Packet Sniffing and Spoofing

Shaoquan Jiang

Review of Network Interface

- NIC (Network Interface Card) is a physical or logical device bridging a machine and a network

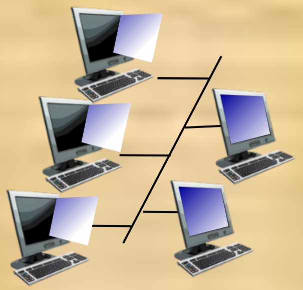
- Each NIC has a MAC address and is assigned an IP address
  - ifconfig –a
  -
    ```
    enp0s3    Link encap: Ethernet  HWaddr 08:00:27:db:4e:fc
              inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
    lo        Link encap: Local Loopback
              inet addr:127.0.0.1  Mask:255.0.0.0
    ```
- Every NIC on the network will hear all the packets coming to it
- NIC checks the destination MAC address for every packet. If this equals its own MAC address, then pass the packet to CPU; otherwise, discard it.

Packet Sniffer

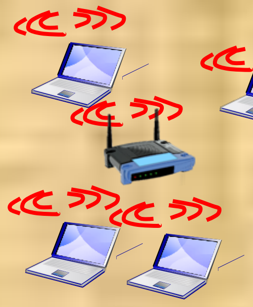
- In other words, NIC only accepts packets belonging to **itself**.
- Packet sniffer will make NIC work differently:
  - NIC will pass any packet it receives to CPU.
  - This requires the machine to be in a **promiscuous mode**

What can be Sniffed in the Promiscuous Mode

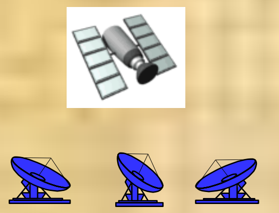
- Then, whose packets will come to the sniffing NIC?
- shared cable (or hub) or shared RF: we can sniffing all sharing users.



shared wire (e.g., cabled Ethernet)

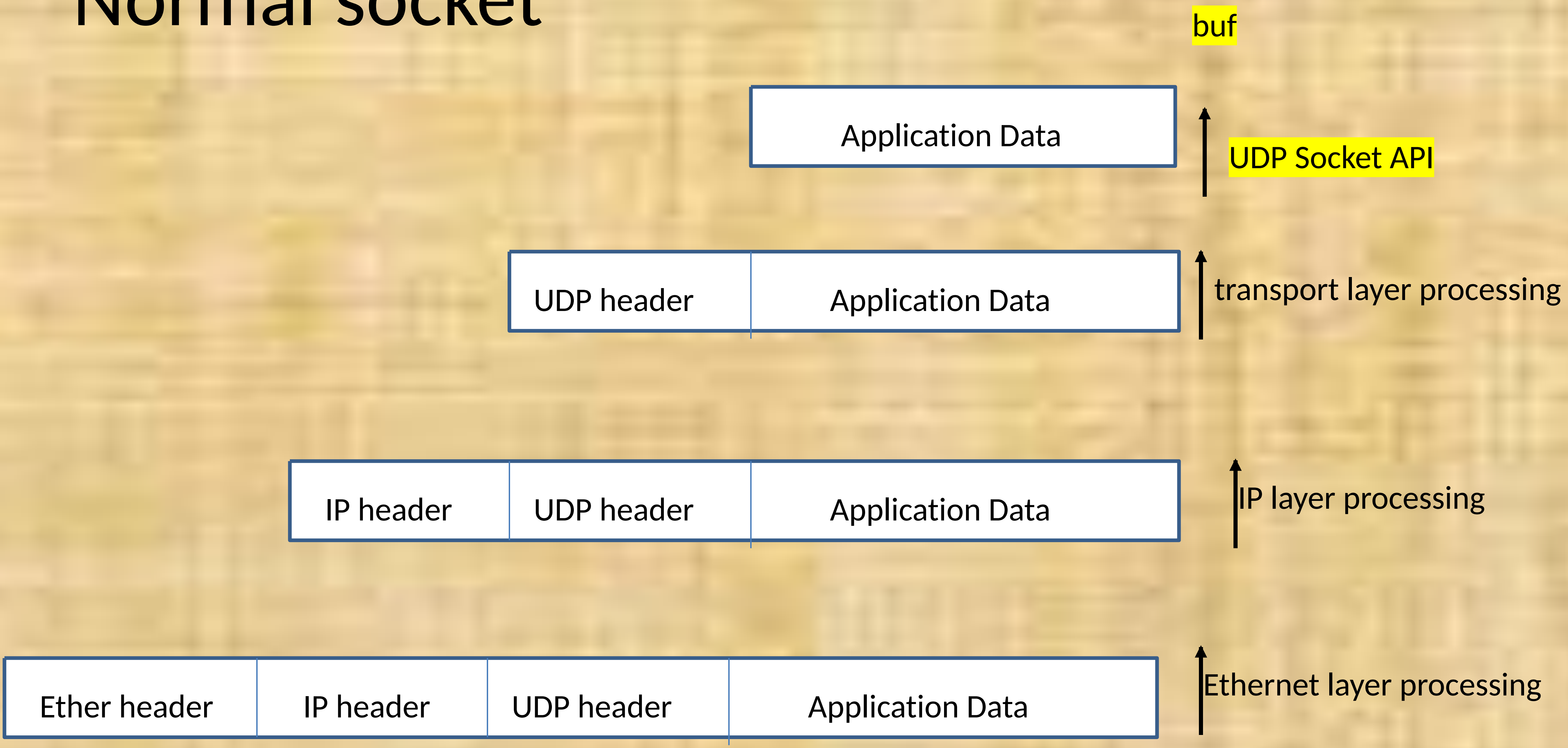


shared RF (e.g., 802.11 WiFi)



shared RF (satellite)

Normal socket



buf

Application Data

UDP Socket API

UDP header | Application Data

transport layer processing

IP header | UDP header | Application Data

IP layer processing

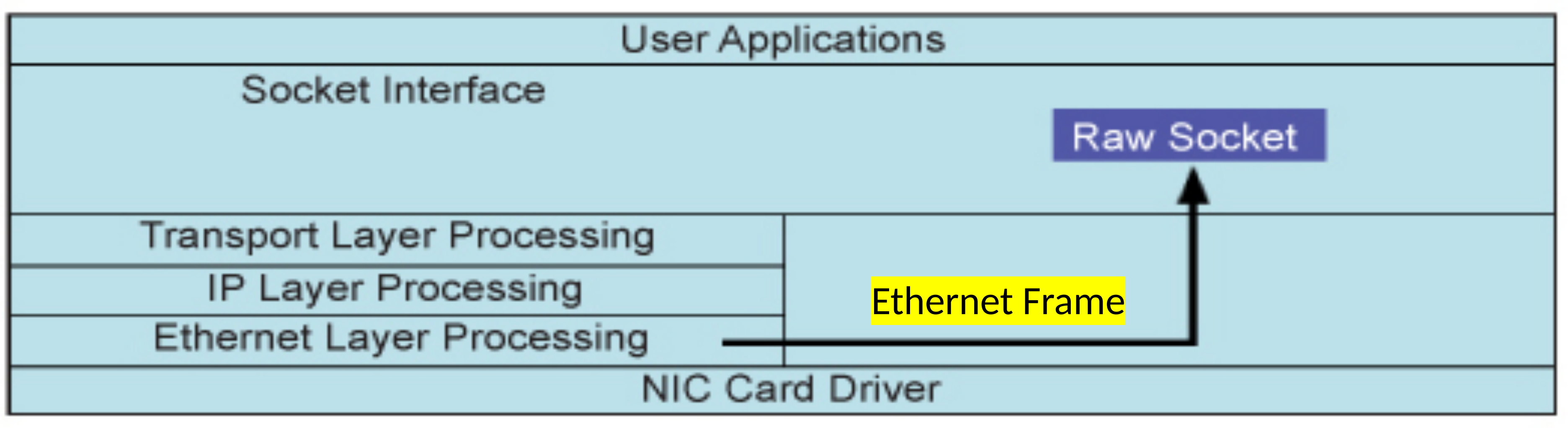
Ether header | IP header | UDP header | Application Data

Ethernet layer processing

Limitations

- The above UDP server socket only receives packets destined to the current machine.
  - We want to sniff packets destined to other machines.
- buf for the above socket only contains application data.
  - UDP, IP, Ethernet headers are stripped off.
  - We want these headers in order to forge a responding packet.
- So raw socket can overcome these issues.

Raw Socket



User Applications

Socket Interface

Raw Socket

Transport Layer Processing

IP Layer Processing

Ethernet Layer Processing

Ethernet Frame

NIC Card Driver

Raw Socket

- ❑ Ethernet frame is directly passed to socket and further to application.
- ❑ Ethernet frame include Ether header and the whole IP packet.

Ether header	IP header	UDP header	Application Data

Using Scapy for sniff and spoof

- VM in seedlab has aready installed scapy
- Use scapy in python program by adding one line:

  from scapy.all import *
- Scapy is written based on raw socket

Construct Packet

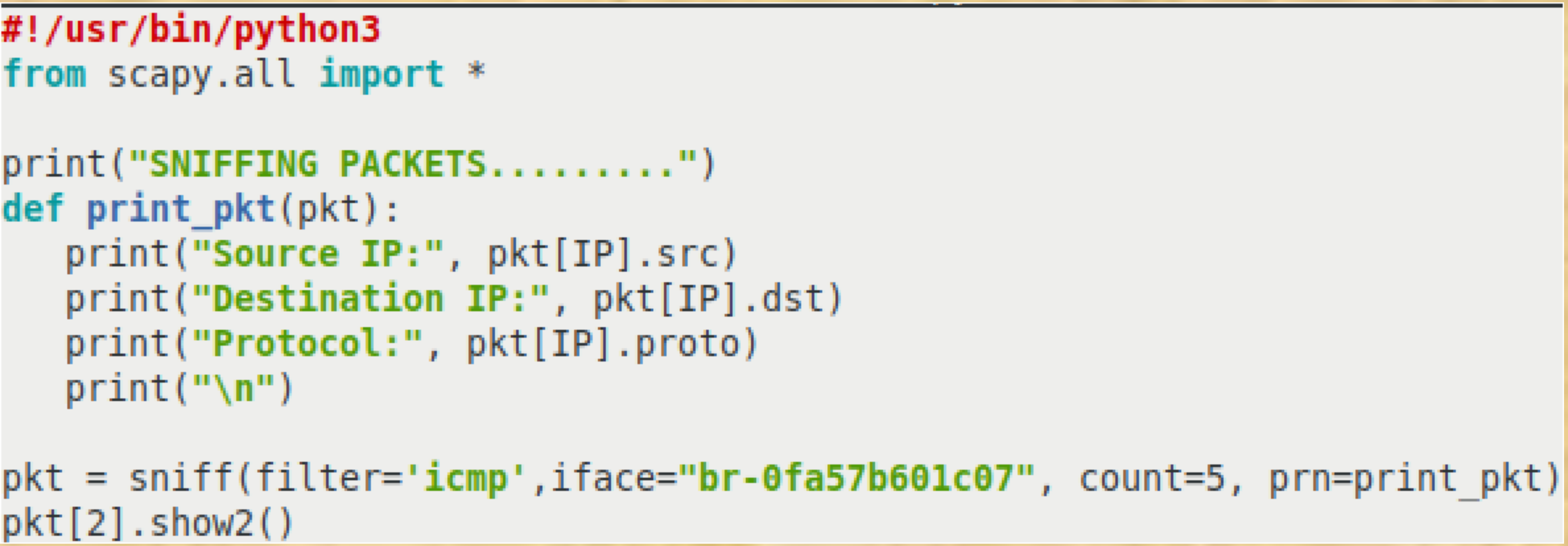
- Construct IP header:
  - check IP fields: ls(IP())
    - You will see many fields such as src, dst, ttl, id … any field in the ip header.
    - Define IP header using keys in IP fields: myip=IP(src=“10.0.2.15”)
- Construct ICMP packet:
  - check ICMP fields: ls(ICMP())
    - Similar to IP(), you will see many fields for ICMP() packet.
    - By default, ICMP() is an echo request packet: type=8.
  - define ICMP header: myicmp=ICMP(id=0x76)
- Construct UDP packet is similar: you can specify sport, dport.
- Form IP packet with myicmp as the payload, using operator /
    - pkt=myip/myicmp

Display the packet content

- Check packet content:
  - pkt.show() # show packet without system supplied fileds (i.e., checksum)
  - pkt.show2() # full details of packet. We **usually** use this.
  - pkt. summary() # show the summary of the packet.

Packing Sniffing Using Scapy

sniff.py



```
#!/usr/bin/python3
from scapy.all import *

print("SNIFFING PACKETS.........")
def print_pkt(pkt):
   print("Source IP:", pkt[IP].src)
   print("Destination IP:", pkt[IP].dst)
   print("Protocol:", pkt[IP].proto)
   print("\n")

pkt = sniff(filter='icmp',iface="br-0fa57b601c07", count=5, prn=print_pkt)
pkt[2].show2()
```

sniff() arguments

- count: Number of packets to capture. 0 means infinity.
- iface: Sniff only on the provided interface.
- prn: callback function on each packet.

  E.g., prn = lambda x: x.summary().
- timeout: Stop sniffing after a given time in seconds (default: None).
- filter: BPF filter

Ex. pkt=sniff(count=5, iface=“enp0s3”,prn=print_pkt, filter=“icmp”)

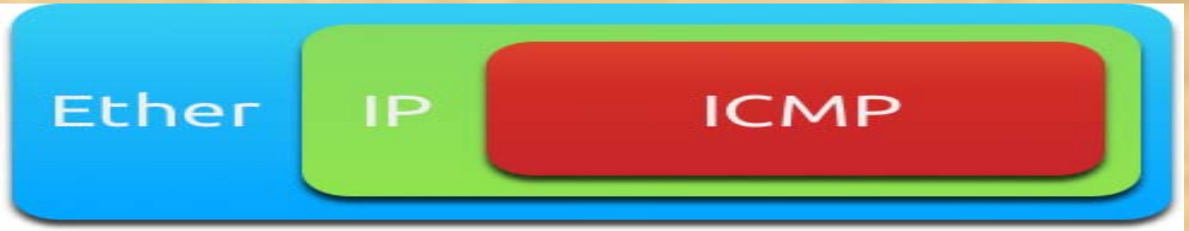
Filter expression

- Use Berkeley Packet Filter (BPF) syntax, specified as follow.
- type: host, net, port, portrange.
- dir: transmission direction such as src, dst.
- proto: protocol such as ether, ip, ip6, arp, tcp, udp.
- Operators are !/not, &&/and, II/or,
- Example:

  1. filter_exp="ip proto tcp && port 5500"
  2. filter_exp="host 10.0.2.6 && port 23"
  3. filter_exp="portrange 6000-6008 or net 10.0.2 or dst host 192.168.0.1"

pkts=sniff(filter=“icmp”)

- pkts is a list of packets. E.g., pkts[1] is the second packet.
- p=pkts[1] is a packet class, visualized as



- access subpacket: p[Ether], p[IP], p[ICMP]
  - p[IP] is a shortcut of p.getlayer(IP)

- p=p[Ether] is a packet class containing fields in Ether header and data field which is IP packet.
- use ls(p[Ether]) to see its fields.

pkt=sniff(filter=“icmp”)

```
>>> p[IP].show()                                   #   p=pkt[1]
###[ IP ]###
version     = 4
   ihl      = 5
   tos      = 0x0
   len      = 84
   id       = 452
   flags    =
   frag     = 0
   ttl      = 63
   proto             = icmp
   chksum            = 0xad2d
   src      = 192.168.0.1
   dst      = 10.0.2.15
   \options   \
###[ ICMP ]###
   type     = echo-reply
   code              = 0
   chksum            = 0x2683
   id       = 0xea7
   seq      = 0x1
###[ Raw ]###
   load     = '$Kp^F(\x05\x00\x08\t\n\x0b\x0c\...’
```

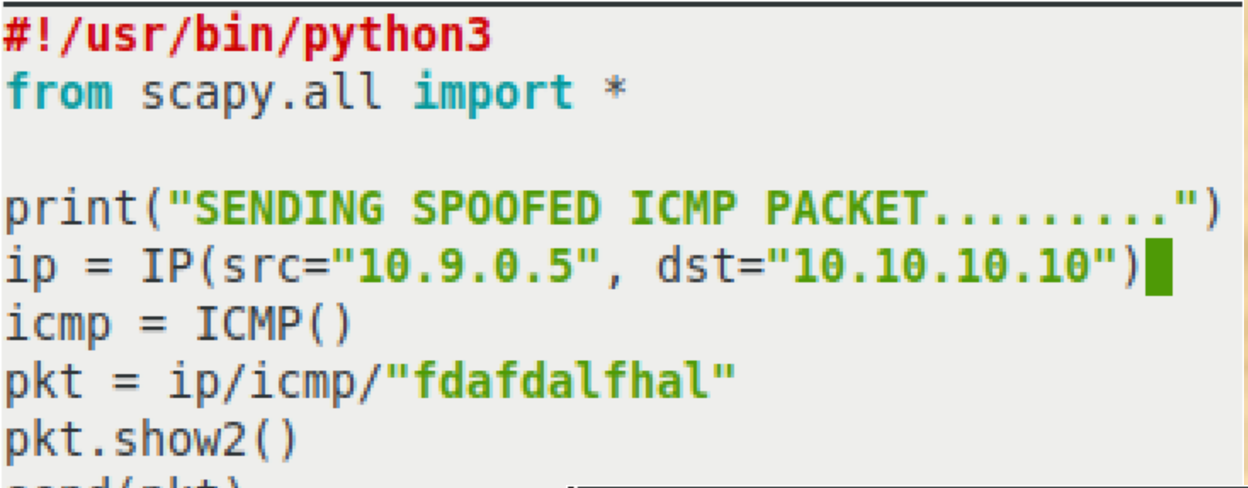
Send packet

- **send**(pkt, verbose=0, loop=0): pkt is an **IP** packet
  - iface: # interface for packet sending.
  - loop: # 1 for sending endlessly and 0 for sending once
  - pkt # it can be one or list of packets.
  - verbose # 1 for display the sending information and 0 for sending silently

- **sendp**(frame, iface="enp0s3")
  - **frame** is a link layer packet, starting with Ether header (using Ether() to construct this header).

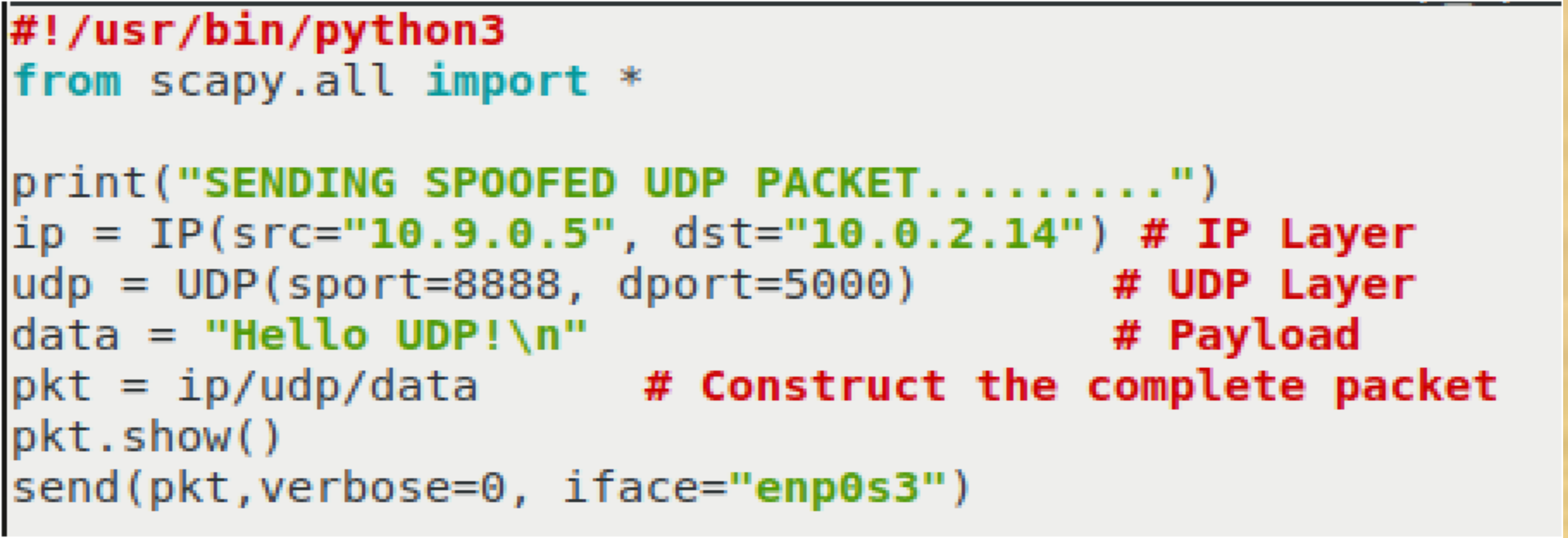
Spoofing ICMP & UDP Using Scapy

icmp_spoof.py



```
#!/usr/bin/python3
from scapy.all import *

print("SENDING SPOOFED ICMP PACKET.........")
ip = IP(src="10.9.0.5", dst="10.10.10.10")
icmp = ICMP()
pkt = ip/icmp/"fdafdalfhal"
pkt.show2()
send(pkt)
```

udp_spoof.py



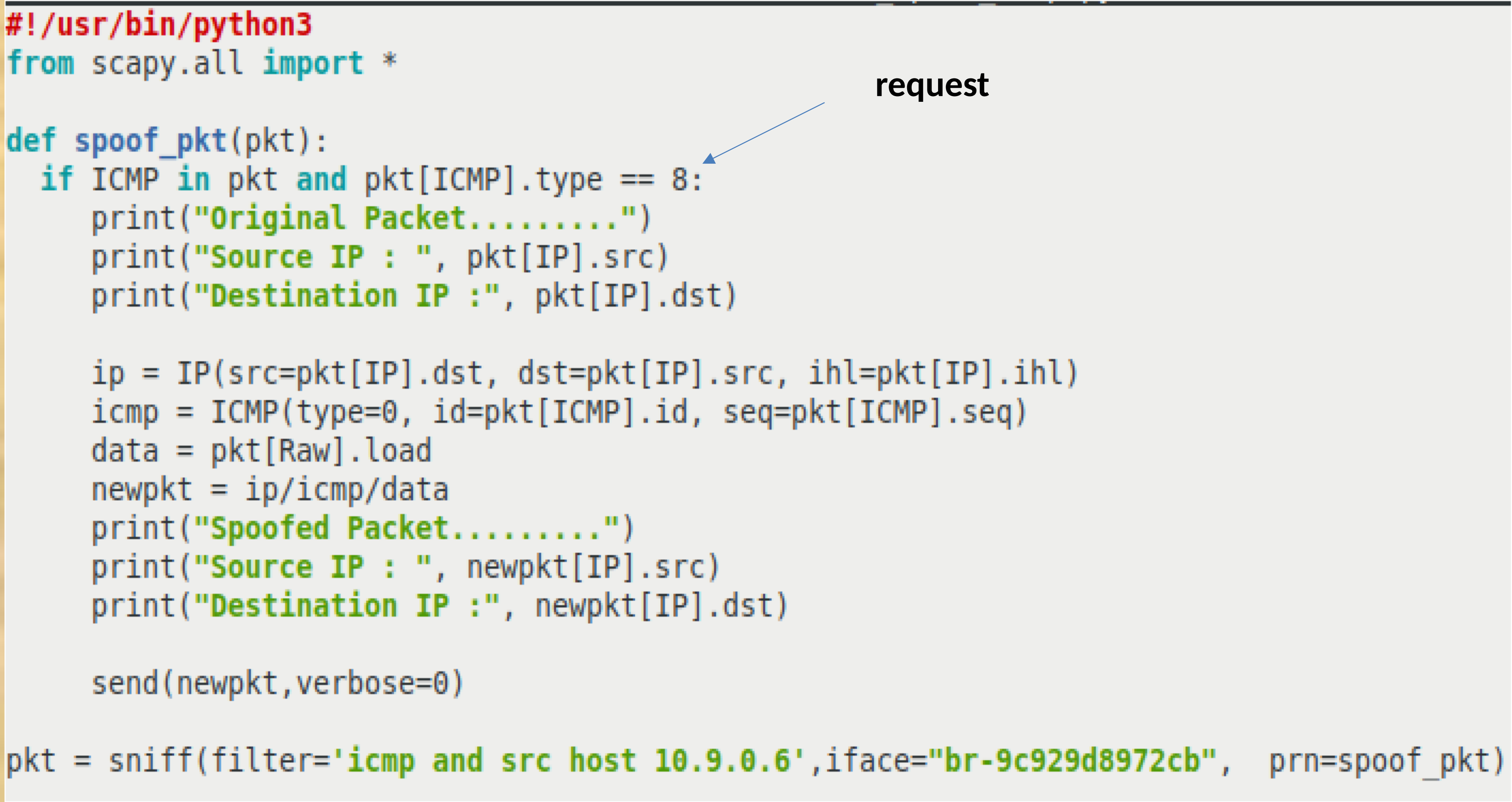
```
#!/usr/bin/python3
from scapy.all import *

print("SENDING SPOOFED UDP PACKET.........")
ip = IP(src="10.9.0.5", dst="10.0.2.14") # IP Layer
udp = UDP(sport=8888, dport=5000)        # UDP Layer
data = "Hello UDP!\n"                    # Payload
pkt = ip/udp/data      # Construct the complete packet
pkt.show()
send(pkt,verbose=0, iface="enp0s3")
```

Run a udp server at 10.0.2.14

$nc -lnuvp 5000

Sniffing and Then Spoofing Using Scapy

sniff_spoof_icmp.py

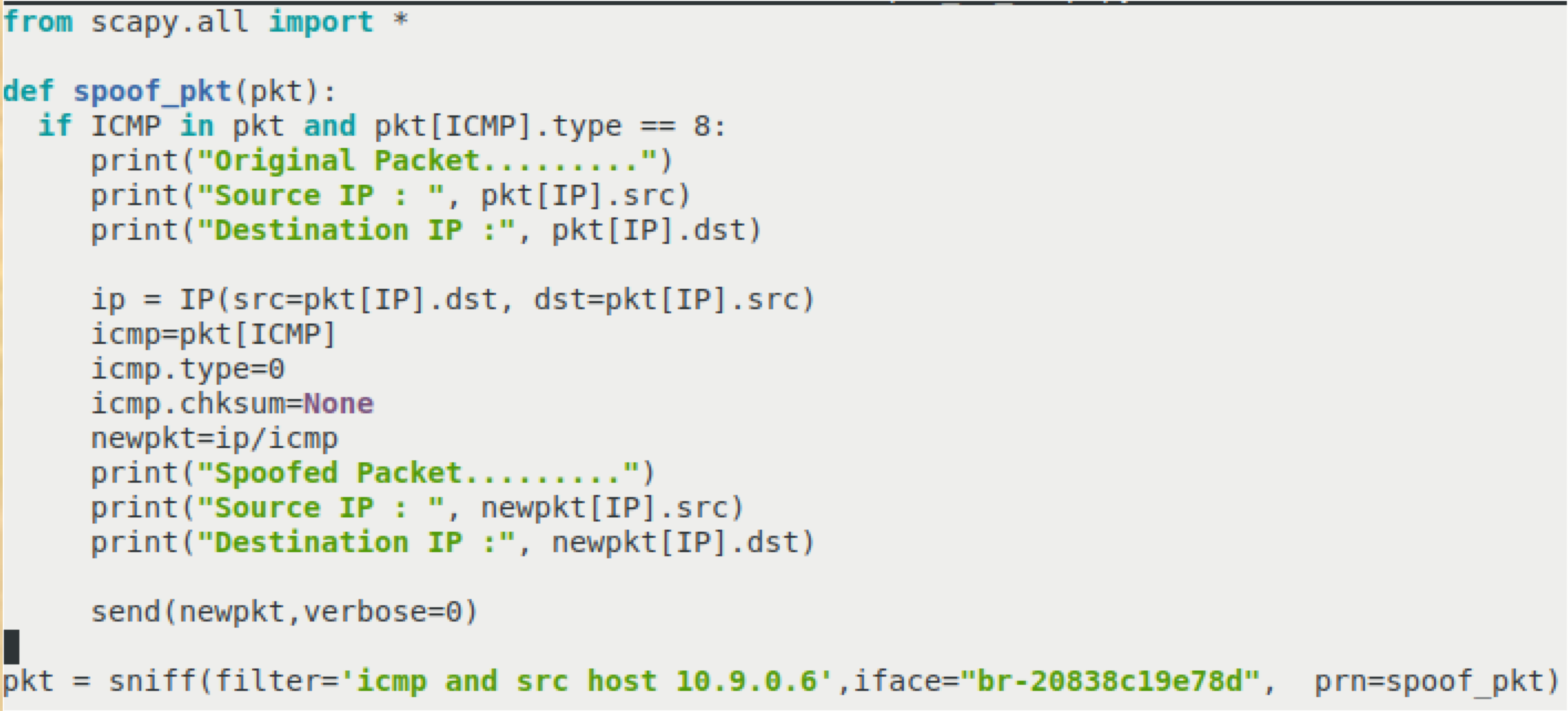


```
#!/usr/bin/python3
from scapy.all import *

def spoof_pkt(pkt):
  if ICMP in pkt and pkt[ICMP].type == 8:
     print("Original Packet.........")
     print("Source IP : ", pkt[IP].src)
     print("Destination IP :", pkt[IP].dst)

     ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
     icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
     data = pkt[Raw].load
     newpkt = ip/icmp/data
     print("Spoofed Packet.........")
     print("Source IP : ", newpkt[IP].src)
     print("Destination IP :", newpkt[IP].dst)

     send(newpkt,verbose=0)

pkt = sniff(filter='icmp and src host 10.9.0.6',iface="br-9c929d8972cb",  prn=spoof_pkt)
```

request

- On 10.9.0.6:
$ping 8.8.8.8
Check what is the reply?
- Turn on wireshark:
Which reply is from our program?

Alternative way to construct ICMP part.



```
from scapy.all import *

def spoof_pkt(pkt):
  if ICMP in pkt and pkt[ICMP].type == 8:
     print("Original Packet.........")
     print("Source IP : ", pkt[IP].src)
     print("Destination IP :", pkt[IP].dst)

     ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
     icmp=pkt[ICMP]
     icmp.type=0
     icmp.chksum=None
     newpkt=ip/icmp
     print("Spoofed Packet.........")
     print("Source IP : ", newpkt[IP].src)
     print("Destination IP :", newpkt[IP].dst)

     send(newpkt,verbose=0)

pkt = sniff(filter='icmp and src host 10.9.0.6',iface="br-20838c19e78d",  prn=spoof_pkt)
```