Lab Assignment on Unit V

Aim: Write a program using UDP Sockets to enable file transfer (Script, Text, Audio and Video one file each) between two machines. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.

Requirements: Fedora 20 with Pentium IV and above, 1 GB RAM, 120 G.B HDD, Monitor, Keyboard, Mouse , Modelio, Eclipse, CDT, Python interpreter, Pydev, J2SE, Wireshark Packet Analyzer Tool.

Theory:

TCP guarantees the delivery of packets and preserves their order on destination. Sometimes the features are not required and since they do not come without performance costs, it would be better to use a lighter transport protocol. This kind of service is accomplished by the UDP protocol which conveys datagram packet s. Datagram packets are used to implement a connectionless packet delivery service supported by the UDP protocol. Each message is transferred from source machine to destination based on information contained within that packet. That means, each packet needs to have destination address and each packet might be routed differently, and might arrive in any order. Packet delivery is not guaranteed. The format of datagram packet is:

| Msg | length | Host | serverPort |

Java supports datagram communication through the following classes:

- DatagramPacket
- DatagramSocket

The class DatagramPacket contains several constructors that can be used for creating packet object. One of them is: DatagramPacket(byte[] buf, int length, InetAddress address, int port); This constructor is used for creating a datagram packet for sending packets of length length to the specifi ed port number on the specifi ed host. The message to be transmitted is indicated in the first argument.

The key methods of DatagramPacket class are: byte[] getData() Returns the data buffer. int getLength() Returns the length of the data to be sent or the length of the data received. void setData(byte[] buf) Sets the data buffer for this packet. void setLength(int length) Sets the length for this packet. The class DatagramSocket supports various methods that can be used for transmitting or receiving data a datagram over the network. The two key methods are: void send(DatagramPacket p) Sends a datagram packet from this socket. void receive(DatagramPacket p) Receives a datagram packet from this socket.

A simple UDP server program that waits for client's requests and then accepts the message (datagram) and sends back the same message is given below. Of course, an extended server program can manipulate client's messages/request and send a new message as a response.

Packet Types

802.11 traffic includes data packets, which are the packets used for normal network protocols; it also includes management packets and low-level control packets.

The 802.11 hardware on the network adapter filters all packets received, and delivers to the host

- all Unicast packets that are being sent to one of the addresses for that adapter, i.e. packets sent to that host on that network;
- all Multicast packets that are being sent to a Multicast address for that adapter, or all Multicast packets regardless of the address to which they're being sent (some network adapters can be configured to accept packets for specific Multicast addresses, others deliver all multicast packets to the host for it to filter);
- all Broadcast packets.

Data Packets
Data packets are often supplied to the packet capture mechanism, by default, as "fake" Ethernet packets, synthesized from the 802.11 header; you don't see the real 802.11 link-layer header.

The driver for the adapter will also send copies of transmitted packets to the packet capture mechanism, so that they will be seen by a capture program as well.

Non-data packets

You might have to capture in monitor mode to capture non-data packets. If not, you should capture with 802.11 headers, as no "fake" Ethernet headers can be constructed for non-data frames.

Management Packets

Management packets are used by peer WLAN controllers to maintain a WLAN network, and as such is seldom of importance above OSI layer 2. They are discarded by most drivers, and hence they do not reach the packet capture mechanism. However, if adapter/driver supports this, you may capture such packets in "monitor mode" as discussed below.

Low-level Control Packets

Control packets are used by peer WLAN controllers to synchronize channel access within contending WLAN hardware, as well as to synchronize packet exchange between peers. It is seldom of importance above OSI layer 2. They are discarded by most drivers, and hence they do not reach the packet capture mechanism. However, if adapter/driver supports this, you may capture such packets in "monitor mode" as discussed below.

802.11 Filter (Modes)

802.11 adapters (or their drivers) will filter packets on the receiving side in several ways. This section will give an overview which mechanisms are used and if/how these filters can be disabled.

Channels (Frequencies)

802.11 uses radio frequencies in the range of 2412-2484 MHz; please note that not all frequencies are allowed to be used in all countries. 802.11 splits the available frequencies in 14 network channels, numbered 1-14 (-> 14 "wireless cables"). The frequency range of a channel partially overlaps with the next one, so the channels are therefore not independent. Channels 1, 6 and 11 have no overlap with each other; those three are the unofficial "standard" for wireless channel independence.

Since the frequency range that's unlicensed varies in each country some places may not have 14 channels. For example, Japan has #1-#14, Europe #1-#13 and the FCC in the US allows #1-#11.

The user has to choose which channel to use for the network adapter/access point. Traffic will only be sent to (or received from) that channel.

This filtering can't be disabled. However, special measuring network adapters *might* be available to capture on multiple channels at once.

SSID/ESSID (Network Name)

In normal operation the user sets the SSID (Service Set Identifier) at the access point and the network adapter. If multiple access points use the same SSID it's called an ESSID (Extended SSID). A network adapter will then filter based on this SSID and hand over packets to the host only of the same SSID as it's currently set itself to.

Monitor mode

In monitor mode the SSID filter mentioned above is disabled and *all* packets of *all* SSID's from the currently selected channel are captured.

Even in promiscuous mode, an 802.11 adapter will only supply to the host packets of the SSID the adapter has joined, assuming promiscuous mode works at all; even if it "works", it might only supply to the host the same packets that would be seen in non-promiscuous mode. Although it can receive, at the radio level, packets on other SSID's, it will not forward them to the host.

For Linux System:
Whether you will be able to capture in monitor mode depends on the card and driver you're using. Newer Linux kernels support the mac80211 framework for 802.11 adapter drivers, which most if not all newer drivers, and some older drivers, supports. See the linuxwireless.org list of 802.11 adapter drivers for some information on what 802.11 drivers are available and whether they support monitor mode; drivers listed as supporting cfg80211 and monitor mode should support enough of the mac80211 framework to allow monitor mode to be controlled in a standard fashion. For additional information, see:

- the seattlewireless.net Linux Drivers page;
- this page of Linux 802.11b information for details on 802.11b wireless cards, including information on the chips they use;
- this page of Linux 802.11b+/a/g/n information for details on 802.11b+, 802.11a, 802.11g, and 802.11n wireless cards, including information on the chips they use;
- the aircrack-ng "What is the best wireless card to buy?" page;
- the aircrack-ng tutorial "Is My Wireless Card Compatible?";
- the aircrack-ng driver compatibility page;

- the LinuxWireless Drivers page and Devices pages.

In order to see 802.11 headers, you will have to capture in monitor mode. (XXX - true for all drivers?)

The easiest way to turn manually turn monitor mode on or off for an interface is with the airmon-ng script in aircrack-ng; your distribution may already have a package for aircrack-ng.

Note that the behavior of airmon-ng will differ between drivers that support the new mac80211 framework and drivers that don't. For drivers that support it, a command such as sudo airmon-ng start wlan0 will produce output such as

```
Interface   Chipset      Driver

 wlan0      Intel 4965 a/b/g/n   iwl4965 - [phy0]
         (monitor mode enabled on mon0)
```

The "monitor mode enabled on mon0" means that you must then capture on the "mon0" interface, *not* on the "wlan0" interface, to capture in monitor mode. To turn monitor mode off, you would use a command such as sudo airmon-ng stop mon0, not sudo airmon-ng stop wlan0.

For drivers that don't support the mac80211 framework, a command such as sudo airmon-ng start wlan0 will not report anything about a "mon0" device, and you will capture on the device you specified in the command. To turn monitor mode off, you would use a command such as sudo airmon-ng stop wlan0.

If you can't install airmon-ng, you will have to perform a more complicated set of commands, duplicating what airmon-ng would do. For adapters whose drivers support the new mac80211 framework, to capture in monitor mode create a monitor-mode interface for the adapter and capture on that; delete the monitor-mode interface afterwards. To do this in newer Linux distributions with the iw command, first run the command ifconfig-a to find out what interfaces already exist with names beginning with mon followed by a number. Then choose a number greater than all of the numbers for mon*N* devices; choose 0 if there are no mon*N* devices. Then run the command iw dev *interface* interface add mon*num* type monitor, where *interface* is the ifconfig name for the adapter and *num* is the number you chose. If that succeeds, bring up the interface with the command ifconfig mon*num* up, and capture on the mon*num* interface. When you are finished capturing, delete the monitor mode interface with the command iw dev mon*num* interface del

On Ubuntu 15.10 (and probably on earlier versions also), I find the easiest way to configure a monitor interface is to plug in your hardware and then run "ifconfig -a". Look at the output and then put entries in /etc/network/interfaces for any interfaces that are related to the hardware you are using and any entries for the monitor interfaces you or wireshark are going to create. You must put two entries in for each interface one for IPV4 and one for IV6 e.g.

```
iface mywificard0 inet manual
iface wywificard0 inet6 manual
```

This stops NetworkManager interfering with then. Here is an example of my interfaces file.

```
# interfaces(5) file used by ifup(8) and ifdown(8)
```

```
auto lo
iface lo inet loopback

# Disable network manager on interface names I want to use for monitoring
# related purposes
iface eth1 inet manual
iface mon0 inet manual
iface eth1 inet6 manual
iface mon0 inet6 manual
# Make sure Wireshark generated wifi interfaces are excluded as well
iface phy0.mon inet manual
iface phy1.mon inet manual
iface phy2.mon inet manual
iface phy3.mon inet manual
iface phy4.mon inet manual
iface phy5.mon inet manual
iface phy6.mon inet manual
iface phy7.mon inet manual
iface phy8.mon inet manual
iface phy0.mon inet6 manual
iface phy1.mon inet6 manual
iface phy2.mon inet6 manual
iface phy3.mon inet6 manual
iface phy4.mon inet6 manual
iface phy5.mon inet6 manual
iface phy6.mon inet6 manual
iface phy7.mon inet6 manual
iface phy8.mon inet6 manual
# Make sure that non monitor wifi interfaces on shared wiphys are ignored
# as well or I will miss some monitor packets
iface BigTenda inet manual
iface LittleBelkin inet manual
iface LittleTenda inet manual
iface Sinmax inet manual
iface Alfa inet manual
iface WNDA3200 inet manual
iface wlp6s0 inet manual
iface BigTenda inet6 manual
iface LittleBelkin inet6 manual
iface LittleTenda inet6 manual
iface Sinmax inet6 manual
iface Alfa inet6 manual
iface WNDA3200 inet manual
iface wlp6s0 inet6 manual
```

I use entries in /etc/udev/rules.d/70-persistent-net.rules to give my networking hardware friendly names. This is optional. Here is an example.

```
# This file was automatically generated by the /lib/udev/write_net_rules
# program, run by the persistent-net-generator.rules rules file.
#
# You can modify it, as long as you keep each rule on a single
# line, and change only the value of the NAME= key.

# PCI device 0x10ec:0x8168 (r8169)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:26:18:7f:d1:20", ATTR{dev_id}=="0x0", ATTR{type}=="1",
KERNEL=="eth*", NAME="eth0"

# PCI device 0x10ec:0x8169 (r8169)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:02:44:b9:0f:7f", ATTR{dev_id}=="0x0", ATTR{type}=="1",
KERNEL=="eth*", NAME="eth1"

# USB device 0x:0x (rt2800usb)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="c8:3a:35:c4:1c:76", ATTR{dev_id}=="0x0", ATTR{type}=="1",
KERNEL=="wlan*", NAME="BigTenda"

# USB device 0x:0x (rtl8192cu)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="ec:1a:59:0e:51:c3", ATTR{dev_id}=="0x0", ATTR{type}=="1",
KERNEL=="wlan*", NAME="LittleBelkin"

# USB device 0x:0x (rt2800usb)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="c8:3a:35:cc:bd:12", ATTR{dev_id}=="0x0", ATTR{type}=="1",
KERNEL=="wlan*", NAME="LittleTenda"

# USB device 0x:0x (rtl8187)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:0f:11:92:06:b2", ATTR{dev_id}=="0x0", ATTR{type}=="1",
KERNEL=="wlan*", NAME="Sinmax"
```

Here is an exmaple script that uses iw to set up a monitor interface. However wireshark will set up a monitor interface for you.

```
DEFAULT_WIPHY=phy0
WIPHY=${1:-$DEFAULT_WIPHY}
DEFAULT_MONIF=mon0
MONIF=${2:-$DEFAULT_MONIF}
DEFAULT_CHANNEL=11
CHANNEL=${3:-$DEFAULT_CHANNEL}

echo "Setting up wifi monitor interface on" $WIPHY
sudo iw phy $WIPHY interface add mon0 type monitor flags none control otherbss
echo "Bringing up $MONIF"
sudo ifconfig mon0 up promisc
echo "Setting wifi channel to" $CHANNEL
sudo iw dev mon0 set channel 11
```

The default setting of monitor flags on a newly created monitor interface is control|otherbss, Just to ensure that the flags are in a known state the above script clears all flags then set these two flags itself. At this time (April 2016) there is no way to read monitor flags back out the kernel.

The monitor interface should now be visible in ifconfig and in Wireshark.

Because the new kernel wifi architecture allows multiple virtual interfaces (vif) to share of physical interface (wiphy) it is essential to ensure that any other vif's sharing a wiphy with your monitor vif do not retune the radio to a different channel or initiate a scan. If this happens you will silently miss packets! "NetworkManager" is a major culprit in this respect.

The golden rule is if the radio is not tuned to the channel you will miss stuff!

For adapters whose drivers don't support the new mac80211 framework, see CaptureSetup/WLAN/Linux_non_mac80211.

Note that some adapters might be supported using the NdisWrapper mechanism. Unfortunately, if you use NdisWrapper, you have the same limitations as Windows for 802.11 capture, which usually means "no monitor mode and no 802.11 headers".