NET=$(echo ${ifconfig_local} | awk -F. '{print $NF}')

#We delete the address generated for the virtual interface

sudo ip -6 addr del ${IPV6BASE}:${V6NET}:2/64 dev sit1

#We put down the interface

sudo ip link set dev sit1 down

#The tunnel is deleted

sudo ip tunnel del sit1 mode sit ttl 255 remote 10.18.0.1 local
${ifconfig_local}

#The route is removed from the table
sudo iproute del ::/0 via ${IPV6BASE}:${V6NET}:1

#We exit with no errors
exit 0

4.8.2. Software: OpenSSL

OpenSSL is an implementation of SSL and TLS for certificate creation. It is used to cover the authentication requirements of the tunnel broker. In our current configuration the tunnel broker will accept connections from anyone with a certificate correctly signed by the certificate authority running in the tunnel broker itself (using openSSL). To aid in the process of certificate creation we will use a set of scripts included in the examples of openVPN known as easy-rsa, located at /usr/share/doc/openvpn/examples/easy-rsa/2.0. OpenSSL is included in the latest distributions of Ubuntu.

4.8.2.1. Configuration

Using easy-rsa the creation of certificates is very easy. A script must be run in the tunnel broker and the certificates must be copied to the pertinent machines where openVPN will use them in the configuration file.

To create the certificate authority:

```
source ./vars
./clean-all
./build-ca
```

For the server certificates:

```
./build-key-server server
```

For the necessary Diffie Hellman parameters

```
./build-dh
```

For each client:

```
./build-key client-name
```
4.9. NAT64

4.9.1. Software: tayga

Tayga is software for Linux which allows the usage of the NAT64 protocol for network translation, allowing ipv6 only hosts to reach ipv4 addresses by using an especial address prefix and adding the ipv4 address as the last 32 bits of the address. The router running tayga will then take care of the translation between protocols.

4.9.1.1. Installation

With one of the interfaces connected to an external NAT for internet connection, run:

```
apt-get install tayga
```

4.9.1.2. Configuration

The tayga configuration file is at `/etc/tayga.conf` and allows us to configure the IPv4 and IPv6 addresses the virtual tayga interface will use, the prefix the router will take as a request for translation and the address pool used to map the IPv4 space and the file where the translations will be stored.

```
#the name of the virtual interface that will do the translation
tun-device nat64

#the ipv4 address assigned to the virtual translation interface
ipv4-addr 10.0.0.4
#the ipv6 address assigned to the virtual translation interface
ipv6-addr 2001:db8:1::3
#the especial prefix that will translate ipv4 addresses to ipv6
prefix 2001:db8:1:ffff::/96
#the pool of addresses used to map ipv6 hosts. This will not be used in our scenario, but needs to be set nonetheless
dynamic-pool 192.169.0.0/24
#the file where the pool mapping will be stored. A necessity, even though we will not use it
data-dir /var/spool/tayga
```

In order to get tayga running we must create the virtual interface and assign the selected addresses to it, as well as add routes to the two sides of the network through the interface. This has all been added in a script on the server at `/root/taygascript`

```
tayga --mktun
#put up the newly created interface
ip link set nat64 up
```
# give the nat64 interface an ipv4 address, which needs to be the same as the ipv4 endpoint address
ipaddr add 10.0.0.3 dev nat64
# give the nat64 interface an ipv6 address, which needs to be the same as the ipv6 endpoint address
ipaddr add 2001:db8:0:1::1 dev nat64
# Add a router to the table for all address in the specified ipv4 pool the nat interface
ip route add 192.169.0.0/24 dev nat64
# Add a router to the table for all address in the specified ipv4 pool the nat interface
ip route add 2001:db8:1:ffff::/96 dev nat64
# we start tayga to do the translation
/etc/init.d/tayga start

# Several rules need to be added in order for the translation to work as intended

# we add a route to IP tables, so packets going out of eth2 on POSTROUTING are masqueraded, in other word, have the ip headers modified
iptables -t nat -A POSTROUTING -o eth2 -j MASQUERADE
# next, we append to the Forward process a rule where packets entering from eth2 and exiting from nat64 must be accepted for specific states
iptables -A FORWARD -i eth2 -o nat64 -m state --state RELATED,ESTABLISHED -j ACCEPT
# Finally, we add a rule to accept packets being forwarded between nat64 and eth2
iptables -A FORWARD -i nat64 -o eth2 -j ACCEPT

# we make sure the ip forwarding is active
echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
echo 1 > /proc/sys/net/ipv6/conf/all/forwarding

4.10. Kernel variables

Most of Window’s registry options for IPv6 are not configurable so changing kernel variables are not an option (Microsoft), but variables can be changed in Linux using commands.

4.10.1. Software: sysctl

Sysctl is a command included in most Linux distributions that allows changing kernel parameters at runtime without the need of recompiling.

To change any of the system variables:
sysctl -w VARIABLE=NEW_VALUE

This will be used to change variables related to network processes, such as the number of DAD requests or the time waited between certain steps.

sysctl -w net.ipv6.conf.eth1.router_solicitation_delay=4
sysctl -w net.ipv6.conf.eth1.dad_transmits=1
5. TRAFFIC ANALYSIS

5.1. Autoconfiguration

Objective: Check how the auto configuration mechanisms for IPv6 work and the packets involved for local and global scope addresses.

5.1.1. Necessary machines:
- At least two of the clients (Ubuntu-1, Ubuntu-2 or Windows-1 if available).
- UbuntuRouter (Turned off at first).
- Ubuntu-ipv6-webserver-DNS and UbuntuRouter-ipv6-1 for the last case.

5.1.2. Autoconfiguration without router

We turn on both Ubuntu-1 and Ubuntu-2, and we run a Wireshark capture at the eth1 interface of Ubuntu-1, filtering only for packets with no address (Unassigned interfaces) or packets whose origin or destination address is fe80::a00:27ff:fe0b:4677 (The address Ubuntu-2 is expected to take by its MAC address 0800270B4677 through the procedure explained in section 2.1.6). The command used to start the remote wireshark capture is:

```bash
ssh root@192.168.56.11 'tshark -i eth1 -f "src host fe80::a00:27ff:fe0b:4677 or dst host fe80::a0:27ff:fe0b:4677" -w -' | wireshark -k -i -
```
The network service is reset on Ubuntu-2 in order for it to go through auto configuration and observe the result in wireshark by using:

```
ssh root@192.168.56.12 '/etc/init.d/networking restart'
```

This capture is saved as `ubuntu-2-no router`
According to RFC 4862:

“Before sending a Neighbor Solicitation, an interface MUST join the all-nodes multicast address and the solicited-node multicast address of the tentative address. The former ensures that the node receives Neighbor Advertisements from other nodes already using the address; the latter ensures that two nodes attempting to use the same address simultaneously should detect each other's presence.”

(IETF, 2007)

This is the process we are observing here. The device has a MAC address of 08:00:27:0B:46:77, which leads to a solicited node address ff02::1:ff0b:4677 as explained in section 2.1.4.1.

On the ICMPv6 header the type is set to 143, which corresponds to Multicast Listener Report messages of the Multicast Listener Discovery version 2 (MLDv2) protocol. (IEEE)

Every multicast address has its own internal format, as explained in RFC 3810, with the first field specifying the type of record of the address. In this case it is set to 4, changed to exclude.
“CHANGE_TO_EXCLUDE_MODE - indicates that the interface has changed to EXCLUDE filter mode for the specified multicast address. The Source Address [i] fields in this Multicast Address Record contain the interface's new source list for the specified multicast address, if it is non-empty.” (IETF, 2004)

The interface is excluding the specified address from its filter, and so it will receive its multicast packets from now on.

The next package is marked as a Neighbor Solicitation and is part of the Duplicate Address Detection process as described in section 2.1.6.1.1. Before assigning the address the interface checks if any other host in the local scope has the same address.

After DAD is over the link local address is assigned and the autoconfiguration process continues. The interface waits a set amount of time (1 second in this instance), before continuing.
After one second passes the interface looks for a global address by sending router solicitations with the target address ff02::2 which is reserved for all local-scope routers.

The ICMP headers is set to 133 (Router Solicitation) and the ICMP options holds the rest of the details. As explain in RFC 4861, the ICMP options must include the source link-layer address (MAC) if the source is not the unspecified address (::). This is why the source link-layer address is announced here.
The message is repeated three times (as controlled by an internal kernel variable). If no answer is received after these requests, no global address is dynamically assigned and the interface is unable to use global scope IPv6.

<table>
<thead>
<tr>
<th>57.468118</th>
<th>fe80::a00:27ff:fe0b:4677</th>
<th>ff02::16</th>
<th>ICMPv6</th>
<th>90 Multicast Listener Report Message v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet II, Src: CadmusCo_0b:46:77 (08:00:27:00:46:77), Dst: IPv6mcast_00:00:00:16 (33:33:00:00:00:16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internet Protocol Version 6, Src: fe80::a00:27ff:fe0b:4677 (fe80::a00:27ff:fe0b:4677), Dst: ff02::16 (ff02::16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internet CONTROL MESSAGE PROTOCOL v6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type: Multicast Listener Report Message v2 (143)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code: 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checksum: 0xb404 [correct]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved: 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Multicast Address Records: 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multicast Address Record Changed to exclude: ff02::1:ff0b:4677</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record Type: Changed to exclude (4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUX Data Len: 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Sources: 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multicast Address: ff02::1:ff0b:4677 (ff02::1:ff0b:4677)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8: 5th packet of the autoconfiguration (no router)

The 5th packet of the capture, which was not shown the previous image, was another multicast listener report as described before. The ICMP payload holds exactly the same information as the previous package.

After auto configuration is done Ubuntu-2’s Routing Table is the following.
This table only shows the default empty addresses as well as two link local addresses, `fe80::a00:27ff:fe0b:4677` from which we just observed the auto configuration process. No default global route is assigned.
5.1.3. Autoconfiguration with router

We turn on Ubuntu-1, Ubuntu-2 and UbuntuRouter and we run a Wireshark capture at the eth1 interface of Ubuntu-1, filtering only for packets with no address (Unassigned interfaces), packets whose origin or destination address is `fe80::a00:27ff:fe0b:4677` (The address Ubuntu-2 is expected to take by its MAC address `0800270b4677` through the procedure explained in section 2.1.6) or packets whose origin or destination address is `fe90::a00:27ff:fe8c:66b6` (The address UbuntuRouter is expected to take by its MAC address `0800278c66b6` through the same procedure). The command used to start the remote wireshark capture is:

```bash
ssh root@192.168.56.11 'tshark -i eth1 -f "src host fe80::a00:27ff:fe0b:4677 or dst host fe80::a0:27ff:fe0b:4677 or src host :: or dst host :: or src host fe80::a00:27ff:fe8c:66b6" -w -' | wireshark -k -i -
```

![Diagram of Autoconfiguration Capture (With Router)](image)

Figure 10: Diagram of Autoconfiguration Capture (With Router)
The network service is reset on Ubuntu-2 in order for it to go through auto configuration and observe the result in wireshark by using:

```
ssh root@192.168.56.12 'tshark/etc/init.d/networking restart'
```

This capture is saved as **ubuntu-2-with router**.
The first packet of the capture belongs to the router. When the capture was started the router was finishing its own autoconfiguration process and the DAD neighbor solicitation for the link local address was captured, since the filter was made to include packets coming from the router’s link local address. This packet does not belong to the process that is under study in this section.

These first 3 packets follow the same structure observed in the previous section. The interface enters the multicast group of its link local address, goes through DAD for its link local address and does a Router Solicitation to acquire a global address. For a detailed explanation of this packages and its options consult section 5.1.2.
This time the router solicitation is answered by a router advertisement from the router using the router’s link local address (fe80::a00:27ff:fe8c:66b6).

The router advertisement is an ICMP message with the type field marked as 134. As specified in RFC 4861 the ICMP package must include three additional fields.

- **Router lifetime**, how long can the interface use the router as a default router, set as 1800 seconds.
• **Reachable Time**, tells hosts how long, in milliseconds, they should consider a neighbor to be reachable after they have received reachability confirmation, set as 0 seconds which means unspecified.

• **Retrans time**, the amount of time in milliseconds that a host should wait before retransmitting Neighbor Solicitation messages, set as 0 seconds which means unspecified.

This is followed by two ICMP option areas.

The first ICMP Option payload is meant for the prefix information as explain in RFC 4862. It includes the prefix and a valid lifetime; which is the maximum router lifetime to be assigned, and a preferred lifetime; which has the same role as the router lifetime on the ICMP header.

The advertised prefix is 2001:db8:0:1::, as specified in the configuration in section 4.1.1.2.
Before assigning any global address the interface must join the solicited-node multicast address for the addresses it intends to take, as previously explained in section 5.1.2. On this instance there are two solicited-node addresses, since Ubuntu assigns two global addresses per interface. The first address is generated in the same manner as the link local address (through the MAC address) but using the prefix specified by the router, and results in `2001:db8:0:1:a00:27ff:fe0b:4677`. The solicited node multicast address that corresponds to this unicast address is `ff02::1:ff0b:4677`.

Figure 14: Multicast Listener Report for Global Addresses (1)
The second address is generated by Ubuntu randomly and changes over time. This address is used for all outgoing global connections. The logic for this implementation is that using an address generated through the MAC address could expose the MAC to all external entities, which could become a security risk. This temporal global address is \textit{2001:db8:0:1:c515:9e3f:1d34:2751} for this particular session.

The solicited node multicast address that corresponds to this unicast address is \textit{ff02::1:ff34:2751}.

This Multicast Listener Report is joining the interface to both multicast addresses.

For every address assigned to the interface DAD must be performed to verify the uniqueness. Here, the interface sends a neighbor solicitation for the random address \textit{(2001:db8:0:1:c515:9e3f:1d34:2751)} directed at the solicited node multicast address associated with the random unicast global address \textit{(ff02::1:ff34:2751)}. By default, the interface will wait 1 second before assigning the address and using it for any other process.
The interface also performs DAD for the other global address to verify its uniqueness. Here, the interface generates a neighbor solicitation to check for duplicates for the MAC generated address (2001:db8:0:a00:27ff:fe0b:4677) directed at the solicited node multicast address associated with that unicast address (ff02::1:ff0b:4677).
The final packet is another Multicast Listener Report with exactly the same components as the last one. The interface sends two Multicast Listener Report Messages every time it is going to add a new address.
This is the final routing table. When compared to the table displayed in **Figure 9: Routing table, Ubuntu-2, no router** the final table includes the two global addresses and the router’s link local address (fe80::a00:27ff:fe8c:66bc) as the default route.
5.1.4. Autoconfiguration on Window with router

We turn on Ubuntu-1, Ubuntu-2, Windows-1 and UbuntuRouter. Following the procedure of previous captures, we need to apply filters for the unassigned address, the router address and the address that Windows will assign.

Window’s default behavior is to generate a link local IPv6 addresses and two global addresses using a pseudo random algorithm in order to hide the interface’s MAC. The Link Local address is generated with an infinite Valid Lifetime, which means that the Link Local address will remain the same through reboots until the user decides to generate it again (Hughes). We can take note of the address for the filter.

![Figure 19: Window’s Addresses for current session](image)

We run the capture using this link local address as the filter.

```
ssh root@192.168.56.11 'tshark -i eth1 -f "src host fe80::160:cc40:c7b9:7181 or dst host fe80::160:cc40:c7b9:7181or src host :: or dst host :: or src host fe80::a00:27ff:fe8c:66b6" -w -' | wireshark -k -i -
```

Windows-1 is rebooted to observe the autoconfiguration process.
Figure 20: Diagram of Autoconfiguration Capture for Windows (With Router)
Windows follows the same basic procedure as Linux for autoconfiguration. Before assigning the randomly generated address, the interface does DAD to check its uniqueness. Since the address space is very large, the chance of choosing an occupied link-local address is extremely small.
The second packet is the router solicitation for the global address.

“With a randomly derived interface ID, the chance of duplicating the link-local address is very small. Therefore, computers running Windows Vista or Windows Server 2008 (or newer) do not wait for DAD to complete before sending Router Solicitation messages using their derived link-local addresses. This is known as optimistic DAD; router discovery and DAD are performed in parallel to save time during the interface initialization process.” (Davies)

Because of optimistic DAD there is very little time between DAD and the first router solicitation.

The structure of the solicitation has no difference with what was observed from Ubuntu.

![Multicast Listener Report](image)

Figure 23: Windows Multicast Listener Report 1
Almost simultaneously the Multicast Listener Report is sent to subscribe the solicited node address (ff02::1:ffb9:7181) associated with the random address (fe80::160:cc40:c7b9:7181), with the same structure explained before. This packet is captured after the DAD and router solicitation with very small time in between so all processes happen simultaneously.
After receiving the router solicitation the router advertisement is sent from the router, with the same structure observed in the previous capture.
Once the router solicitation is received Windows assigns two global addresses, one based on the link local address and a new random address, both using the prefix specified by the router.

- If a router advertisement is received, create tentative public addresses corresponding to the global or unique local address prefixes in the message based on the random interface ID derived in Step 1 (Link Local address), and perform DAD.
- Create tentative temporary addresses corresponding to the global or unique local address prefixes in the message using new randomly derived random interface IDs, and perform DAD.

(Davies)

The interface joins the solicited node multicast address (**ff02::1:ff62:4624**) for the tentative temporary address (**2001:db8:0:1:194b:2732:4862:4624**).
DAD is performed for the first global address, the one based on the Link Local address. The same structure as previous DAD packets is observed. The unassigned address is used for all DAD requests, even though it is directed to a link-local scope address.

At almost the same time, DAD is performed for the new random global address (2001:db8:0:1:194b:2732:4862:4624). This is a temporary address (as defined in RFC 3041) and it is used to provide a level of anonymity to client-initiated communications.
A second multicast Listener Report is sent, verifying the subscription to the solicited node addresses of the two global addresses.
The final routing table includes the link local address, as well as the global addresses and the router’s link local address as the default route for all other communications.
5.1.6 Autoconfiguration with static Global address

We turn on UbuntuRouter-ipv6-1 and Ubuntu-ipv6-webserver-DNS. We run the capture on the Router while applying filters for the unassigned addresses, the local and global addresses for the webserver (fe80::a00:27ff:fe3b:8927 and 2001:db8:0:4::2) and the local and global addresses for the router (fe80::a00:27ff:fe11:4b63 and 2001:db8:0:4::1).

ssh root@192.168.56.18 'tshark -i eth1 -f "(src host fe80::a00:27ff:fe3b:8927 or dst host fe80::a00:27ff:fe3b:8927 or src host :: or dst host :: or src host 2001:db8:0:4::1 or dst 2001:db8:0:4::1 or src host 2001:db8:0:4::2 or dst 2001:db8:0:4::2 or src host fe80::a00:27ff:fe11:4b63 or dst fe80::a00:27ff:fe11:4b63)" -w -' | wireshark -k -i -

The network manager is restarted on Ubuntu-ipv6-webserver-DNS.

ssh root@192.168.56.19 '/etc/init.d/networking restart'

The capture is saved as static address autoconfiguration.
Figure 30: Diagram of static address autoconfiguration
The IPv6 webserver has an important key difference to previous analyzed setups. The global address is set statically in the configuration and is not acquired from a router. We aim to see how this affects the autoconfiguration process.

```
auto eth0
iface eth0 inet6 static
address 2001:db8:0:4::2
netmask 64
gateway 2001:db8:0:4::1
```

Figure 31: Manual interface configuration for the ipv6 webserver

**Figure 32:** First Multicast Listener Report Message of Static Global Autoconfiguration
The first packet is a multicast listener report subscribing the interface to the solicited node address for the link local address generated from the MAC (fe80::a00:27ff:fe3b:8927, with a solicited node address of ff02::1:ff3b:8927) and the global statically set address (2001:db8:0:4::2, with a solicited node address of ff02::1:ff00:2).

Figure 33: DAD for link local address on Static Address Capture

The interface performs DAD on the link local address.

Figure 34: DAD for global address on Static Address Capture
The first global address is directly assigned from the manual configuration. Duplicate Address detection is performed on the manual address in the form of an ICMP Neighbor Solicitation.

As usual, two multicast report messages are sent with the same information.

The interface sends a router solicitation to acquire a second address via a router. Since no configured router is present it is never responded.
In previous sections the Link Layer (MAC) address of the default gateway (router) was acquired from the router advertisement. This time the gateway was configured statically as observed in Figure 31. During autoconfiguration the interface sends a neighbor solicitation aimed at the gateway in order to discover its Link Layer address.

The destination of this Neighbor Solicitation is the solicited node multicast address of the neighbor (ff02::1:ff00:1). Its Link Layer destination is the equivalent multicast MAC address (33:33:ff:00:00:01).
The interface of the router that is marked as the gateway responds to the Neighbor Solicitation. Among the ICMPv6 options the Link Layer address is clearly specified.

By checking the webserver’s MAC resolution table after autoconfiguration has finished we can see that both the link local and the global address of the router are associated with their interface’s MAC and marked as the router.

5.2. Ping

Objective: Ping is one of the simplest and most important tools when testing connections. Ping is going to be used to test the connection between 2 hosts using local and global addresses, and then test the connection with an “external” (beyond the Link scope) IPv6 machine.
5.2.1. Necessary machines

- At least two of the clients (Ubuntu-1, Ubuntu-2 or Windows-1 if available).
- UbuntuRouter
- Any machine in IPv6 group and the necessary machines to reach it

5.2.2. Diagram of the connection

Figure 40: Machines needed for Ping
5.2.3. Between hosts using local address

We turn on the necessary client machines and the IPv6 network. The MAC address tables of the clients should be empty on boot.

Next, we start a capture on Ubuntu-1, filtering in packages with the source or destination address of Ubuntu-1 (fe80::a00:27ff:fe0b:4677) and Ubuntu-2 (fe80::a00:27ff:fe95:f40c), which we are about to ping.

ssh root@192.168.56.11 'tshark -i eth1 -f "(src host fe80::a00:27ff:fe0b:4677 or dst host fe80::a00:27ff:fe0b:4677 or src host fe80::a00:27ff:fe95:f40c or dst host fe80::a00:27ff:fe95:f40c)" -w -' | wireshark -k -i -

This capture is saved as local ping.

After the capture starts we send local pings from Ubuntu-1 to Ubuntu-2

ssh root@192.168.56.11 'ping6 -I eth1 ff02::1:ff0b:4677'
Figure 41: Diagram of Ping capture 1
Before Ubuntu-1 can send the ping it requires the MAC address of the destination. To discover this address, it follows the procedure specified in RFC 4861 and sends a Neighbor solicitation.

The neighbor solicitation has a destination address of ff02::1:ff0b:4677, the solicited node multicast address of Ubuntu-2. Its Link Layer destination is set to the equivalent multicast MAC address 33:33:ff:0b:46:77.
The interface on Ubuntu-2 answers with a neighbor advertisement that includes its MAC address on the ICMPv6 options.

Ubuntu-1 now has the Ubuntu-2 interface on its MAC resolution table and is able to send pings.
The first ping request is sent from Ubuntu-1 as confirmed in the ICMPv6 header type (128) and its sequence number (1). The entire exchange happens in the link-local scope, so the hop limit of the IPv6 header is not affected.
The ping is answered with a ping reply as evidenced by the type (129) and Sequence number (1) of the ICMP header. The reply has the same characteristics as the request.

**Figure 45: Ping Reply**

---

85
The rest of the capture is composed of similar pings of a different sequence.

As specified by the routing table, all link-local packets (fe80::/64) are sent directly via the reachable interface. The specific interface used as the origin is specified via the ping command.

The MAC resolution table only has two addresses, the one associated with the router (08:00:27:8c:66:b6), acquired through the router solicitation during autoconfiguration and the Ubuntu-2 MAC address (08:00:27:0b:46:77) acquired through the neighbor solicitation and advertisement analyzed before.
5.2.4. Between host and external machine using global address

We turn on the necessary client machines and the IPv6 network. The MAC address tables of the clients should be empty on boot.

Next, we start a capture on Ubuntu-1 filtering in packets with a source or destination equal to the MAC generated global address of Ubuntu-1 (2001:db8:0:1:a00:27ff:fe95:f40c) or the router address (2001:db8:0:4::2) or the random global address used by Ubuntu-1 on this instance. The random address is generated on each Ubuntu session and must be put on the filter for each case.

```
ssh root@192.168.56.11 'tshark -i eth1 -f "(src host 2001:db8:0:1:a00:27ff:fe95:f40c or dst 2001:db8:0:1:a00:27ff:fe95:f40c or src host RANDOM_ADDRESS_GOES_HERE or dst host RANDOM_ADDRESS_GOES_HERE or src host 2001:db8:0:4::2 or dst host 2001:db8:0:4::2)re" -w - ' | wireshark -k -i -
```

This capture will be saved as **Global Ping 1**.

We also, launch the same capture with the same filter, but on the destination

```
ssh root@192.168.56.11 'tshark -i eth1 -f "(src host 2001:db8:0:1:a00:27ff:fe95:f40c or dst 2001:db8:0:1:a00:27ff:fe95:f40c or src host RANDOM_ADDRESS_GOES_HERE or dst host RANDOM_ADDRESS_GOES_HERE or src host 2001:db8:0:4::2 or dst host 2001:db8:0:4::2)re" -w - ' | wireshark -k -i -
```

This capture will be saved as **Global Ping 2**.

After the capture starts we send local pings from Ubuntu-1 to Ubuntu-2

```
ssh root@192.168.56.11 'ping6 2001:db8:0:4::2'
```
Figure 49: Diagram of Ping Capture 2

Figure 50: Pings from Global Ping 1
There is not much difference on the ping’s header compared to the link-local case.

However, there is a change on the procedure. The interface does not send a Neighbor advertisement before the pings in order to discover the link layer (MAC) address of the next hop (the router).

This happens because during autoconfiguration the machine saves the MAC address of the router advertisements as the router’s MAC. This is clearly labeled on the MAC resolution table as we can observer on the following figure.
Figure 53: Ubuntu-1 MAC resolution table before pings

Figure 54: Pings from Global Ping 2

Figure 55: Detail of ping request 2

Figure 56: Detail of ping response 2
There are no mayor differences on the captured pings seen from the side of the webserver. Once again we observe that there is no Neighbor Solicitation to discover the MAC address of the next hop. This is because, as noted before, interfaces configured with a static gateway generate a Neighbor Request in order to acquire the MAC address of the gateway during autoconfiguration.

```
root@Ubuntu-ipv6-webserver-DNS: # ip -6 neigh show
2001:db8:0:4::1 dev eth0 lladdr 08:00:27:11:4b:63 router STALE
```

Figure 57: Webserver MAC resolution table
5.3. **HTTP request over IPv6**

**Objective:** Observe a webpage request purely over IPv6.

5.3.1. **Necessary machines**
- Windows-1
- UbuntuRouter
- UbuntuRouter-ipv6-gateway
- UbuntuRouter-ipv6-1
- Ubuntu-ipv6-webserver-DNS

5.3.2. **Diagram of the connection**

![Diagram of the connection]

**Figure 58:** Machines needed for HTTP request
5.3.3. HTTP request using DNS ipv6

We boot all the necessary machines on the network and set a capture on UbuntuRouter-ipv6-1, filtering in the packets which have an origin or destination address of the Ubuntu-ipv6-webserver-DNS (2001:db8:0:4:2). Since the interface doing the request will have a random address, it is easier to simply filter by the static webserver/DNS. The result will be saved as **http request ipv6**.

```
ssh root@192.168.56.18 'tshark -i eth0 -f "(src host 2001:db8:0:4::2 or dst host 2001:db8:0:4::2)" -w - | wireshark -k -i -
```

On Windows-1, we click on the Internet Explorer logo and go to the address **www.webserver4.com**

![Diagram of HTTP capture](image_url)
Before doing an HTTP request the name must be resolved through a DNS query.

The first parameter of the request header is a transaction ID, which is unique to each query-response type and is used to distinguish every query with its response when several queries are issued. Set to 0xe939 for this case.

The flags that deliver more information regarding the query follow:
• **First flag (1 bit)** identifies if the packet is a query (0) or a response. Query for this case.

• **Second Flag (4 bits)** identifies the type of query the message is carrying. All zeros signal a standard query.

• **Third Flag (1 bit)** identifies if the message has been truncated due to it being too long for the transport medium. This usually happens only over UDP packets, since TCP handles the fragmentation on itself. The current query is not fragmented.

• **Fourth flag (1 bit)** identifies if the query can be resolver recursively. Recursive name resolution is the process by which a DNS server uses the hierarchy of zones and delegations to respond to queries for which it is not authoritative. On our case the request can be recursive.

• **Fifth flag (1 bit)** is reserved and always set to 0.

• **Sixth flag (1 bit)** that indicated that all data must be fully authenticated through the server’s local security policies. The client will still accept non authenticated responses if there are no other choices.

The following parameters indicate the number of questions included in the query. In this case we only have one name resolution question.

Followed is the number of responses, authority responses and additional responses. All set to zero, since this is a query.

Three parameters are included in the actual query. The name to be resolved (As set through internet explorer, www.webserver4.com), the query type and class. The type is set to **AAAA**, which according to RFC 3596 corresponds to an IPv6 address record. Class is set to IN, which means Internet, the resolution corresponding to an Internet address (RFC 2929). The only other commonly used class is CH (chaos) is used for querying DNS server versions.
Figure 61: DNS query response
The minimalist DNS server responds with the default header parameters and with the ID set to 0xe939, the same as the query. Most of the flags are kept the same as the query, while some important flags are changed or added by the server.

- **The response flag** is changed to 1 to mark the packet as a response
- **The authoritative flag** is set to 1, to mark that this server as an authority for the domain. This is the default configuration for resolutions done from the local list and that did not need recursion.
- **Recursion available** is set to 1, to mark that the server is capable of doing recursive queries. This is the default response, even though in practice no recursion has been configured for the DNS.
- **Answer authenticated** is set to 0, which means it was not authenticated by the server due to the fact that no security policies are configured in the server. Still, the client will accept this response.
- **The reply code** is set to all zeros, so no error was registered.

The number of questions remains the same (the query is included in the response) and one answer is marked as included.

The body of the queries remains the same, but the header of the answer includes some different details. The name, type and class remain the same as the query, but now the header includes a time to live, which is often used to set, among DNS servers of different levels, how long they can cache the specified resolution without having to query again. A bigger time to live would reduce the amount to traffic centered on authoritative servers at risk of making certain sites (that change their addresses often) unreachable. In our specific case, we are using a minimalist DNS with no configuration regarding zones or authorities, so it does not need to do recursive request of any kind and always has TTL set to 0, with the only effect being that the client will never cache addresses and will always request the resolution. The two other fields simply represent the length of the response data and the resolved address.

Figure 62: TCP connection being established
After the address is acquired a TCP session is established between the client, using the port 49157 and the server using the http 80 port. The details of a TCP connection are beyond the scope of this work, but it is important to have a general view of its role. The communication is between the server’s global address (2001:db8:0:4::2) and window’s temporal global address (2001:db8:0:1:e9cd:20000:7aa5:1dd3).

![HTTP GET Request](image)

**Figure 63: HTTP GET**

Internet explorer makes an HTTP request using an HTTP GET packet. The HTTP header contains several pieces of information that the webserver may use to fetch a specific HTML document.

The **accept** parameter of the header is used to communicate which types of elements are acceptable. In our case, all types accepted by Internet Explorer are accepted.

The **accept language** suggest a language for the fetched document based on the configuration of the Operating System of Browser. In our case, the Windows Virtual Machine is entirely configured in English, so all documents are requested on English. The server can use this tag to send the web page on the appropriate language, but the browser will accept and render any page sent back.

**User agent** is a tag identifying the software being used to render the page (the internet browser, the agent acting in behalf of the user). The server may use this information to render a webpage made to work for a specific browser. Even though we are using Internet Explorer as the browser; the packet reports the user agent as Mozilla 4.0. This is because during the earlier version releases
of Internet Explorer the most important alternatives, Netscape Browser and Firefox, supported more advanced features than Internet Explorer. Web developers limited the features of their pages on all request with an Internet Explorer user agent. Later versions of Internet Explorer started reporting a User Agent of Mozilla to get full features on websites instead of limited pages.

Accept Encoding specifies the types of compression the client browser can handle for the document. In order to save bandwidth, some server compress all HTML documents sent if the client supports that type of compression. In this capture, Internet Explorer supports gzip and deflate algorithm.

Hosts specifies the host name and port requested of the server. If no port is specified it is assumed to be the default port (80).

Connection specifies the type of connection the client prefers to be established. A keep-alive connection is a permanent connection where the client and server will use the same ports for every page, rather than establishing a new TCP session for every page request.

Many of these parameters are informative and can be used by the server and webpages for adaptive content.

The http GET is acknowledged by the TCP connection.
The server answers with an HTTP OK packet, containing some information on the server and the requested HTML page.

The date parameter of the header shows the date of the response (according to the server’s internal clock).

Server specifies the software used by the server to respond to the requests, in our case the apache software version 2.2.
The **Last modified** parameter shows the last time the HTML document was modified by the server. This way the client can verify to be using the latest version of the HTML document.

The **Etag** is parameter used for web page caches. Every resource on the server is associated with a specific Etag, and any change to the resource causes a change on the Etag. The client can use the Etag to verify if the content is the same as the one present in the cache and so save bandwidth by displaying media included in the content from cache without having to download it again.

Sometimes the data flow is interrupted or the client has a partial cache of a resource. In those instances the client may request only a part of the resource to save bandwidth. The **accept-range** indicates with what range is the server able to use for partial requests, in this case it indicates the server is able to answer partial requests to the byte level.

The **vary** parameter tells the browser that two cacheable responses of the same resource will be the same even if the Accept-Encoding request header is deferent.

**Content-Encoding** indicates the algorithm used for compression, in this case, gzip.

**Content Length** indicates the length of the data sent back, 128 bytes.

**Keep Alive** indicates the parameters for the permanent tcp connection. A time-out of 5 seconds means that the connection will be kept alive for 5 second of inactivity. Max of 100 means that the server will allow a maximum of a hundred simultaneous connections from the client.

The connection parameter stays the same as the request.

**Content type** indicates the type of data inside the response packet, an HTML document for this case.

Finally, the content itself is displayed as sent to the client, a simple HTML test page.
The HTTP OK packet is acknowledged.

A second DNS query is done. By default, Internet Explorer requests a favicon icon for the webpage icon. Since the DNS response has a TTL of 0, every request must first be resolved from the DNS server. This query is identical to the previous one, except for the transaction ID.

Figure 67: Second DNS query (For Favicon)
The query is responded with the same parameters we have seen before but with the new transaction ID.

A second TCP connection is established for the Favicon request.
A second HTTP GET request is done for the favicon icon using the same parameters.

The GET request is acknowledged.
No favicon is configured on the server, so the request resource is not found. The server response has the same parameters as the last one, except for the status code (404 for not found) and the data, which is a default “not found” page for Apache.

The HTTP response is acknowledged and the TCP connection is finished.
5.4. FTP over IPv6

Objective: Observe an FTP exchange purely over IPv6.

5.4.1. Necessary machines

- Ubuntu-1 or Ubuntu-2
- UbuntuRouter
- UbuntuRouter-ipv6-gateway
- UbuntuRouter-ipv6-1
- Ubuntu-ipv6-webserver-DNS
5.4.2. Diagram of the connection

Figure 73: Machines needed for FTP
5.4.3. FTP connection and file transfer over IPv6

“FTP requires two connections (also named channels) per session: a control channel and a data channel. The control channel is a persistent connection over which commands are sent and short responses are received. The data channel is a connection that is typically reestablished for each data transfer, such as a directory listing, or a file upload or download.” (Microsoft)

The control channel is transparent to the new IP version, same as HTTP. It is established with no problems or reference to IPv6 addresses at the moment the FTP connection starts.

However, the data channel is established when a file will be downloaded, uploaded or directory listed. FTP assumes that the addresses used for the data connection will be 32 bits in length (IPv4 addresses). With the introduction of IPv6 and extension was added to FTP on RFC 2428 to allow the specifications of extended addresses, as will be observed shortly.

We boot all the necessary machines on the network and set a capture on UbuntuRouter-ipv6-1, filtering in the packets which have the origin or destination address of the Ubuntu-ipv6-webserver-DNS (2001:db8:0:4:2). Since the interface doing the request will have a random address, it is easier to simply filter by the static webserver/DNS. The result will be saved as ftp ipv6.

ssh root@192.168.56.18 'tshark -i eth0 -f "(src host 2001:db8:0:4::2 or dst host 2001:db8:0:4::2)" -w -' | wireshark -k -i -
On Ubuntu-1 we run the command to connect to the server.

```
ftp 2001:db8:0:4::2
```

Figure 74: Diagram of FTP capture

First, the TCP connection is established between the host and the FTP server.

Figure 75: TCP connection established for FTP
FTP packets usually consist of a response code and a response argument associated with the code.

The current response is triggered by the establishment of the FTP connection, as signaled by its response code 220 (Ready for new user). Its argument is the message to be displayed at the client’s console as specified in the configuration file in section 4.5.1.1.2.

The FTP packet with the welcome message is acknowledged by TCP.

On Ubuntu-1 the prompt asks for a user and is supplied the *anonymous* username.
The client sends a request FTP packet. A request packet is usually formed by a request command and a request argument that contains additional information about the command.

On this packet the request command is set to USER, meaning that the client is specified a user to log in, and the argument is set to anonymous, the name of the user that is to be logged in.

The request packet is acknowledged by TCP.

As soon as the username is received the server sends as FTP response with the 331 response code. The user is correct and now requires password in order to log in. The response argument is the message that is now displayed on the client’s prompt.

TCP acknowledges the reception of the FTP response.

Since we are making an anonymous FTP connection the password is left blank at the client. Only enter is pressed.
The client sends an FTP request with the PASS command. Since the password is blank, no request argument is included.

The server sends back a FTP response with the 230 code, signaling that the user has been correctly logged in. The response argument is what is displayed on the client’s prompt.
The next FTP request uses the SYST command which asks for information about the server’s operating system.

The server responds with a 215 response code and the system type on the argument: *UNIX Type: L8*, a UNIX server that uses binary mode to transfer files.

The FTP response specifying the system is acknowledged by TCP.

The FTP client is now ready to get or push files into the folder specified by the configuration file. On the folder a small text file (dummy.txt) has been placed in order to fully test an FTP exchange. To get the file the client simply runs the necessary FTP command:

```
get dummy.txt
```
In order to get the file the client must first request the server to go into binary mode, which will allow the server to transfer files bit by bit. This is the mode FTP servers use to transfer most file formats.

The TYPE command requests control of the binary flag. A request argument of I asks for the flag to be turned on.

The FTP response accepts the command (code 200) and on its argument confirms the change to binary mode.

The FTP response is acknowledged by TCP.
Now the server requires the addresses necessary for the data connection. It is here where the extensions made by RFC 2428 take action through the EPRT command.

“The EPRT command allows for the specification of an extended address for the data connection. The extended address MUST consist of the network protocol as well as the network and transport addresses. The format of EPRT is:

EPRT<space><d><net-prt><d><net-addr><d><tcp-port><d>” (IETF, 1998)

Here the EPRT command is used by the client with the argument structured as specified on RFC 2428 and separated into fields by wireshark.

The first parameter of the argument is set to 2, the number set to IPv6 addresses.

“AF Number  Protocol
        --------  --------
        1           Internet Protocol, Version 4 [Pos81a]
        2           Internet Protocol, Version 6 [DH96]” (IETF, 1998)
The second parameter is the client’s public IPv6 address (2001:db8:0:1:9863:ec41:a7cc:e7b2) in string representation.

The third parameter is the active port that the client has available for the data connection (41543).

![FTP response, EPRT successful](image)

Figure 92: FTP response, EPRT successful

The server sends an FTP response accepting the command (response code 200). On the response argument the server message recommends using EPSV.

“The EPSV command requests that a server listen on a data port and wait for a connection. (...)When the EPSV command is issued with no argument, the server will choose the network protocol for the data connection based on the protocol used for the control connection.” (IETF, 1998)

EPSV is an alternative to EPRT for FTP over IPv6. Rather than specifying a client address and port to the server it would allow the server to listen on a data port and wait for the client to connect using the same protocol used for the control connection. If the control connection is through IPv6, the server will listen on an IPv6 port.

In our case EPRT suffices. Using EPSV is only a recommendation.
Now the server has already switched into binary mode and has the details of the IPv6 TCP data connection.

Next the client issues the RETR FTP command that actually requests a file. The request argument specifies the name of the file taken from the command prompt, *dummy.txt*.

The server established the TCP connection to be used for the data transmission using the details that were specified in EPRT command.

The server sends an FTP response to announce that the file is correct and that it is about to use the established data connection (Response code 150).
The file is sent through the data connection. Since we are sending a plain text file its contents are readable on the capture.

The transmission generates its appropriate set of TCP ACK to verify reception. Once the file has been appropriately sent the TCP connection is finished.

Once the file is sent and the TCP data connection is closed the server generates a FTP Response with the 226 code signaling the end of the data connection. The file transmission is complete and the message is shown on the client’s prompt.

TCP acknowledges the reception of the FTP response.
To finish the TCP connection the following command is typed on the command prompt:

```
bye
```

The client generates a FTP request with QUIT command to signal the end of the connection.

```
```

The FTP response is generated announcing the end of the connection with a response code of 221.

```
221 Goodbye.
```

The last packet is acknowledged and the control connection comes to an end.
5.5. Manual Tunnel

Objective: manually configure the two ends of an IPv6 to IPv4 tunnel and try toping over it, observing the encapsulation.

5.5.1. Necessary machines:

- At least one of the clients (Ubuntu-1 in the referenced capture)
- Ubuntu Router
- UbuntuRouter-ipv4-gateway
- UbuntuRouter-ipv4-1
- Ubuntu-tunnelBroker
- Ubuntu-ipv6-webserver-tunnel
5.5.2. Diagram of the connection

Figure 103: Machines Needed for Manual Tunnel
5.5.3. Ping request over Manual Tunnel

The manual tunnel is established through a series of commands that, for convenience, have been condensed in a script in both endpoints of the tunnel. The contents and inner working of the scripts are explained in detail in section 4.6.

In order to check the encapsulation and changes in the packets going through the tunnel we run a capture in three different points of the network.

The first capture happens through Ubuntu-1 (before entering the tunnel), filtering for packets with a destination or origin address of the server on the IPv6 Island (2001:db8:0:8:3).

```
ssh root@192.168.56.11 'tshark -i eth1 -f "(src host 2001:db8:0:8::3 or dst host 2001:db8:0:8::3) and dst host !fe80::a00:27ff:fe95:f40c" -w -' | wireshark -k -i -
```

This capture is saved as a local ping capture.

The second capture goes through UbuntuRouter-ipv4-1 (in the middle of the tunnel), on its eth0 interface. Since no other traffic is currently going through the network, we can apply the capture with no filters.

```
ssh root@192.168.56.15 'tshark -i eth0 -f "port !22" -w -' | wireshark -k -i -
```

The capture is saved as router ping capture.

On the other side of the tunnel the capture happens on the eth2 interface of the Ubuntu-tunnelBroker (after leaving the tunnel), filtering for packets with a destination or origin address of the server on the IPv6 Island (2001:db8:0:8:3).

```
ssh root@192.168.56.23 'tshark -i eth2 -f "(src host 2001:db8:0:8::3 or dst host 2001:db8:0:8::3) and dst host !fe80::a00:27ff:fe95:f40c" -w -' | wireshark -k -i -
```

The capture is saved as island ping capture.
Figure 104: Diagram of Manual Tunnel Capture

```
ssh root@192.168.56.13 '/root/manualTunnel'
ssh root@192.168.56.23 '/root/manualTunnel'
```
Finally, we set send a group of pings to test the connection.

```
ssh root@192.168.56.11 'ping6 2001:db80:8::3'
```

### 5.5.3.1. Capture before tunnel

![Figure 105: Ping Request before manual tunnel](image)

The ping request has the same parameters as the global ping observed in section 5.2.4. We will take note of the hop limit of 64 for future comparison.
Figure 106: Ping response before manual tunnel
5.5.3.2. Capture during tunnel

<table>
<thead>
<tr>
<th>Host Address</th>
<th>MAC Address</th>
<th>Operation</th>
<th>Destination MAC Address</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000000000</td>
<td>CadmusCo_2f:cd:ac</td>
<td>Broadcast</td>
<td>ARP</td>
<td>68 Who has 11.0.0.2? Tell 11.0.0.1</td>
</tr>
<tr>
<td>2.0001066000</td>
<td>CadmusCo_42:72:39</td>
<td>ARP</td>
<td>CadmusCo_2f:cd:ac</td>
<td>42 11.0.0.2 is at 08:00:27:42:72:39</td>
</tr>
</tbody>
</table>

Before the pings go through the IPv4 tunnel an ARP request is needed to discover the MAC addresses needed for link-layer header. The details of ARP are outside the scope of this work, but it is important to note that in IPv6 it has been substituted by Neighbor discovery messages.
Table 108: Encapsulated ping request

<table>
<thead>
<tr>
<th>Time stamp</th>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00:27</td>
<td>CadmusCo 2f:cd:ac</td>
<td>08:00:42</td>
</tr>
<tr>
<td>08:00:27:42</td>
<td>08:00:42:72:39</td>
<td></td>
</tr>
</tbody>
</table>

**Internet Protocol Version 4, Src: 10.0.0.3 (10.0.0.3), Dst: 12.0.0.3 (12.0.0.3)**


**Type: echo (ping) request (ICMPv6)**

**Code: 0**

**Checksum: 0x05f8**

**Identifier: 0x85f5**

**Sequence: 0**

**Data (56 bytes)**
The same ping request has been encapsulated over IPv4 and now has two IP headers (one for each version). The data and the IPv6 header are kept mostly the same, with the only difference being a reduction of one in the hop limit for the hop before the tunnel. The IPv6 header’s hop limit is only affected by IPv6 jumps.

The IPv4 header uses the tunnel endpoints as Source (10.0.0.3) and Destination (12.0.0.3) and has the flag marked for disabling fragmentations to make up for the fact the routers do not fragment IPv6 packets.

Time to live is two hops down from the default 64 hops due to the packet already going through two IPv4 jumps. The IPv4 Time to Live is only affected by IPv4 jumps.

Finally, the protocol field in the IPv4 header is marked as 41, a number assigned to IPv6 encapsulated over IPv4 as specified in RFC 2473.
<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Hop limit</th>
<th>Hop count</th>
<th>TTL expiry</th>
<th>Source port</th>
<th>Destination port</th>
<th>Source address</th>
<th>Destination address</th>
</tr>
</thead>
<tbody>
<tr>
<td>40.0.82.124</td>
<td>2001:db8:0:8::3</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>555555</td>
<td>555555555555</td>
<td>40.0.82.124</td>
<td>2001:db8:0:8::3</td>
</tr>
</tbody>
</table>

**Figure 109: Encapsulated ping response**
5.5.3.3. Capture after tunnel on IPv6 Island

After the tunnel the ping is de-encapsulated on the IPv6 Island and the packet is once again a purely IPv6 packet. The IPv6 Hop Limit is reduced by one for the next hop. The entire tunnel counts as only one hop for the IPv6 header and no other header parameter has evidence of going through a tunnel.

Figure 110: Ping Request after manual tunnel
Figure 111: Ping response after manual tunnel
5.6. ISATAP tunnel

Objective: Have an IPv4 only client ping an IPv6 server in the IPv6 Island using an ISATAP tunnel, where the ISATAP server is Ubuntu-tunnelBroker.

5.6.1. Necessary machines

- Ubuntu-3
- UbuntuRouter-ipv4-gateway
- UbuntuRouter-ipv4-1
- Ubuntu-tunnelBroker
- Ubuntu-ipv6-webserver-tunnel
5.6.2. Diagram of the connection

Figure 112: Machines needed for ISATAP tunnel

5.6.3. Pings over ISATAP tunnel

The ISATAP tunnel is established through a series of commands that, for convenience, have been condensed in a script in both endpoints of the tunnel. The contents and inner working of the scripts are explained in detail in section 4.7.

The ISATAP tunnel allows a purely IPv4 client (Ubuntu-3) to tunnel its way towards an IPv6 island. In order to observe the process completely we set up two captures; one during the tunnel and one on the IPv6 islands where packets have been de-encapsulated.
The first capture goes through UbuntuRouter-ipv4-1, on its eth0 interface. Since no other traffic is currently going through the network, we can apply the capture with no filters.

```
ssh root@192.168.56.15 'tshark -i eth0 -f "port !22" -w -' | wireshark -k -i -
```

The capture is saved as **ISATAP tunnel ping capture**.

On the other side of the tunnel the capture happens on the eth2 interface of the Ubuntu-tunnelBroker, filtering for packets with a destination or origin of the server on the IPv6 Island (2001:db8:0:8:3).

```
ssh root@192.168.56.23 'tshark -i eth2 -f "(src host 2001:db8:0:8::3 or dst host 2001:db8:0:8::3" -w -' | wireshark -k -i -
```

The capture is saved as **ISATAP island ping capture**.
Finally, we set send a group of pings to test the connection.

```
ssh root@192.168.56.23 '/root/startISATAP'
ssh root@192.168.56.24 '/root/startISATAP'
```
ssh root@192.168.56.24 'ping6 2001:db80:8::3'

5.6.3.1. Capture during ISATAP tunnel

![Router Solicitation on ISATAP configuration](image)

As explained on section 2.1.7.2.1, ISATAP creates a temporal tunnel using a link local addresses. In order to continue the tunnel the ISATAP client requires a router advertisement from the endpoint to generate a global address. Here, it requests the router advertisement using a router solicitation.

Wireshark decodes the encapsulated addresses on the summary, but the IPv6 packet is encapsulated over IPv4 using the addresses of the endpoints (12.0.0.3 and 16.0.0.1).

The IPv6 addresses used for the exchange are generated by prepending a prefix to the IPv4 address as explained on section 2.1.7.2.1. The IPv4 address of 16.0.0.1 is written as \textbf{0c000003} in hexadecimal. Prepending the prefix results with the IPv6 address of \textit{fe80::200:5efe:000:3}. The IPv4 address of 12.0.0.3 is written as \textbf{10000001} in hexadecimal. Prepending the prefix results with the IPv6 address of \textbf{fe80::200:5efe:1000:1}.
As for the router solicitation’s body, only the basic ICMPv6 options are used. Compared to previous examples of router solicitations (such as the one observed in section 5.1.3) no extra information is included (such as the link layer source), as there are only two endpoints to the tunnel and two nodes to the link local network.

![Router Advertisement on ISATAP configuration](image)

**Figure 115: Router Advertisement on ISATAP configuration**

The router advertisement is sent using the same radvd software used for autoconfiguration and includes the same ICMP header and option information explained in section 5.1.3. The prefix advertised is the same as specified in the configuration in 4.7.1.2.
Next, the pings are carried over the ISATAP tunnel.

The IPv4 addresses used as origin and destiny are the tunnel’s endpoints.

The IPv6 address used for the origin is the randomly created (using the prefix) global address for the client (For the tunnel) and the IPv6 address used as the destiny is the final destination’s address on the island (2001:db8:0:8::3).
The ping is responded with the same identifiers.
The IPv4 header generated for the ISATAP tunnel follows the same structure generated for the manual tunnel, with an independent time to live, a don’t fragment flag and the tunnel endpoints as the source and end addresses, as well as IPv6 as the protocol identification.
The IPv6 header shows an ISATAP destination/source element detailing the IPv4 address of the ISATAP client, but this is taken from the larger IPv6 address by wireshark.

5.6.3.2. Capture after ISATAP tunnel

Figure 119: Ping request after ISATAP tunnel

After the packet has been through the tunnel and the IPv4 header is removed, we are left with the IPv6 ping. Only the Hop Limit changes, the entire tunnel representing only one hop for IPv6 header.
Figure 120: Ping reply after ISATAP tunnel

The reply follows the same structure before entering the tunnel, with the ISATAP IPv4 destination clearly marked by wireshark.
5.7. **Tunnel Broker**

*Objective:* Create an entity capable of simulating a Tunnel Broker which can create certificates and deliver instructions to a client on how to connect to a tunnel, automating the tunnel creation process. Get all necessary certificates and connect to the tunnel to ping a remote webserver.

**5.7.1. Necessary machines**

- Ubuntu-3
- UbuntuRouter-ipv4-gateway
- UbuntuRouter-ipv4-1
- Ubuntu-tunnelBroker
- Ubuntu-ipv6-webserver-tunnel
5.7.2. Diagram of the connection

![Diagram of the connection]

**Figure 121: Machines needed for tunnel broker capture**

5.7.3. Key exchange, connection and ping over Tunnel Broker tunnel

The tunnel broker generates a series of certificates and keys on request that are unique for every client. On the test network this is achieved using OpenVPN to manage the tunnels and OpenSSL to create and manage certificates. The configuration used on both programs is explained in detail in section 4.8.

A purely IPv4 client (Ubuntu-3) can securely tunnel its way towards an IPv6 island once it has acquired its keys and certificates. In order to observe the process completely we set up two captures; one during the tunnel and one on the IPv6 islands where packets have been de-encapsulated.
We apply similar captures as the ISATAP tunnel, starting with UbuntuRouter on its eth0 interface.

```bash
ssh root@192.168.56.15 'tshark -i eth0 -f "port !22" -w -' | wireshark -k -i -
```

The capture is saved as **tunnelBroker tunnel capture**.

On the other side of the tunnel the capture happens on the eth2 interface of the Ubuntu-tunnelBroker, filtering for packets with a destination or origin address of the server on the IPv6 Island (2001:db8:0:8:3).

```bash
ssh root@192.168.56.23 'tshark -i eth2 -f "(src host 2001:db8:0:8::3 or dst host 2001:db8:0:8::3)" -w -' | wireshark -k -i -
```

The capture is saved as **tunnelBroker island capture**.
A custom script is included on Ubuntu-3 to simulate the process of requesting a certificate from a client using SSH. On execution, the terminal on the client side will display a set of questions (with their default answers) that will be used for the creation of the keys and certificate.

```bash
ssh root@192.168.56.24 "/root/getClientCertificateclientName"
```
Where `clientName` is a unique name, any string will work. If the certificates and keys of the client are erased from the client’s hard drive a new name must be chosen, using the same name as a previous attempt will create a null key that cannot be used for the tunnel.

Most of the questions can be answered using the default answer (by pressing enter) and all yes/no questions need to be answered as yes. If the process is completed successfully the client will copy over its keys, certificates and configuration files to the `/etc/openvpn` folder.

Then, the tunnel is created by starting openvpn on the client using the configuration file given by the tunnel broker.

```
ssh root@192.168.56.24 'openvpn /etc/client.conf&'
```

Once the tunnel is established, a group of pings are sent to test the connection.

```
ssh root@192.168.56.24 'ping6 2001:db80:8::3'
```

### 5.7.3.1. Capture during Tunnel Broker tunnel

The initial key exchange uses SSH for security, so while wireshark is able to identify the exchange as SSH it is unable to see its contents. The inner workings of SSH are beyond the scope of this document.

![Fragment of tunnelBroker client registration](image)
The exchange is pretty long and continues up to packet 340 of the capture. No details can be seen from the encryption.

After the exchange the tunnel is established.

![Fragment of tunnel creation, encrypted](image)

As specified in the openVPN configuration, the much simpler UDP is used for all VPN exchanges with all the data encrypted in such a way that we cannot read its contents. The process of establishing the tunnel continues until packet 522.
Once again, the process is encapsulated on UDP packets and encrypted, so we have no way of verifying the contents of the packets except for the observation that the unencrypted and de-encapsulated pings are clearly visible after the tunnel broker on the IPv6 island, as will be observed in the next section.
5.7.3.2. Capture after tunnel Broker tunnel

The process of registering on the tunnel broker and creating the tunnel happens outside the IPv6 Island, so nothing is reflected in the capture until we send the pings towards the webserver.

Figure 126: ping request after tunnel broker
Compared to the ISATAP tunnel, the ping request has no header parameters indicating a tunnel of any kind.

However the pings that have gone through the tunnel broker do not have the source address set as the tunnel’s endpoint, different from both the ISATAP and manual tunnels. Instead, the origin is marked as the randomly generated address (2001:f5a:53:6::2) assigned to the client by the Tunnel Broker using the prefix in the openVPN configuration as explained in section 4.8.1.3. The address is generated when the tunnel is created and mounted on a virtual interface on Ubuntu-3.

The Ubuntu-3 client, in a way, forms part of the IPv6 network where the tunnel endpoint is its default route.
The ping reply follows the same structure and has the same addresses. Regrettably, due to the encryption no conclusion can be reached regarding the hop limit.
5.8. **NAT64**

*Objective*: create a NAT64 translation point that allows the IPv6 only clients to ping an IPv4 only web server using an especial address prefix. In order to simplify the translation, the IPv6 DNS server translates the `www.webserver4.com` name into the IPv4 address in IPv6 format, prefix included.

5.8.1. **Necessary machines**

- At least one of the IPv6 only clients
- Ubuntu Router
- UbuntuRouter-ipv4-gateway
- UbuntuRouter-ipv4-1
- Ubuntu-ipv4-webserver-DNS
- UbuntuRouter-ipv6-gateway
- UbuntuRouter-ipv6-1
- Ubuntu-ipv6-webserver-DNS
5.8.2. Diagram of the connection

Figure 129: Machines needed for NAT64
5.8.3. IPv6 only ping to an IPv4 guests

NAT 64 allows an IPv6 only interface to reach an IPv4 only interface transparently. This connection is achieved through a router configured to translate all IPv6 addresses with a specific prefix to an IPv4 address and back. In order to make the entire process transparent and independent to the IPv6 machine, the DNS is configured to return IPv4 addresses inside an IPv6 address with the prefix for certain names. In order to make the comparison we simultaneously need to make the capture before and after the translation on UbuntuRouter.

We launch a capture on the eth1 interface of UbuntuRouter with no filters to capture the IPv6 side of the pings.

```
ssh root@192.168.56.13 'tshark -i eth1 -w -' | wireshark -k -i -
```

This capture is saved as **NAT64-IPv6**.

Simultaneously we launch a capture at the same UbuntuRouter but on interface eth2 filtering for packets coming or going from the webserver address (12.0.0.2) or the interface’s address (10.0.0.3).

```
root@192.168.56.13 'tshark -i eth2 -f "(src host 10.0.0.3 or dst host 10.0.0.3) and (src host 12.0.0.2 or dst host 12.0.0.2)" -w -' | wireshark -k -i -
```

This capture is saved as **NAT64-IPv4**.
By executing the NAT64 script on UbuntuRouter we ensure that NAT64 is running.

```
ssh root@192.168.56.13 '/root/taygascript'
```
Finally we send the pings from a client machine, such as Ubuntu-1, using DNS name resolution to www.webserve4.com.

```
ssh root@192.168.56.11 'ping6 www.webserver4.com'
```

### 5.8.3.1. IPv6 side

![DNS request for NAT64](image)

**Figure 131: DNS request for NAT64**

First, the ping utility sends a DNS request for the address associated with www.webserver4.com to the DNS in the IPv6 network. This is the name associated with the IPv4 only webserver, but the request is of the AAAA type and the DNS should return an IPv6 address made by a prefix and an IPv4 address.
As configured, the DNS responds with the address of $2001:db8:1:ffff::c00:2$, which is the hexadecimal equivalent of $2001:db8:1:ffff::12.0.0.2$, the IPv4 address of the webserver (12.0.0.2) appended to the NAT64 prefix ($2001:db8:1:ffff::$).
Before sending every ping request the ping utility does a reverse DNS lookup. This is done in order to correctly display the current name associated with the address on the screen. On this packet we observe the first of these reverse lookups, as evident by the Type parameter of the query, PTR (Domain Pointer). The address to be reverse-checked is encoded in reverse order in the name (2.0.0.0.0.0.c.0.0.0.0.0.0.0.0.f.f.f.f.1.0.0.0.2.b.d.0.1.0.0.2), followed by the constant ip6.arpa that marks all IPv6 reverse DNS requests.
The minimalistic DNS server is capable of answering reverse DNS queries and sends a response back with the server’s name www.webserver4.com.
The ping request on IPv6 follows the structure we have analyzed so far. The target address is the one resolved from the DNS.

The reply ping has the DNS resolved address as the origin. There is no indication on any parameter that the ping has come from a NAT64. The process is transparent for the IPv6 client.
5.8.3.2. IPv4 side

An equivalent ping is generated after going through the NAT64 translator. The next section focuses on the differences on the IP header before and after the translator, but there is little difference on the ICMP header. ICMPv6 has been changed to standard ICMP and the type parameter has been changed from 128 to the equivalent of 8 in the older version. Every other parameter has been preserved, such as the identifier and sequence number. Wireshark shows both the identifier and sequence number in both Big Endian and Little Endian format, but both are for the same parameter inside the packet.

The origin address of the IPv4 packet is the IPv4 interface of the NAT64 translator (10.0.0.3), the same behavior as a standard NAT.

The destination of the IPv4 packet is the address extracted after the prefix (12.0.0.2).
The ping response has its original parameters; the type will be changed from 0 to 129 after the NAT64 translates the header into ICMPv6.

The destination of the packet is the IPv4 interface of the NAT64 translator.

5.8.3.3. Header Comparison

NAT64 translates packet in a way that is transparent to both endpoints. The translator takes the computational task of translating the header of each packet from one protocol to the next using several rules that can be observed by comparing the headers of any packet before and after translation.
The following table shows the differences in a more convenient format.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>IPv6 header</th>
<th>IPv4 Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Header Length</td>
<td>not present</td>
<td>20 Bytes</td>
</tr>
<tr>
<td>DSCP</td>
<td>default</td>
<td>default</td>
</tr>
<tr>
<td>Payload Length</td>
<td>present</td>
<td>Substituted by total length</td>
</tr>
<tr>
<td>Identification</td>
<td>Not present</td>
<td>default</td>
</tr>
<tr>
<td>Flags</td>
<td>not present</td>
<td>Don’t fragment</td>
</tr>
<tr>
<td>Fragment Offset</td>
<td>not presented</td>
<td>0</td>
</tr>
<tr>
<td>TTL</td>
<td>64</td>
<td>61</td>
</tr>
<tr>
<td>Protocol – Next Header</td>
<td>ICMPv6 (58)</td>
<td>ICMP (1)</td>
</tr>
</tbody>
</table>

Table 6: Header comparison for NAT64

The version header is changed and the obligatory field headers present on IPv4 and not in IPv6 are added, such as Header Length, flags, Fragment offset and protocol. The header length is added on IPv4; the total length is computed from the payload length and the fixed header. The protocol parameter is computed from the next header. As for the fragmentation parameters, IPv6 does not fragment; therefore the generated IPv4 packages have a fragment offset of 0 and the do not fragment flag on. IPv6 packets do end-to-end fragmentation using additional headers if necessary but do not allow individual routers to carry out fragmentation.

The TTL or equivalent parameter goes down by 3 every time the packet is translated. This is due to this particular implementation of NAT64, where a virtual interface is created to do the translation. Packets containing the prefix are sent to the virtual interface, adding 2 virtual hops for every packet in addition to the normal hop.
Kernel variables

Kernel variables can be changed on Linux machines while the OS is running using commands described in section 4.10. Several of these variables relate to the inner-workings of the IPv6 implementation. Here, each variable is tested for its behavior and documented for future references.

5.9.1. net.ipv6.conf.eth1.router_solicitations

This variable controls the number of router solicitations sent during autoconfiguration. If none of the Router Solicitations are answered no global address is assigned. The default value is 3.

![Figure 141: router_solicitations set to 3 from the router_solicitations_3 capture](image1)

By changing the value to 2 only two router solicitations are sent.

```
sysctl -w net.ipv6.conf.eth1.router_solicitations=2
```

![Figure 142: router_solicitations set to 2 from the router_solicitations_2 capture](image2)
5.9.2. net.ipv6.conf.eth1.router_solicitation_interval

This variable controls the amount of time between router solicitation messages. The default value is 4 as can be observed in Figure 141: router_solicitations set to 3 from the router_solicitations_3 capture, where the second router solicitation is sent 4 seconds after the first and the third 4 seconds after the second.

By changing the value to 2 router solicitations are sent quicker, 2 seconds between each other.

```
sysctl -w net.ipv6.conf.eth1.router_solicitation_interval=2
```

![Figure 143: router_solicitation_interval set to 2 from the router_solicitation_interval_2 capture](image)

5.9.3. net.ipv6.conf.eth1.router_solicitation_delay

This variable controls how long does the system waits between putting the interface up and sending the first router solicitation message. The default value is of one second, but it exact practical value is difficult to measure due to the fact that wireshark marks time from the first packet captured and not from the interface going up.

```
sysctl -w net.ipv6.conf.eth1.router_solicitation_delay=6
```

![Figure 144: router_solicitation_delay set to 1 from the router_solicitation_delay_1 capture](image)

By changing the value to 6 we can see an increment in the time between the first packet and the first Router solicitation.

![Figure 145: router_solicitation_delay set to 6 from the router_solicitation_delay_6 capture](image)
5.9.4. net.ipv6.conf.eth1.dad_transmits

This variable controls the number of DAD verification packages (neighbor solicitation) sent when a new address is assigned. The default value is 1 as seen in Figure 144: router_solicitation_delay set to 1 from the router_solicitation_delay_1 capture.

By changing the value to 3 each address will be checked for duplicates 3 times.

```
sysctl -w net.ipv6.conf.eth1.dad_transmits=3
```

![Figure 146: dad_transmits set to 3 from the dad_transmits_3 capture](image)

5.9.5. net.ipv6.conf.eth1.accept_dad

This variable enables or disables the use of DAD.

If the value is set to 0 no DAD will be used. If the value is set to 1 a DAD neighbor solicitation will be sent, but detecting a duplicate (A neighbor advertisement) will do nothing. If the value is set to 2 a DAD neighbor solicitation will be sent and if a duplicate is detected IPv6 will be disabled.

The default value is 1, as seen on figure Figure 144: router_solicitation_delay set to 1 from the router_solicitation_delay_1 capture. If we change the variable to 0 no neighbor solicitation is sent.

```
sysctl -w net.ipv6.conf.eth1.accept_dad=0
```
In order to verify the interface’s behavior when the variable is changed to 2 we temporarily change the MAC address of one of the neighbors to the same MAC address as the client and boot the neighbor first. This will ensure that a duplicate address exists on the network that it will respond to the neighbor solicitations.

If the value of the variable is changed to 2 IPv6 is disabled when the neighbor solicitation is answered.

```
sysctl -w net.ipv6.conf.eth1.accept_dad=2
```

5.9.6. net.ipv6.conf.eth1.accept_ra

This variable controls if the interface will accept router advertisements for autoconfiguration or ignore them.
A value of 0 will cause the interface to ignore all router advertisements, not using any of their information for global address autoconfiguration. A value of 1 will make the interface accept router advertisements for autoconfiguration as long as IP forwarding is disabled (if the interface does not belong to a router).

The default value is 1. On a client with no IP forwarding enabled the client will accept the router advertisement and assign a global address.

By enabling IP forwarding and restarting the interface the router advertisements are ignored and no global address is assigned.

```
sysctl -w net.ipv6.conf.eth1.forwarding=1
```

The same result is observed by setting the variable to 0, regardless of the IP forwarding.
sysctl -w net.ipv6.conf.eth1.forwarding=0

sysctl -w net.ipv6.conf.eth1.accept_ra=0

Figure 152: accept_ra set on 0, ip forwarding disabled

5.9.7. net.ipv6.conf.eth1.hop_limit

This variable controls the hop limit of all packets generated (not forwarded) by the interface. The default value is 64.
By changing the hop limit to 50 all outgoing packets will have a reduced hop limit.
6. CONCLUSION AND FUTURE RECOMMENDATIONS

The exhaustion of the IPv4 address space makes the change to IPv6 necessary. However, the current services and networks rely heavily on IPv4 and cannot change to IPv6 on a short term. The first updated networks have created IPv6 islands among the IPv4 network. These islands will continue to grow until they form most of the network. However, both protocols will function simultaneously for a while. This is why it is very important to understand both the IPv6 protocol and the protocols that will allow it to interact with IPv4 during the transition.

The objective of this project was to create a virtual network where the basic features of IPv6 could be tested and studied on every level, as well as its interactions with an IPv4 network.

One of the most important new features of IPv6 is auto configuration. On IPv4 networks clients rely on the DHCP protocol to automatically acquire addresses. IPv6 autoconfiguration is a mechanism that allows IPv6 interfaces to automatically acquire addresses. Compared to DHCP it does not require a centralized server. Autoconfiguration generates an address for the local scope using the interface’s MAC and checks for its uniqueness. Generating the global address requires a network prefix provided by the closest router. Once the prefix is received the interface can check the global address for uniqueness and use it. Routers require software in order to general the router advertisements that communicate the network prefix. The necessary software is easily configurable and reliable.

A version of DHCP exists for IPv6, but autoconfiguration could act as a replacement if properly configured, since it is already included into the protocol suite and it includes all that is necessary for address generation and duplicate detection without having to rely on a central server.

Linux based machines generate link local and (if the router is available) global addresses based on the interface’s MAC. After the addresses have been assigned DAD is used to check for duplicates. However, as the time of writing Linux’s default behavior when detecting a duplicate is to ignore it. This variable can be changed to disable IPv6 on the interface when a duplicate is detected.

Windows generates its Link Local Address randomly and generates two global addresses, one derived from the Link Local Address and another at random. Windows implements optimistic
DAD, meaning that the interface does not wait for DAD to conclude before using the generated address.

HTTP, like most protocols that are on the layers above the IP protocol, works on IPv6 without much difference. Some other protocols have important key differences in order to make them compatible with the changes brought by IPv6.

ICMPv6 is the IPv6 version of ICMP. The two protocols are almost identical, only different on the numbers used for the type parameter and some minor changes to the header. This was done in order to accommodate for the new functionalities of IPv6 that require ICMP, such as duplicate address detection and router solicitation during autoconfiguration.

FTP requires two TCP connections during regular use. One is the Control Connection that is used to pass along FTP commands and the other is the data connection that is used to pass files. The data connection is established before every file is sent. The Control Connection is transparent to the IP protocol used but the Data connection assumes that 32 bit addresses will be used, so it requires an especial command, EPRT, to specify the longer IPv6 addresses that will be used for the Data Connection.

As for the protocols used for IPv4-IPv6 transition, a number of conclusions can be drawn from each analysis.

Tunnels are a straight forward, effective and simple way of getting IPv6 traffic through an IPv4 only area. Manual tunnels require some configuration on both ends of the tunnel. This makes them unpractical for users and impossible for cases on which access to both endpoints is not possible. They are better suited for specific cases between two hosts.

ISATAP is a good example of a protocol used to automate the process. On an ISATAP tunnel all that the client needs to do is to connect to an IPv4 reachable ISATAP server using an ISATAP client. The server and the ISATAP protocol will handle the configuration of the tunnel and the addressing. While the protocol is simple and straightforward for clients the ISATAP server requires a lengthy configuration, both to build the ISATAP tunnel and to provide the Router Advertisements needed for the generation of global IPv6 addresses.

Both of the tunnels mentioned so far don’t implement any form of security. This means that both the encapsulated traffic and the tunnel details (if any) are perfectly visible for anyone in the path. It is also worth noting that any user may connect to the endpoints of the tunnels given the correct configuration. The use of these tunnels is limited on instances when the content of packets or the tunnel details must be kept secret, or when access to the tunnel must be limited to authorized users. A tunnel broker is better for such cases.

A tunnel broker provides a tunnel on a virtual private network. A set of certificates and keys are needed for each client’s connection in order to authorize the client and to encrypt the contents of the packets. Any middleman in the connection will not be able to read the content of the
packets or the details of the tunnel. This security requires additional complex configuration, both for the tunnel and the generation of the certificates and keys of the VPN. A small oversight in the configuration could lead to a working tunnel with important security holes that could allow a malicious user to access the tunnel.

The tunnel broker is the best option when protection of the tunnel details and authentication control is of priority.

Finally, NAT64 is a protocol that allows IPv6 only clients to access IPv4 address seamlessly using address translation. An especial prefix for IPv6 address is selected, and any packet received by the router with a destination address that includes the prefix is translated into an IPv4 packet and mapped. All packets coming from the IPv4 network from the intended address are translated back into IPv6. Both sides have no obvious sign of going through a translator.

This protocol delivers excellent results when IPv6 clients need to access the IPv4 network. However, it has some important disadvantages. All the computational task of the translation falls on the router doing the translation and can possible generate an important overhead. Also, the protocol requires a significant amount of configuration, both on the router doing the translation and on the clients. The clients will need to do all IPv4 requests as IPv6 address with the configured NAT64 prefix. This can be simplified by configuring the DNS to answer all AAAA resolutions for an IPv4 only site as an IPv6 address with the prefix.

As for future recommendations, the current virtual network has room for possible expansions. Dozens of protocols exists for IPv6-IPv4 interaction and several others could be tested by adding extra virtual machines. However, it is important to note that additional virtual machines will require addition memory and CPU.

As a final note, it is recommended that future implementations refrain from extensively using Windows virtual machines. They require a great deal of Memory and CPU time and they resist any extensible configuration, reveal very little information and in general act as poor devices for this kind of experimentation.
7. BIBLIOGRAPHY


VirtualBox Developers VirtualBox Main page [En línea] // VirtualBox. - 21 de October de 2013. - https://www.virtualbox.org/.


Way Alan How DNS64 and NAT64 operate [En línea] // Youtube. - 12 de December de 2013. - https://www.youtube.com/watch?v=uGxPWUC9i3U.

8. GLOSSARY

- **Certificate authority**: an entity that issues digital certificates.
- **Certificate**: on this instance, an electronic certificate. An electronic document used to show ownership of a public key, including a digital signature that can only be generated by the correspondent private key.
- **DAD**: Duplicate address detection, a process done by auto configuring IPv6 hosts which uses ICMP packages to check for other hosts that already have the same address that the hosts intends to take.
- **DHCP**: Dynamic Host Configuration Protocol, used as a standardized protocol used to dynamically distribute and assign addressers to hosts on a network.
- **DNS**: Domain Name System, a protocol that allows associating readable names with IP addresses stored in a hierarchical distributed systems. Clients that require an address can query a DNS server with a name and get the associated address. The reverse process can also be done.
- **Ethernet**: a family of computer networking technologies used for to physically connect multiple computers between each other. It is de facto standard for wired connection in home and office networks.
- **Favicon**: a small, low resolution icon associate with a web site or web page. Usually shown on the tab bar.
- **FTP**: File Transfer Protocol. A protocol used to transfer files between two hosts over a TCP network.
- **HTTP**: Hypertext Transfer Protocol allows for the sending of hypertext (webpages) from a server to a requesting client.
- **ICMP**: Internet Control Message Protocol, used by network devices to send query or error messages.
- **IPv4**: Internet Protocol version 4, the fourth version of the Internet Protocol, used to route most traffic on the internet. A layer four connectionless protocol for packet-switched networks.
- **IPv6**: Internet Protocol version 6, the successor to IPv4, which has a bigger address space and incorporates a set of features that where added by newer protocols on IPv4.
- **ISATAP**: Intra-Site Automatic Tunnel Addressing Protocol, an automatic tunneling protocol that requires a centralized server to assign addresses and establish the tunnel.

- **Linux**: a group of free and open source operating systems working under the Linux kernel.

- **MAC**: Media access control, a type of 48 bit address associated with the link layer protocol of Ethernet and its derivatives. Every physical interface ever built should have a different MAC address.

- **Multihoming**: a computer or device connected to more than one computer network. When a multihoming device has only one interface, the interface has several addresses, each for one different network.

- **NAT**: is a methodology for changing the information of an IP package while in transit through a routing device. This is often used to remap the IP address so a set of computers in a local network is able to work under one public address but have individual private addresses.

- **Ping**: a computer network utility that allows testing the reachability of a network interface by sending an ICMP message targeted at the interface’s address and getting a response back.

- **RFC**: Request for Comments, a publication by the Internet Engineering Task Force and the Internet Society meant to explain the methods, behaviors, research or innovations related to internet protocols.

- **RSA**: a widely use systems for message encryption and secure data transition.

- **SSH**: A protocol that allows to securely and remotely access and run commands on a remote machine.

- **TCP**: Transmission Control Protocol, one of the core members of the Internet Protocol Suite. TCP allows sending a stream of packets in an ordered, reliable and error-checked environment. It requires the establishment of a channel before transmission begins.

- **Ubuntu**: The most widely used Linux Distribution (Linux based operating system).

- **UDP**: User Datagram Protocol, one of the core members of the Internet Protocol Suite. UDP allows sending messages (datagrams) without prior communication to set up special transmission channels. It does not guarantee reception.

- **Windows**: a commercial operating system maintained by Microsoft. It is the most widely used personal computer operating system.
9. ATTACHEMENTS