

## TRAINING SESSION NAME

# Firewall

### Table of contents

- [A. Main Content](#)
  - [1. Technical concept](#)
    - [1.1 IPv4/ IPv6](#)
      - [1.1.1 what is IPv4/ IPv6](#)
      - [1.1.2 compare IPv4 and v6](#)
    - [1.2 Unicast/ Multicast/ Broadcast](#)
      - [1.2.1 Unicast](#)
      - [1.2.2 Multicast](#)
      - [1.2.3 Broadcast](#)
  - [2. SAD, SDD](#)
  - [3. History Issues, lesson learn](#)
  - [4. Demo](#)
    - [4.1 Direct broadcast:](#)
    - [4.2 Multicast](#)
  - [5. Development plan for future](#)
- [B. Questions, Exercises](#)
- [C. References](#)
- [D. Sharing-Training Activities\( if have\)](#)

## A. Main Content

### 1. Technical concept

#### 1.1 IPv4/ IPv6

##### 1.1.1 what is IPv4/ IPv6

- **IPv4** (Internet Protocol version 4) – is is the fourth version of the Internet Protocol (IP).
- And **IPv6** is sixth version, has been develop to support IPv4 and meet the need of using IP in the world.
- **IPv4** has the length of **32 bit**, used to identify and store IP address. It can store maximum is  $2^{32}$  (>4 billion) address.
- **IPv6** is **128bit**, can store  $3.4 * 10^{38}$ .

IPv4	vs.	IPv6
Deployed 1981		Deployed 1998
32-bit IP address		128-bit IP address
4.3 billion addresses		$7.9 \times 10^{28}$ addresses
Addresses must be reused and masked		Every device can have a unique address
Numeric dot-decimal notation		Alphanumeric hexadecimal notation
192.168.5.18		50b2:6400:0000:0000:6c3a:b17d:0000:10a9 (Simplified - 50b2:6400::6c3a:b17d:0:10a9)
DHCP or manual configuration		Supports autoconfiguration

### 1.1.2 compare IPv4 and v6

Detail	IPv4	IPv6
<b>Address length</b>	32-bit	128-bit
<b>Auto address configuration</b>	need a DHCP server	Support method for device to configure automatically
<b>connection integrity</b>	Unachievable	Achievable
<b>number of address</b>	It can generate $4.29 \times 10^9$ address space	Address space of IPv6 is quite large it can produce $3.4 \times 10^{38}$ address space
<b>Security feature</b>	dependent on application	IPSEC is an inbuilt security feature in the IPv6 protocol
<b>Address representation</b>	decimal and use dot (.) to separate 4 fields Example of IPv4: 192.168.2.	hexadecimal and use dot (:) to separate 4 fields Example of IPv6: fd53:7cb8:383:3::15
<b>Fragmentation</b>	Fragmentation performed by Sender and forwarding routers	In IPv6 fragmentation performed only by the sender
<b>Packet flow</b>	In IPv4 Packet flow identification is not available	In IPv6 packet flow identification are Available and uses the flow label field in the header



Detail	IPv4	IPv6
checksum	In IPv4 checksum field is available	In IPv6 checksum field is not available
Transmission	Unicast/ Multicast/ <b>Broadcast</b>	Unicast/Multicast/ <b>AnyCast</b>
Encryption and Authentication	In IPv4 Encryption and Authentication facility not provided	In IPv6 Encryption and Authentication are provided
IP header	IPv4 has a header of 20-60 bytes.	IPv6 has header of 40 bytes fixed
convert to other version	IPv4 can be converted to IPv6	Not all IPv6 can be converted to IPv4
Class	IPv4's IP addresses are divided into five different classes. Class A , Class B, Class C , Class D , Class E.	IPv6 does not have any classes of IP address.

### Auto address configuration

- In IPv4 hosts were originally configured manually. Later, [host configuration protocols like DHCP](#) enabled servers to allocate IP addresses to hosts that joined the network.
- IPv6 takes this a step further, by defining a method for some devices to automatically configure their IP address and other parameters **without the need for a server**.

It also defines a method whereby the IP addresses on a network can be renumbered (changed en masse). These are the sorts of features that make TCP/IP network administrators drool.

### connection integrity

### Security feature

### Fragmentation

- Both IPv4 and IPv6 packet can be fragment by sender, but only **IPv4 router** can do it.
- Under IPv4, when router receives a packet larger than the next hop's MTU has two options:

- drop the packet if the *Don't Fragment* (DF) flag bit is set in the packet's header and send an [Internet Control Message Protocol](#) (ICMP) message which indicates the condition *Fragmentation Needed* (Type 3, Code 4),
- fragment the packet and send it over the link with a smaller MTU.

## **Encryption and Authentication**

## **Convert to other version**

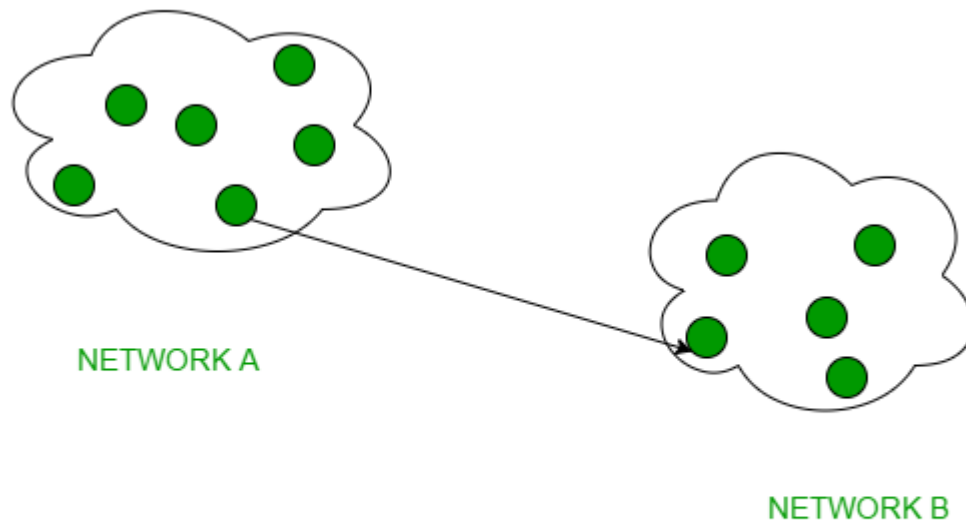
### **1.2 Unicast/ Multicast/ Broadcast**

The "cast" term here stand for some data(stream of packets) is being transmitted from one point to other point(s).

Let's see some of the “**cast**” concepts that are being used in the computer networks field.

#### **1.2.1 Unicast**

- Unicast is useful when there is a participation of a single sender and a single recipient (point to point). So, in short, you can term it a one-to-one transmission.
- For example, if a device having IP address 192.168.2.11 in a network wants to send the data packets to the device with IP address 20.12.4.2 in the other network, then unicast comes into the picture.
- This is the most common form of data transfer over networks.



UNICAST EXAMPLE

### 1.2.2 Multicast

- In multicasting, one/more senders and one/more recipients participate in data transfer traffic.
- In this method traffic recline between the boundaries of unicast (one-to-one) and broadcast (one-to-all). Multicast lets servers direct single copies of data streams that are then simulated and routed to hosts that request it.
- IP multicast requires the support of some other protocols like **IGMP (Internet Group Management Protocol)**, **Multicast routing** for its work. Also in Classful IP addressing **Class D** is reserved for multicast groups.

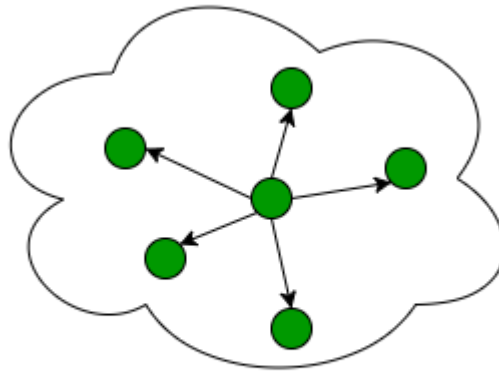
### 1.2.3 Broadcast

Broadcasting transfer (one-to-all) techniques can be classified into two types:

- **Limited Broadcasting:**

If you have to send a stream of packets to all the devices over the network that your reside, this broadcasting can help you to do so.

For this, it will append 255.255.255.255 called **Limited Broadcast Address** in the destination address of the datagram (packet) header which is reserved for information transfer to all the recipients from a single client (sender) over the network.



NETWORK CLUSTER

- **Direct Broadcasting:**

This is useful when a device in one network wants to transfer packet stream to all the devices over the other network.

This is achieved by translating all the Host ID part bits of the destination address to 1, referred to as **Direct Broadcast Address** in the datagram header for information transfer.

This mode is mainly utilized by television networks for video and audio distribution. One important protocol of this class in Computer Networks is [Address Resolution Protocol \(ARP\)](#) which is used for resolving an IP address into a physical address which is necessary for underlying communication.

- Finally, we will take a look on the table below to find out the difference of them:

Feature	Unicast	Broadcast	Multicast
<b>Transmission</b>	Data will be sent to <b>only one</b> recipient	Data will be sent to <b>all</b> recipients in a network	Data will be sent to <b>a group</b> of recipients
<b>Addressing</b>	Uses a unique destination address	Uses a special broadcast address	Uses a special multicast address



<b>Delivery</b>	Guaranteed delivery	Not all devices may be interested in the data	Not all devices may be interested in the data
<b>Network Traffic</b>	Generates the least amount of network traffic	Generates the most amount of network traffic	Generates moderate network traffic
<b>Security</b>	More secure because data is sent to a specific recipient	Less secure because data is sent to all devices in the network	Moderately secure because data is sent to a specific group of devices
<b>Examples</b>	Email, file transfer	DHCP requests, ARP requests	Video streaming, online gaming

## 2. SAD, SDD

## 3. History Issues, lesson learn

## 4. Demo

### 4.1 Direct broadcast:

- command for device 1:

```
sudo ip addr add 10.12.1.11/24 dev eth0 #set IP
```

```
sudo systemctl restart NetworkManager
```

- command for device 2:

```
sudo sysctl net.ipv4.icmp_echo_ignore_broadcasts=0 #enable echo reply
```

```
sudo ip addr add 10.12.1.22/24 dev eth0 #set IP
```

```
sudo systemctl restart NetworkManager
```

- command for device 3:

```
sudo sysctl net.ipv4.icmp_echo_ignore_broadcasts=0 #enable echo reply
```

```
sudo ip addr add 10.12.1.33/24 dev eth0 #set IP
```

```
sudo systemctl restart NetworkManager
```

- Use device 1 ping to direct broadcast address:

```
ping 10.12.1.255 -b
```

=>> only device 3 can receive and reply ping:

```
(root@kali)-[/home/kali/tai]
# ping 10.12.1.255 -b
WARNING: pinging broadcast address
PING 10.12.1.255 (10.12.1.255) 56(84) bytes of data.
64 bytes from 10.12.1.22: icmp_seq=1 ttl=64 time=1.55 ms
64 bytes from 10.12.1.33: icmp_seq=1 ttl=64 time=1.76 ms
64 bytes from 10.12.1.22: icmp_seq=2 ttl=64 time=1.30 ms
64 bytes from 10.12.1.33: icmp_seq=2 ttl=64 time=1.30 ms
64 bytes from 10.12.1.22: icmp_seq=3 ttl=64 time=1.49 ms
64 bytes from 10.12.1.33: icmp_seq=3 ttl=64 time=1.82 ms
```

- set up device 2 to IP 10.12.2.22/24:

```
sudo ip addr del 10.12.1.22/24 dev eth0 #delete IP
```

```
sudo ip addr add 10.12.2.22/24 dev eth0 #set IP
```

=>> only device 3 can receive and reply ping:

```
(root@kali)-[/home/kali/tai]
# ping 10.12.1.255 -b
WARNING: pinging broadcast address
PING 10.12.1.255 (10.12.1.255) 56(84) bytes of data.
64 bytes from 10.12.1.33: icmp_seq=1 ttl=64 time=1.55 ms
64 bytes from 10.12.1.33: icmp_seq=2 ttl=64 time=1.62 ms
64 bytes from 10.12.1.33: icmp_seq=3 ttl=64 time=1.97 ms
```

## 4.2 Multicast

### device\_1 send.py

```
import socket

MCAST_GRP = '224.1.1.1'
MCAST_PORT = 5007
# regarding socket.IP_MULTICAST_TTL
# -----
# for all packets sent, after two hops on the network the packet will not
```



```
# be re-sent/broadcast (see https://www.tldp.org/HOWTO/Multicast-HOWTO-6.html)
MULTICAST_TTL = 2

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, MULTICAST_TTL)

# For Python 3, change next line to 'sock.sendto(b"robot", ...' to avoid the
# "bytes-like object is required" msg (https://stackoverflow.com/a/42612820)
sock.sendto("robot", (MCAST_GRP, MCAST_PORT))
```

### **device\_2 receive2.py**

```
import socket
import struct

MCAST_GRP = '224.1.1.1'
MCAST_PORT = 5007
IS_ALL_GROUPS = True

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
if IS_ALL_GROUPS:
    # on this port, receives ALL multicast groups
    sock.bind(('', MCAST_PORT))
else:
    # on this port, listen ONLY to MCAST_GRP
    sock.bind((MCAST_GRP, MCAST_PORT))
mreq = struct.pack("4sl", socket.inet_aton(MCAST_GRP), socket.INADDR_ANY)

sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)

while True:
    # For Python 3, change next line to "print(sock.recv(10240))"
    print (sock.recv(10240))
```

### **device\_3 receive3.py**

```
import socket
import struct

MCAST_GRP = '224.12.1.1'
MCAST_PORT = 5007
IS_ALL_GROUPS = True

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
if IS_ALL_GROUPS:
    # on this port, receives ALL multicast groups
    sock.bind(('', MCAST_PORT))
else:
    # on this port, listen ONLY to MCAST_GRP
    sock.bind((MCAST_GRP, MCAST_PORT))
mreq = struct.pack("4sl", socket.inet_aton(MCAST_GRP), socket.INADDR_ANY)

sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
```

```
while True:
    # For Python 3, change next line to "print(sock.recv(1024))"
    print (sock.recv(1024))
```

- add 3 file above to each device.
- Initial command:

- command for device 1:

```
sudo ip addr add 10.12.1.11/24 dev eth0 #set IP
```

```
route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0 # add device 1 to multicast 224.0.0.0
```

```
sudo systemctl restart NetworkManager
```

- command for device 2:

```
sudo ip addr add 10.12.1.22/24 dev eth0 #set IP
```

```
route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0 # add device 2 to multicast 224.0.0.0
```

```
sudo sysctl net.ipv4.icmp_echo_ignore_broadcasts=0 #enable echo reply
```

```
sudo systemctl restart NetworkManager
```

- command for device 3:

```
sudo ip addr add 10.12.1.33/24 dev eth0 #set IP
```

```
route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0 # add device 3 to multicast 224.0.0.0
```

```
sudo sysctl net.ipv4.icmp_echo_ignore_broadcasts=0 #enable echo reply
```

```
sudo systemctl restart NetworkManager
```

- make sure all device were joined multicast by command:

```
ifconfig | grep MULTICAST
```

\* if it not, turn on by command:

```
ifconfig eth0 multicast
```

```
(root@kali)-[/home/kali/tai]
# ifconfig | grep MULTICAST
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
```

- at device 1 (sender), check and see device 2,3 has joined multicast,(device has joined will reply):

*ping 224.0.0.1*

```
(root@kali)-[/home/kali/tai]
# ping 224.0.0.1
PING 224.0.0.1 (224.0.0.1) 56(84) bytes of data:
64 bytes from 10.12.1.33: icmp_seq=1 ttl=64 time=1.49 ms
64 bytes from 10.12.1.22: icmp_seq=1 ttl=64 time=1.49 ms
64 bytes from 10.12.1.33: icmp_seq=2 ttl=64 time=1.15 ms
64 bytes from 10.12.1.22: icmp_seq=2 ttl=64 time=1.53 ms
64 bytes from 10.12.1.33: icmp_seq=3 ttl=64 time=1.31 ms
64 bytes from 10.12.1.22: icmp_seq=3 ttl=64 time=1.31 ms
^C
— 224.0.0.1 ping statistics —
3 packets transmitted, 3 received, +3 duplicates, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 1.153/1.377/1.527/0.133 ms
```

- at device 2, 3, run python file to listen on multicast group:

device 2 will join 224.1.1.1

device 3 will join 224.12.1.1

- at device 1, run python file to send data to multicast group 224.1.1.1

=> device 2 will receive message and device 3 have nothing

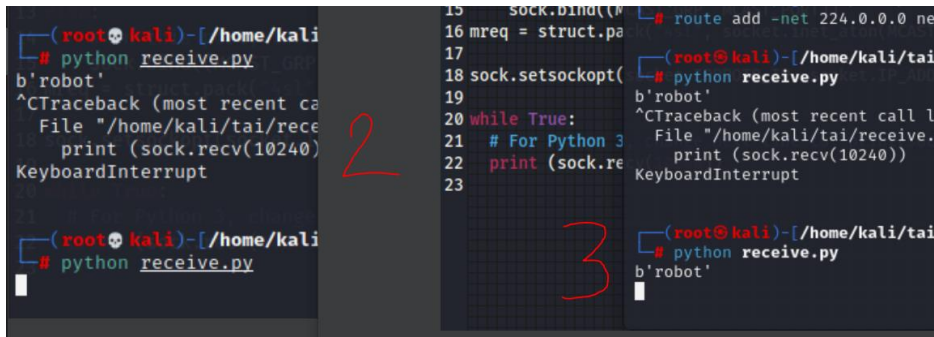
```
File "/home/kali/tai/receive.py", line 15, in <module>
    print(sock.recv(10240))
KeyboardInterrupt
2
(root@kali)-[/home/kali/tai]
# sudo sysctl net.ipv4.icmp_echo_ignore_broadcasts=1
net.ipv4.icmp_echo_ignore_broadcasts = 1
(root@kali)-[/home/kali/tai]
# python receive.py
b'robot'

15 sock.bind((IPV4_MULTICAST_ADDR, IPV4_MULTICAST_PORT))
16 mreq = struct.pack('Ii', IPV4_MULTICAST_ADDR, IPV4_MULTICAST_PORT)
17
18 sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, 64)
19
20 while True:
21     # For Python 3
22     print(sock.recv(10240))
23
3
(root@kali)-[/home/kali/tai]
# python receive.py
Traceback (most recent call last):
  File "/home/kali/tai/receive.py", line 15, in <module>
    print(sock.recv(10240))
OSError: [Errno 19] No such device

(root@kali)-[/home/kali/tai]
# route add -net 224.0.0.0 netmask 255.255.255.255 dev eth0
root@kali:~#
```

- change MCAST\_GRP = '224.1.1.1' in send.py to 224.12.1.1 and send again,

=>> device 3 will receive data and device 2 have nothing



```

15 sock.bind((
16 mreq = struct.pa
17
18 sock.setsockopt(
19
20 while True:
21     # For Python 3
22     print (sock.re
23
# route add -net 224.0.0.0 net
# python receive.py
b'robot'
^CTraceback (most recent call la
File "/home/kali/tai/receive.p
print (sock.recv(10240))
KeyboardInterrupt

# python receive.py
b'robot'
^CTraceback (most recent call la
File "/home/kali/tai/receive.p
print (sock.recv(10240))
KeyboardInterrupt
  
```

## 5. Development plan for future

## B. Questions, Exercises

TBD

## C. References

No .	Info	Link/ file/ name of ebook
1	List of IP version	<a href="https://en.wikipedia.org/wiki/List_of_IP_version_numbers">https://en.wikipedia.org/wiki/List_of_IP_version_numbers</a>
2	unicast/multicast/broadcast	<a href="https://www.geeksforgeeks.org/difference-between-unicast-broadcast-and-multicast-in-computer-network/">https://www.geeksforgeeks.org/difference-between-unicast-broadcast-and-multicast-in-computer-network/</a>
3	compare IPv4, IPv6	<a href="https://www.geeksforgeeks.org/differences-between-ipv4-and-ipv6/">https://www.geeksforgeeks.org/differences-between-ipv4-and-ipv6/</a>
4	IPv6 renumbering	<a href="http://www.tcpipguide.com/free/t_IPv6AutoconfigurationandRenumbering.htm">http://www.tcpipguide.com/free/t_IPv6AutoconfigurationandRenumbering.htm</a>
5	IP fragment	<a href="https://en.wikipedia.org/wiki/IP_fragmentation">https://en.wikipedia.org/wiki/IP_fragmentation</a>
6	IPv6 header format	<a href="https://www.computernetworkingnotes.com/networking-tutorials/ipv6-header-structure-format-and-fields-explained.html">https://www.computernetworkingnotes.com/networking-tutorials/ipv6-header-structure-format-and-fields-explained.html</a>



7	IPv4 header format	<a href="https://www.computernetworkingnotes.com/networking-tutorials/ipv4-header-structure-and-fields-explained.html">https://www.computernetworkingnotes.com/networking-tutorials/ipv4-header-structure-and-fields-explained.html</a>
8	source code multicast demo	<a href="https://stackoverflow.com/questions/603852/how-do-you-udp-multicast-in-python">https://stackoverflow.com/questions/603852/how-do-you-udp-multicast-in-python</a>

