CIS 4360: Computer Security Fundamentals

Network Security

Viet Tung Hoang

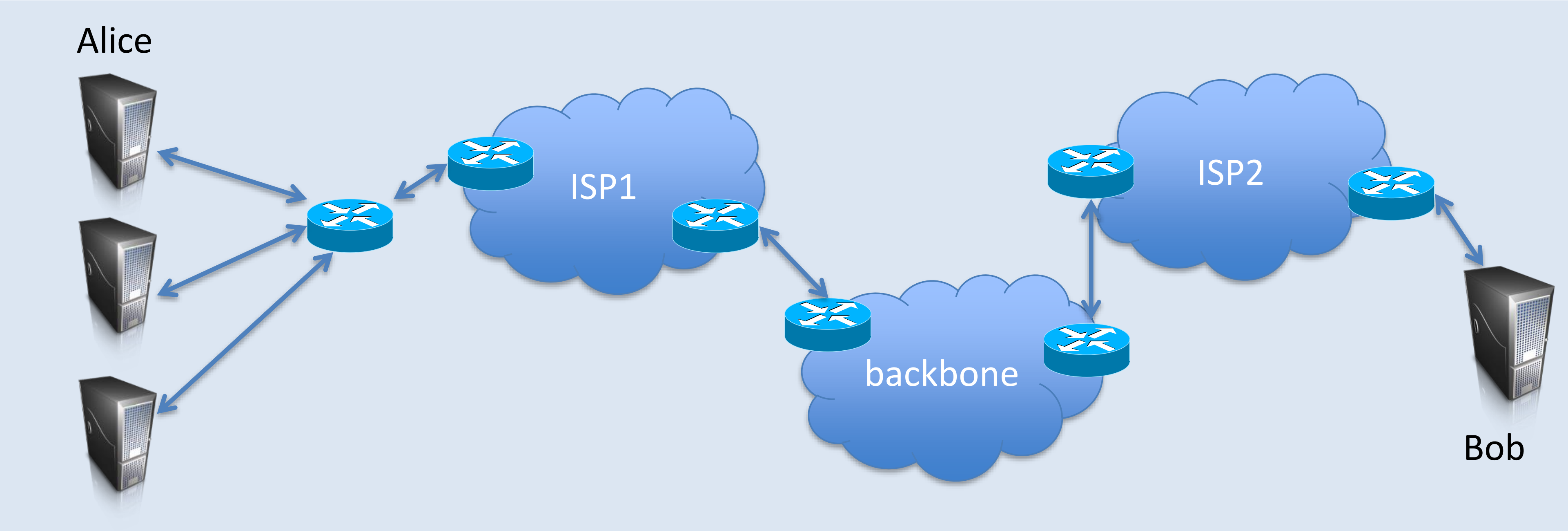
The slides are based on those of Prof. Stefano Tessaro, University of Washington and the book “Internet Security: A hands-on approach” by Kevin Du

Network Security

Looking at it from the right perspective

- Classical internet protocols are <u>not robust</u>
  - Design assumes benign behavior and correct implementations
- Typical attack vectors:
  - Malformed messages
  - Malformed protocol execution
  - Combined with faulty implementation / bad handling of unusual situations

Internet



Local area network (LAN)

Ethernet

802.11

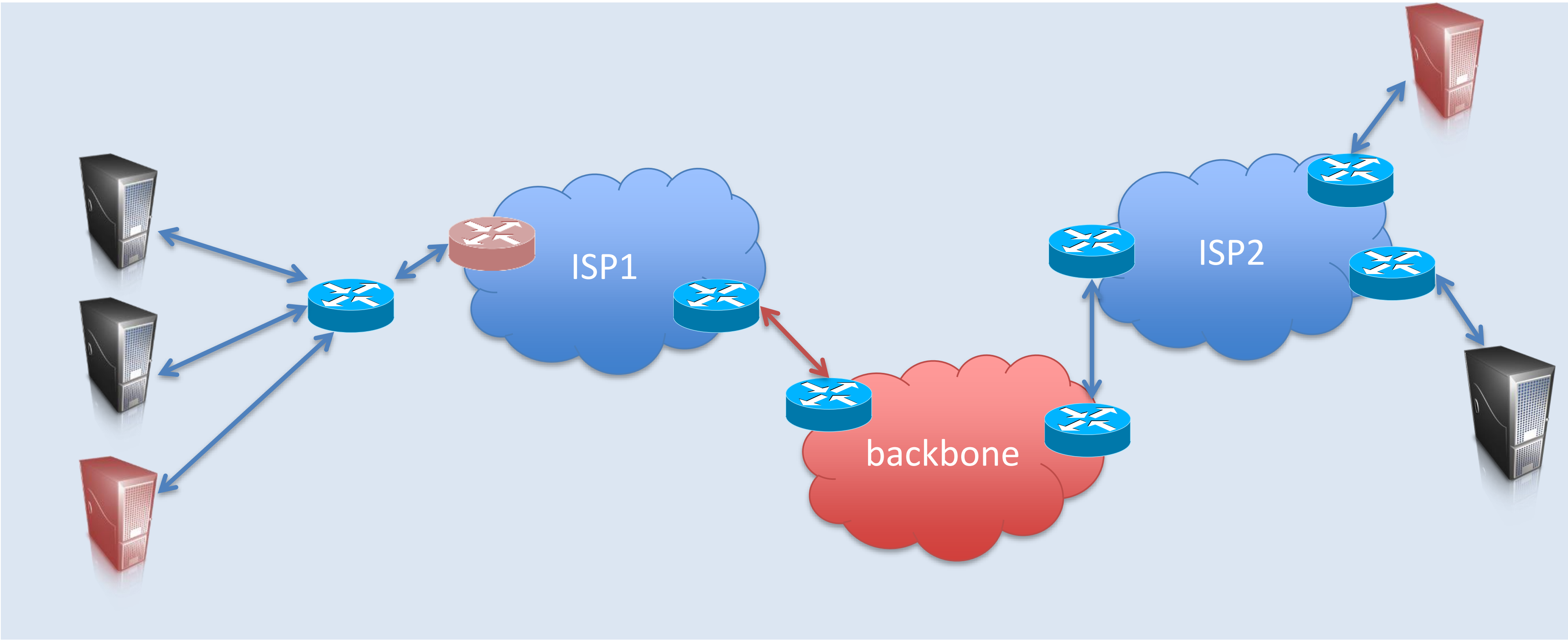
Internet

TCP/IP

BGP (border gateway protocol)

DNS (domain name system)

Internet threat models



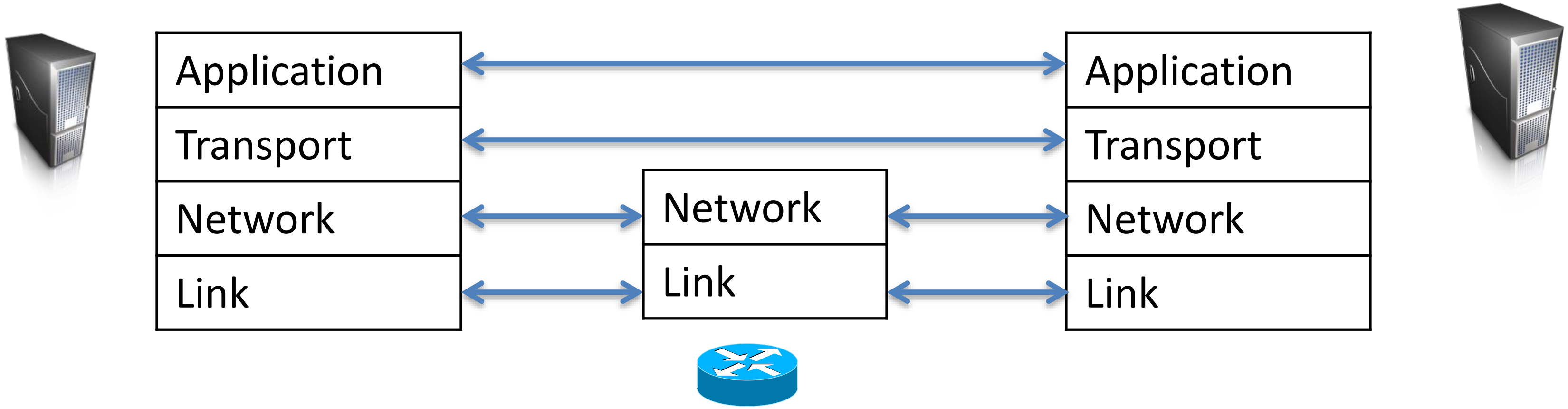
(1) Malicious hosts

(2) Subverted routers or links

(3) Malicious ISPs or backbone

Internet protocol stack

Application	HTTP, FTP, SMTP, SSH, etc.
Transport	TCP, UDP
Network	IP, ICMP, IGMP
Link	802x (802.11, Ethernet)

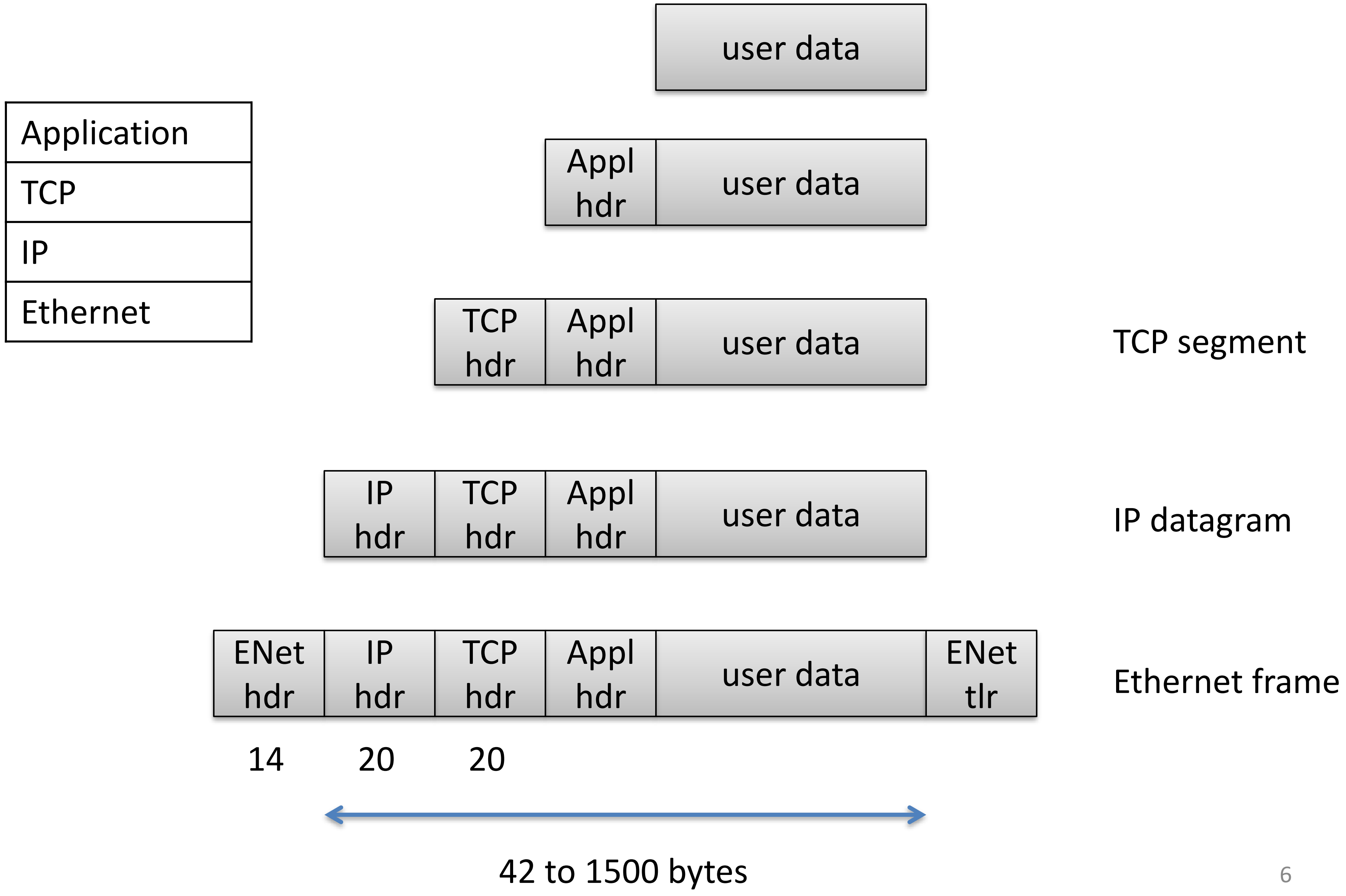


Application
Transport
Network
Link

Network
Link

Application
Transport
Network
Link

Internet protocol stack



Application

TCP

IP

Ethernet

user data

Appl hdr | user data

TCP hdr | Appl hdr | user data — TCP segment

IP hdr | TCP hdr | Appl hdr | user data — IP datagram

ENet hdr | IP hdr | TCP hdr | Appl hdr | user data | ENet tlr — Ethernet frame

14 20 20

42 to 1500 bytes

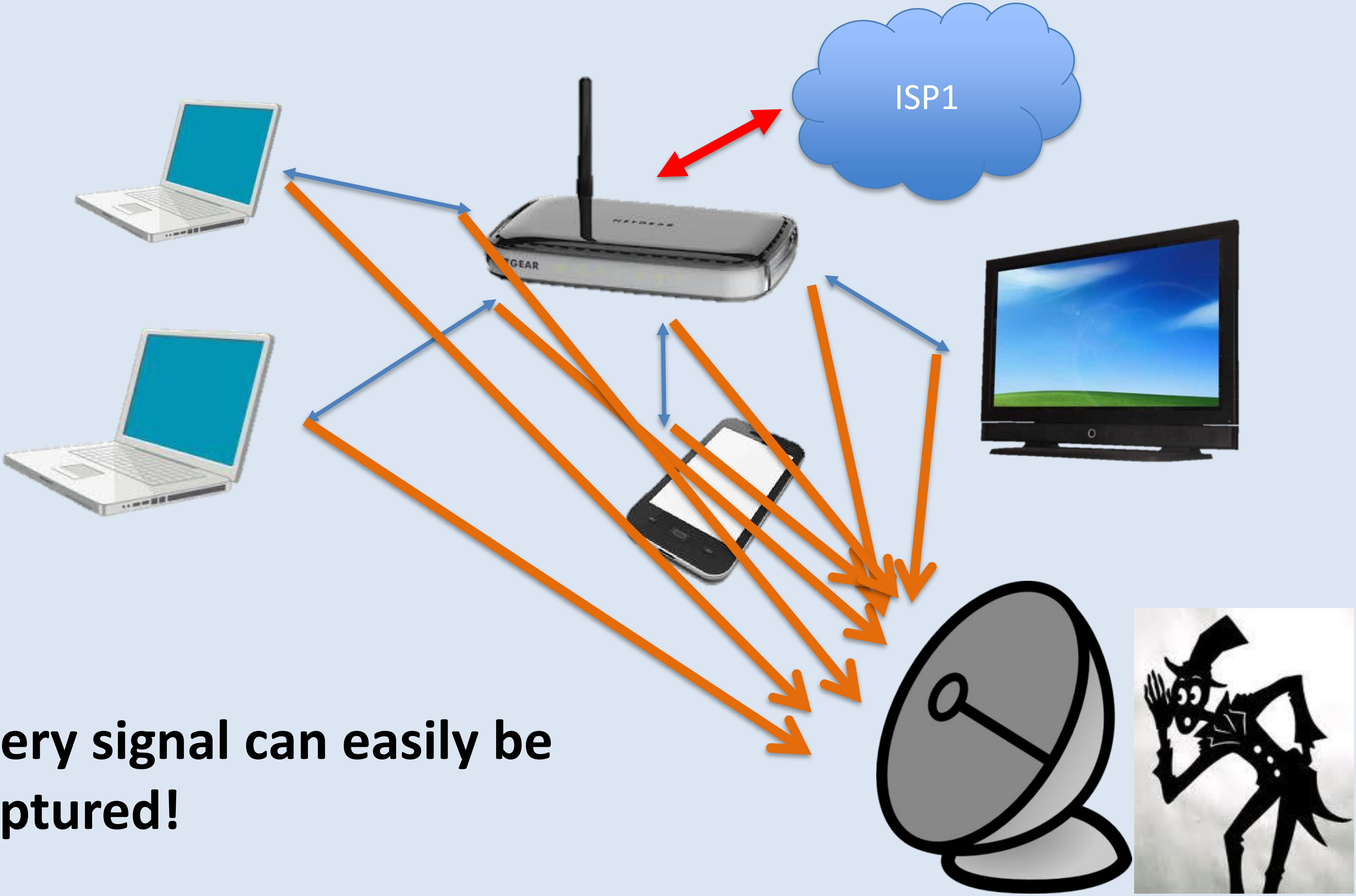
Agenda

1. **Link Layer Issues**
2. Network Layer Issues
3. Transport Layer Issues
4. Application Layer Issues

Link Layer – WiFi

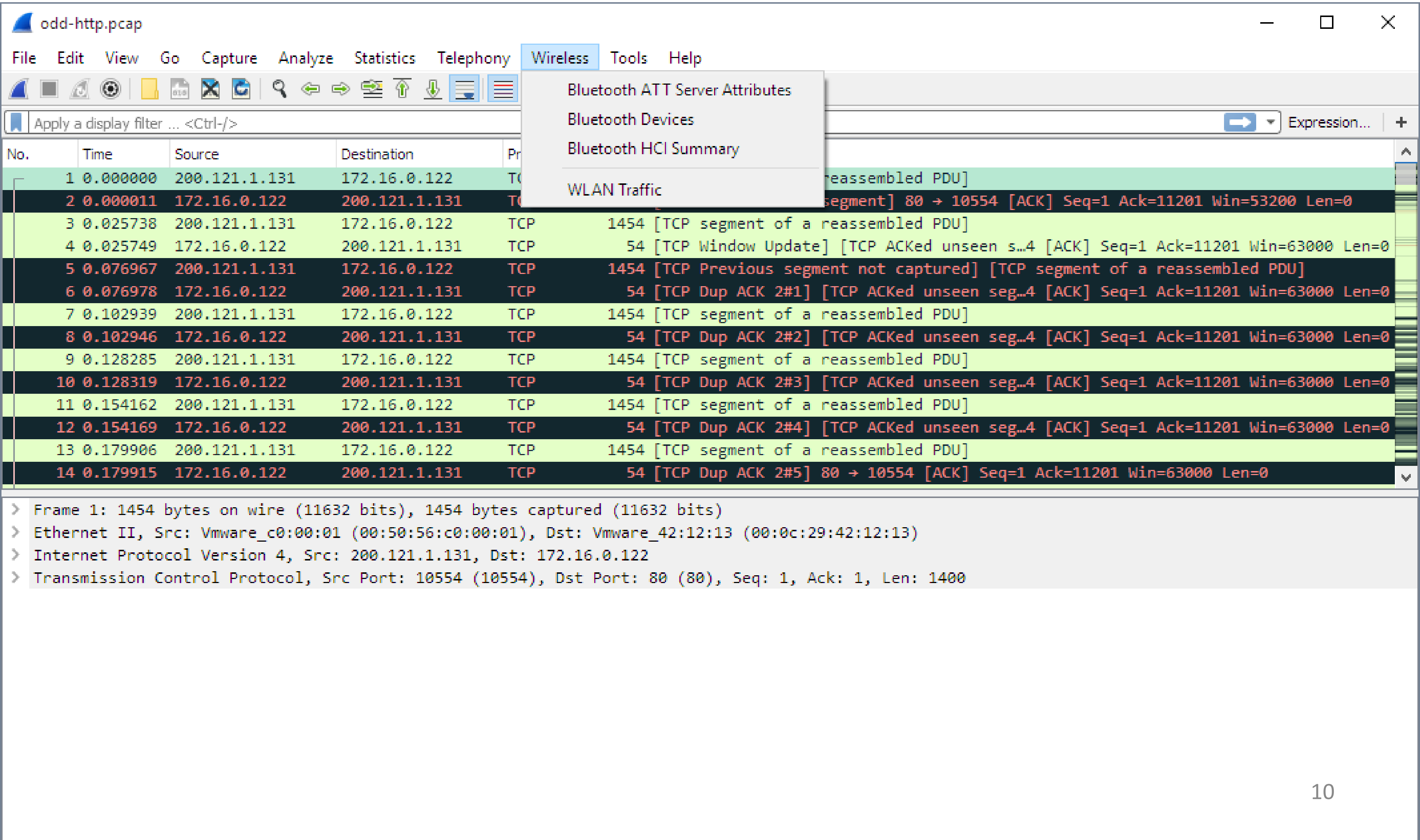
- Most common way to connect to a network

Packet Sniffing



Every signal can easily be captured!

Packet Sniffing: WireShark



```
No.  Time      Source         Destination    Pr
   1 0.000000  200.121.1.131  172.16.0.122   TC                       reassembled PDU]
   2 0.000011  172.16.0.122   200.121.1.131  TC                       egment] 80 → 10554 [ACK] Seq=1 Ack=11201 Win=53200 Len=0
   3 0.025738  200.121.1.131  172.16.0.122   TCP      1454 [TCP segment of a reassembled PDU]
   4 0.025749  172.16.0.122   200.121.1.131  TCP        54 [TCP Window Update] [TCP ACKed unseen s…4 [ACK] Seq=1 Ack=11201 Win=63000 Len=0
   5 0.076967  200.121.1.131  172.16.0.122   TCP      1454 [TCP Previous segment not captured] [TCP segment of a reassembled PDU]
   6 0.076978  172.16.0.122   200.121.1.131  TCP        54 [TCP Dup ACK 2#1] [TCP ACKed unseen seg…4 [ACK] Seq=1 Ack=11201 Win=63000 Len=0
   7 0.102939  200.121.1.131  172.16.0.122   TCP      1454 [TCP segment of a reassembled PDU]
   8 0.102946  172.16.0.122   200.121.1.131  TCP        54 [TCP Dup ACK 2#2] [TCP ACKed unseen seg…4 [ACK] Seq=1 Ack=11201 Win=63000 Len=0
   9 0.128285  200.121.1.131  172.16.0.122   TCP      1454 [TCP segment of a reassembled PDU]
  10 0.128319  172.16.0.122   200.121.1.131  TCP        54 [TCP Dup ACK 2#3] [TCP ACKed unseen seg…4 [ACK] Seq=1 Ack=11201 Win=63000 Len=0
  11 0.154162  200.121.1.131  172.16.0.122   TCP      1454 [TCP segment of a reassembled PDU]
  12 0.154169  172.16.0.122   200.121.1.131  TCP        54 [TCP Dup ACK 2#4] [TCP ACKed unseen seg…4 [ACK] Seq=1 Ack=11201 Win=63000 Len=0
  13 0.179906  200.121.1.131  172.16.0.122   TCP      1454 [TCP segment of a reassembled PDU]
  14 0.179915  172.16.0.122   200.121.1.131  TCP        54 [TCP Dup ACK 2#5] 80 → 10554 [ACK] Seq=1 Ack=11201 Win=63000 Len=0
```

```
> Frame 1: 1454 bytes on wire (11632 bits), 1454 bytes captured (11632 bits)
> Ethernet II, Src: Vmware_c0:00:01 (00:50:56:c0:00:01), Dst: Vmware_42:12:13 (00:0c:29:42:12:13)
> Internet Protocol Version 4, Src: 200.121.1.131, Dst: 172.16.0.122
> Transmission Control Protocol, Src Port: 10554 (10554), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 1400
```

Packet Sniffing: WireShark

Need to use WireShark in “monitor mode”

- Sees every packet sent over a Wifi channel
- Easy to do in Mac OS but limited in Windows
- Mostly disallowed by network policies

Solution – WPA2 personal (WPA2-PSK)

- Device and access points share pre-shared secret key **PSK** (aka **PMK**, pairwise master key), derived from a passphrase and SSID
- Upon connect, 4-way handshake protocol generates temporary session key **PTK**
- Encrypts with key = **PTK**

Agenda

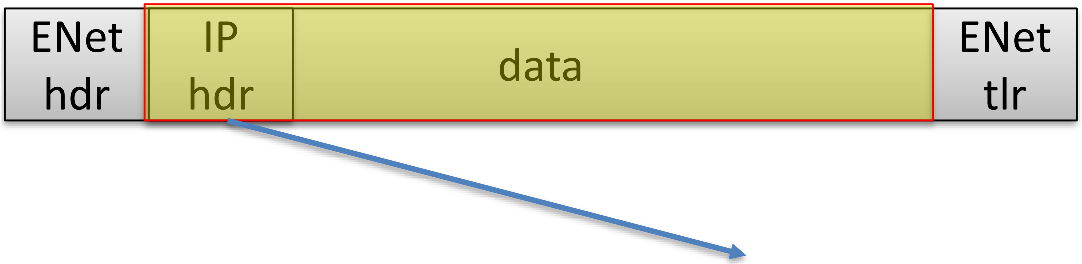
1. Link Layer Issues
2. **Network Layer Issues**
3. Transport Layer Issues
4. Application Layer Issues

IP protocol (IPv4)

Goal: The IP protocol is used to relay packets between two hosts, each assigned a corresponding IP address.

- Connectionless
  - no state, packets have no ordering guarantees
- Unreliable
  - no guarantees, packets may be dropped
- No integrity

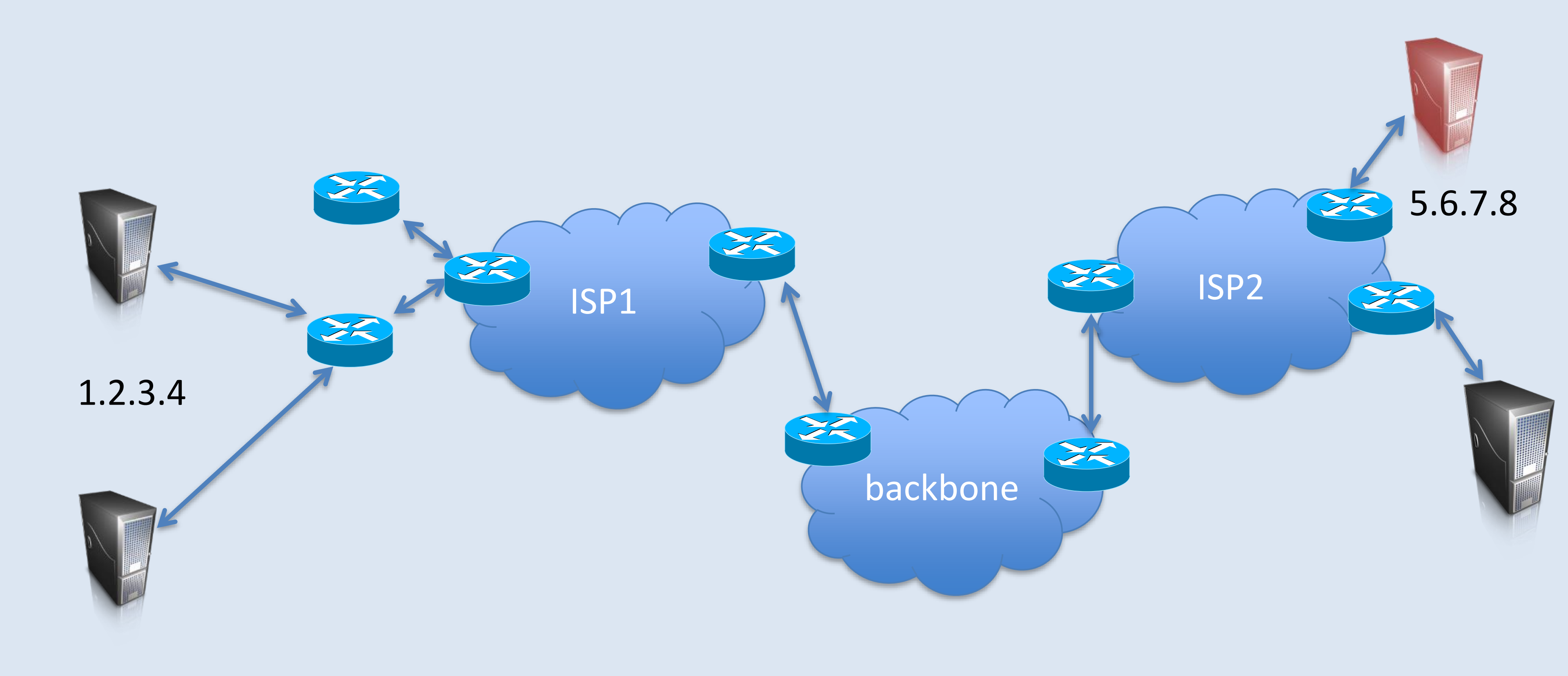
IPv4



Example: Ethernet frame containing IP datagram

<table>
<tr><td>4-bit
version</td><td>4-bit
hdr len</td><td>8-bit
type of service</td><td colspan="2">16-bit
total length (in bytes)</td></tr>
<tr><td colspan="3">16-bit
identification</td><td>3-bit
flags</td><td>13-bit
fragmentation offset</td></tr>
<tr><td colspan="2">8-bit
time to live (TTL)</td><td>8-bit
protocol</td><td colspan="2">16-bit
header checksum</td></tr>
<tr><td colspan="5">32-bit
source IP address</td></tr>
<tr><td colspan="5">32-bit
destination IP address</td></tr>
<tr><td colspan="5">options (optional)</td></tr>
</table>

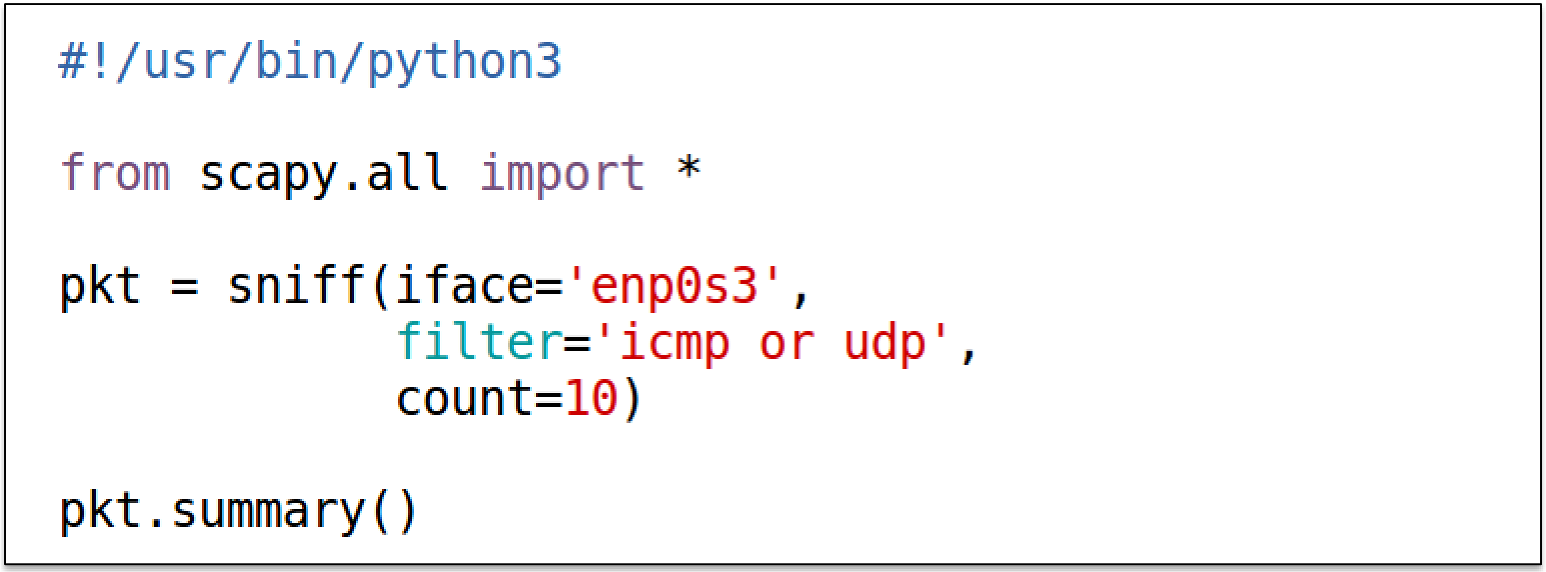
Security issues with IP



Basic issues:

- Anyone can talk to anyone
- No source address authentication in general (spoofing)

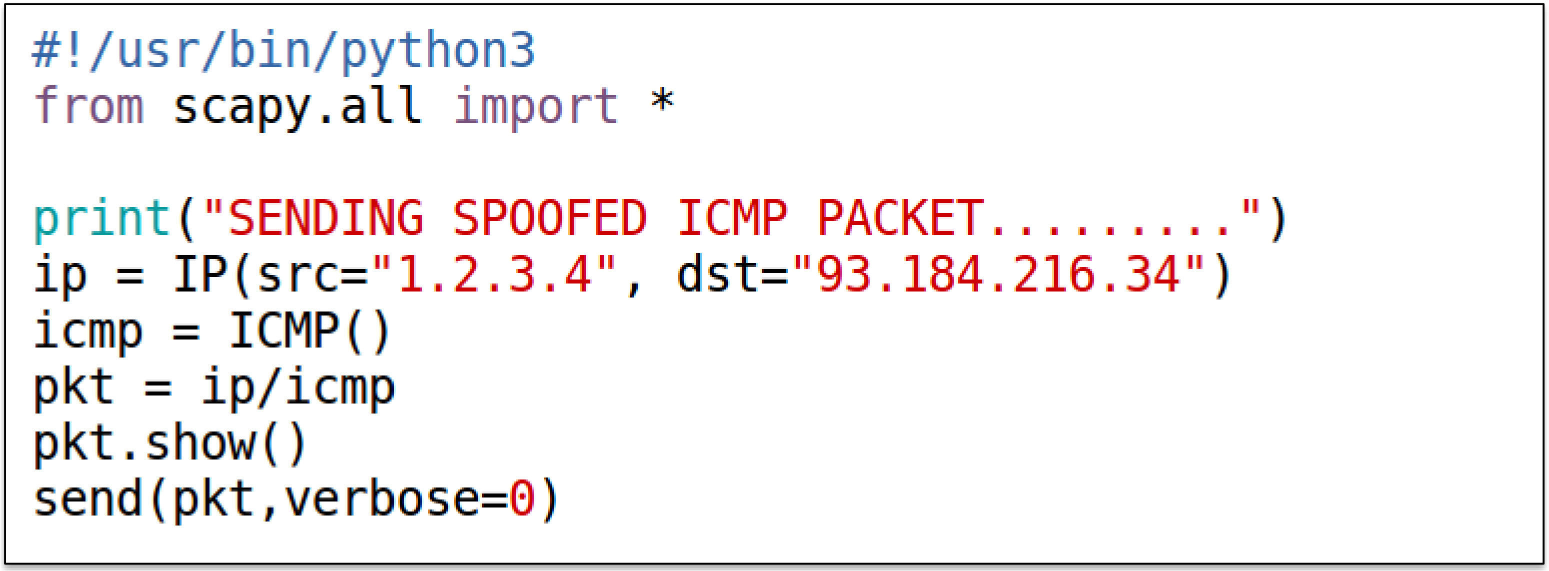
Automate Sniffing and Spoofing: Scapy



```python
#!/usr/bin/python3

from scapy.all import *

pkt = sniff(iface='enp0s3',
            filter='icmp or udp',
            count=10)

pkt.summary()
```

Source: “Internet Security: A hands-on approach” by Kevin Du

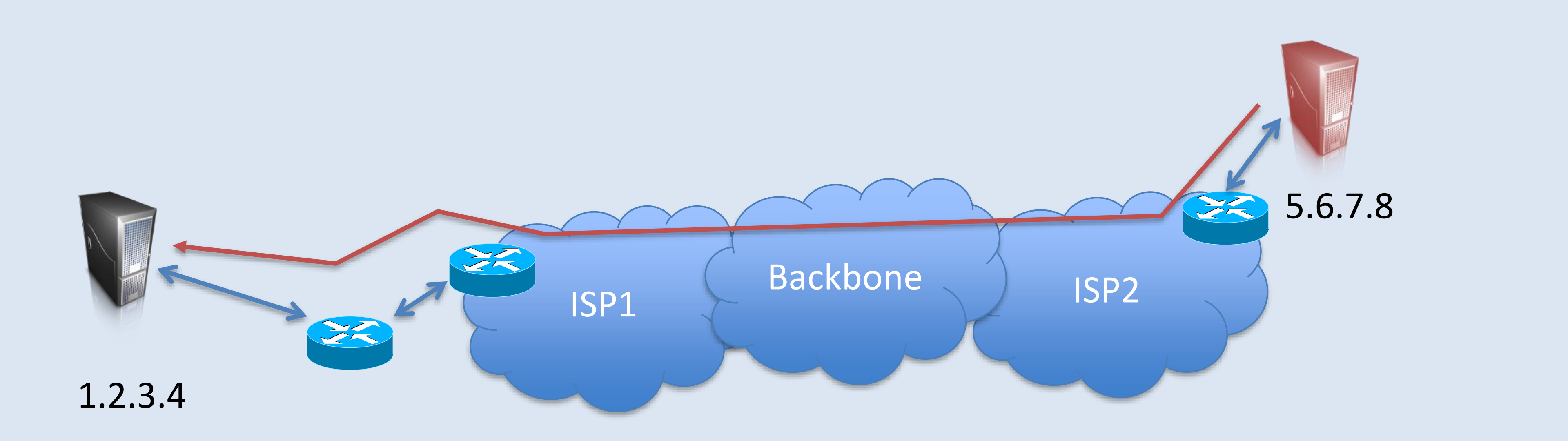
Automate Sniffing and Spoofing: Scapy



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

print("SENDING SPOOFED ICMP PACKET.........")
ip = IP(src="1.2.3.4", dst="93.184.216.34")
icmp = ICMP()
pkt = ip/icmp
pkt.show()
send(pkt,verbose=0)
```

Source: “Internet Security: A hands-on approach” by Kevin Du

Denial of Service (DoS) attacks



Goal: prevent legitimate users from accessing victim (1.2.3.4)

Example: ICMP ping flood

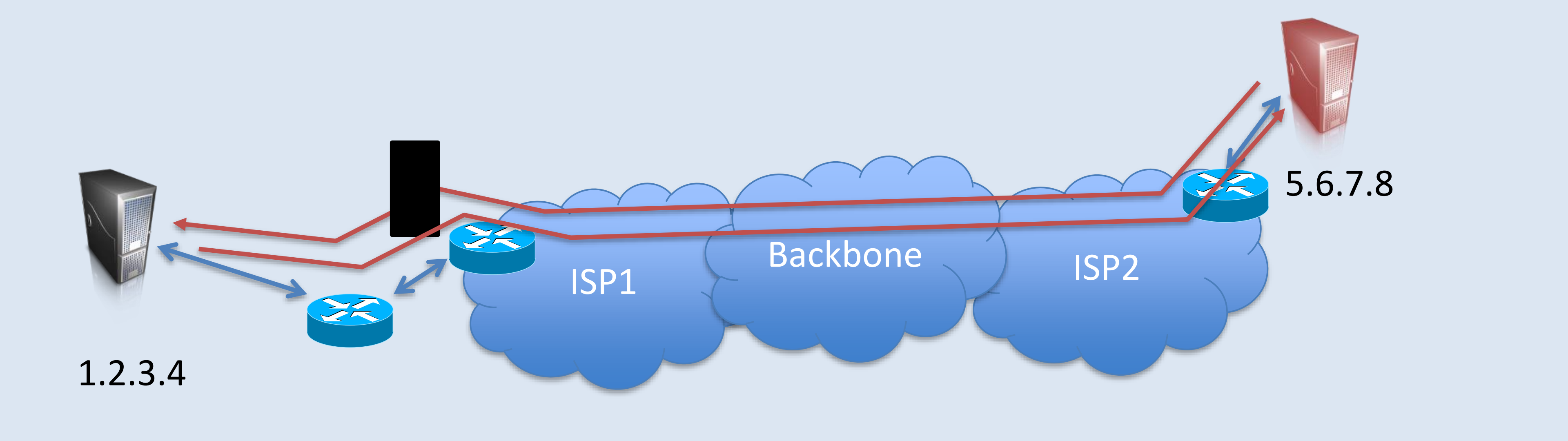
ICMP = Internet Control Message Protocol, used to relay control / error / diagnostic message, on top of IP

```
$ ping www.example.com
PING www.example.com (93.184.216.119): 56 data bytes
64 bytes from 93.184.216.119: icmp_seq=0 ttl=56 time=11.632 ms
64 bytes from 93.184.216.119: icmp_seq=1 ttl=56 time=11.726 ms
64 bytes from 93.184.216.119: icmp_seq=2 ttl=56 time=10.683 ms
64 bytes from 93.184.216.119: icmp_seq=3 ttl=56 time=9.674 ms

--- www.example.com ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 9.674/10.929/11.726/0.831 ms
```

Send ICMP “echo” message

- Echo request ("ping”): ICMP message whose data is expected to be received back in an echo reply ("pong”)
- Host must respond to all echo requests with an echo reply containing the exact data received in the request message.

Denial of Service (DoS) attacks

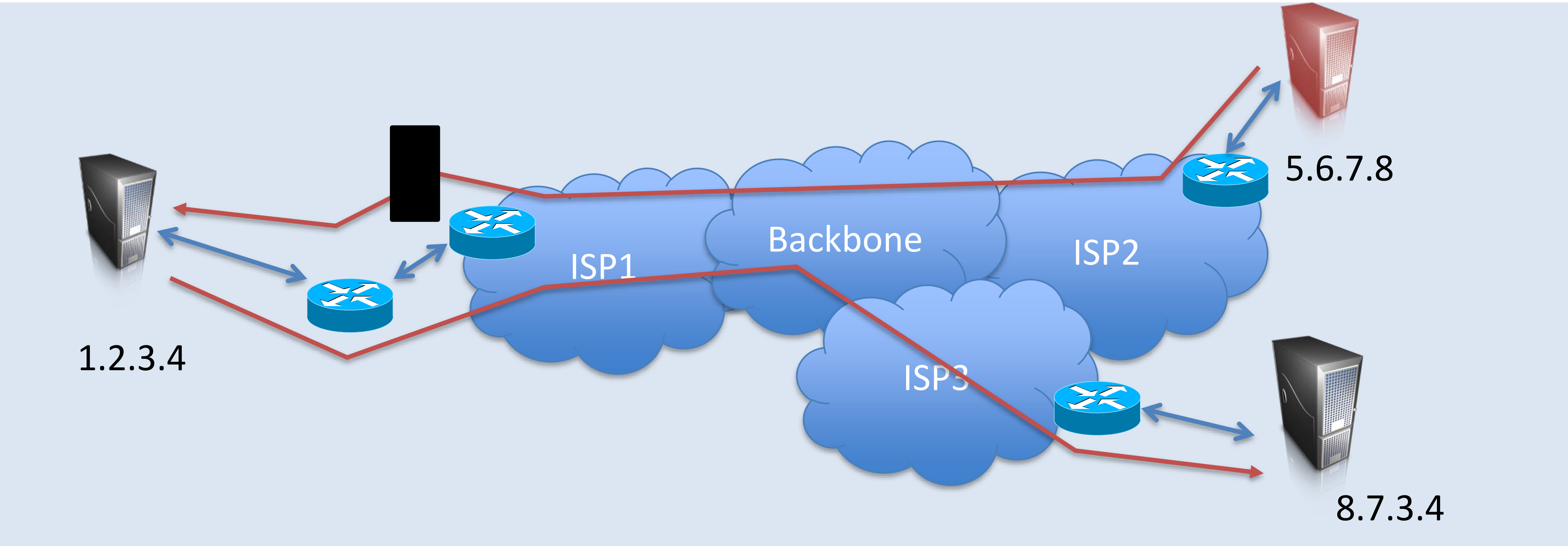


Goal: prevent legitimate users from accessing victim (1.2.3.4)

Possible attack: "ICMP ping flood"

- Attacker sends ICMP pings as fast as possible to victim
- When will this work as a DoS? Attacker resources > victim's
- How can this be prevented? Ingress filtering near victim

Denial of Service (DoS) attacks



How can attacker avoid ingress filtering?

Attacker can send packet with fake source IP

This is a so-called "spoofed" packet

Packet will get routed correctly, but replies will not

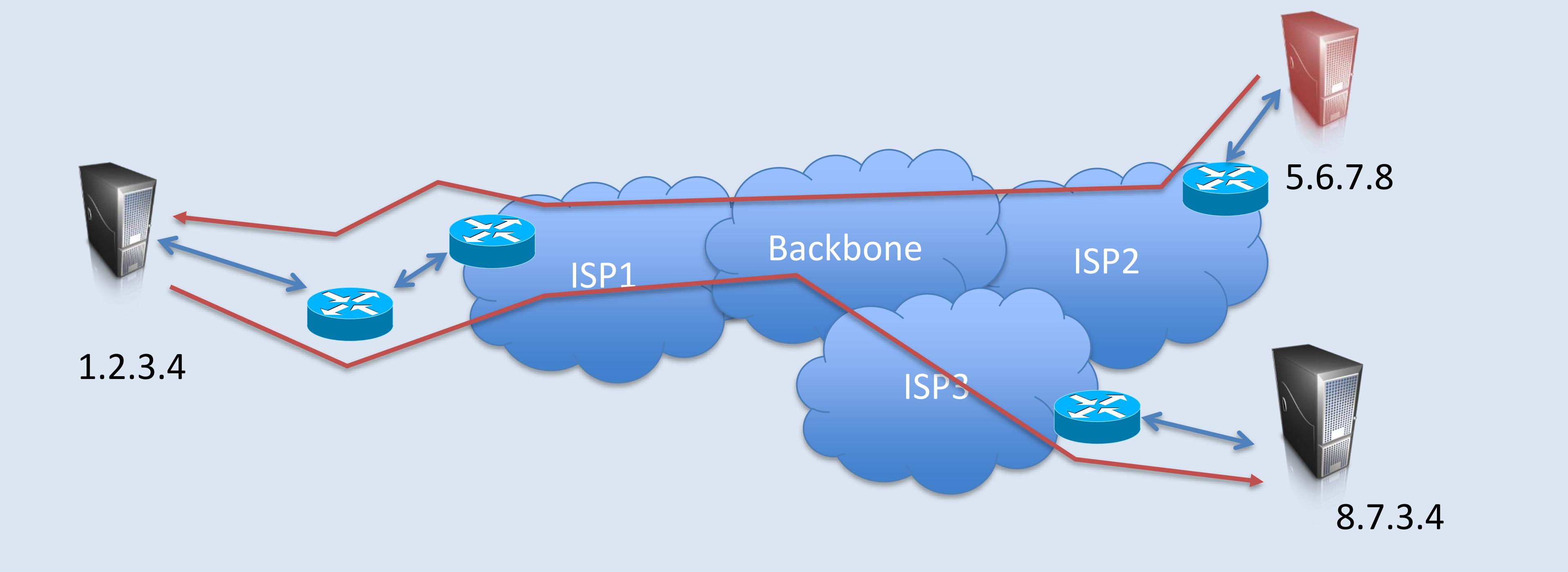
Send IP packet with

source: 8.7.3.4
dest: 1.2.3.4

from 5.6.7.8

Filter based on source now is incorrect!

DoS reflection attacks



Note a valid packet sends a reply to 8.7.3.4

- Attacker can bounce an attack against 8.7.3.4 off 1.2.3.4
- "Frame" 1.2.3.4

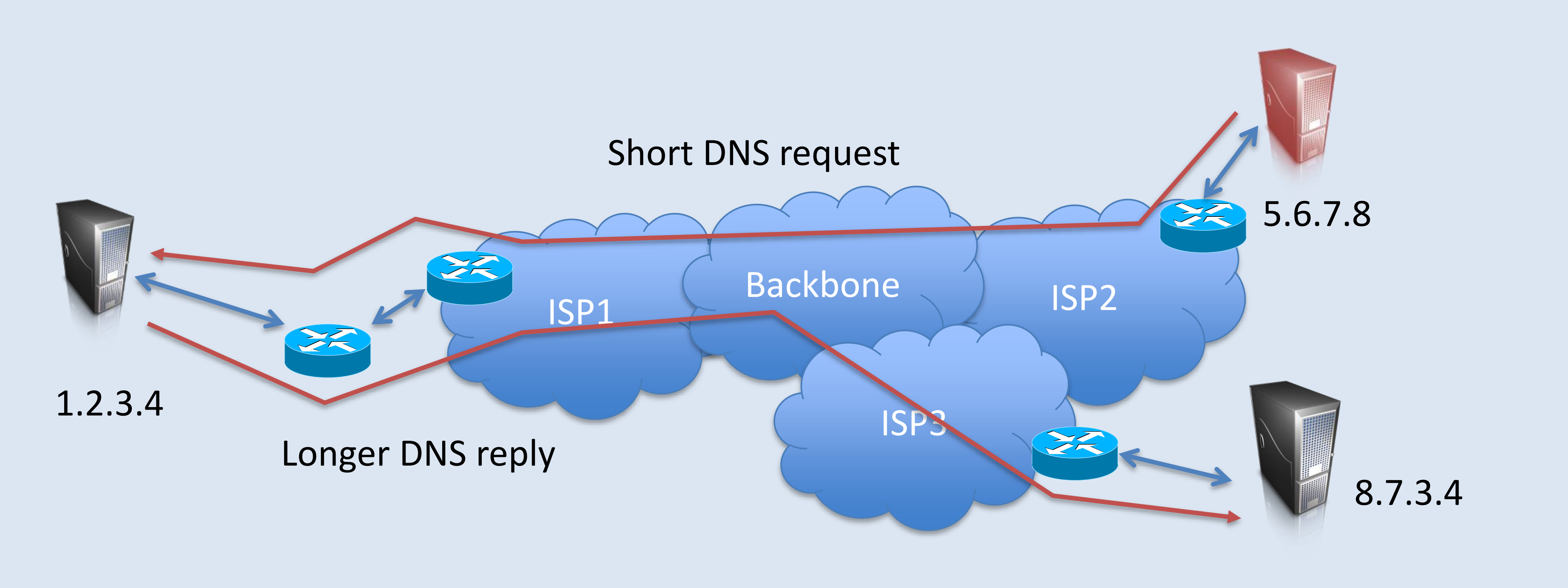
Denial of Service (DoS) attacks

DoS works better when there is ***asymmetry*** between victim and attacker

- Attacker uses few resources to cause
- Victim to consume lots of resources

Possible approach: Reflection attacks abusing a service where size of incoming packet << size of outgoing packet

Denial of Service (DoS) attacks



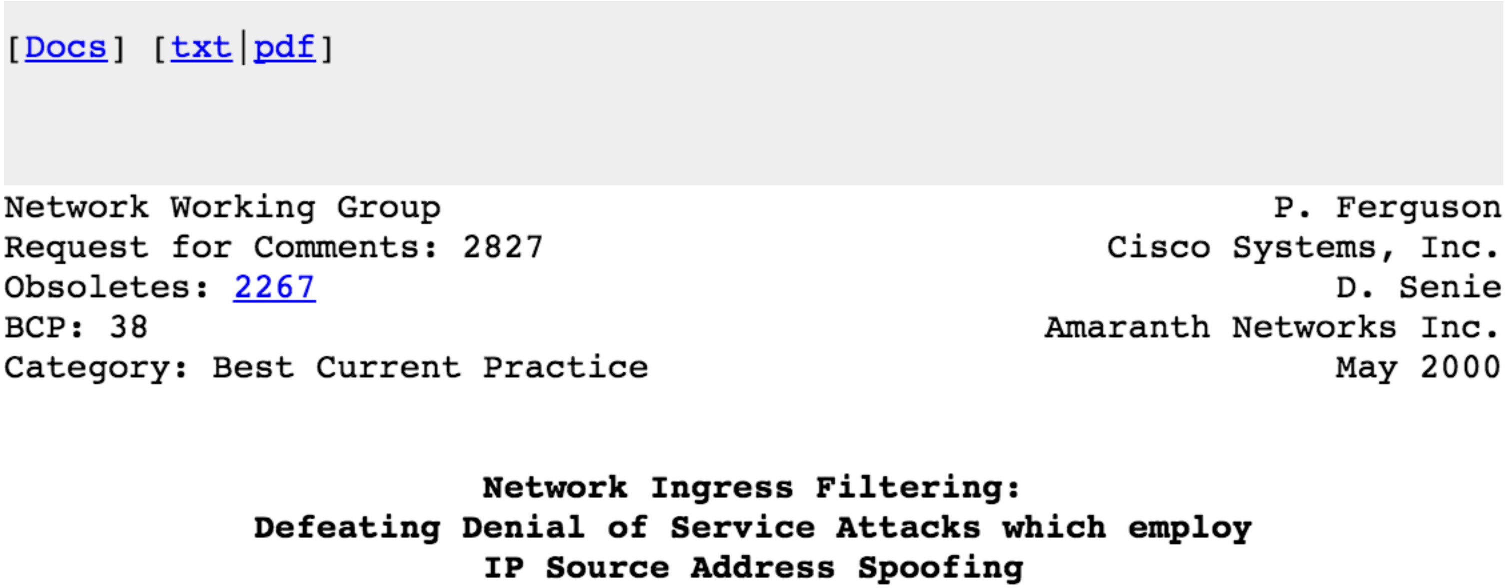
Example: DNS reflection attacks

Send DNS request w/ spoofed target IP (~65 byte request)

DNS replies sent to target (~512 byte response)

Dealing with spoofing: BCP 38

- Spoofed IPs means we cannot know where packets came from
- BCP 38 (RFC 2827): upstream ingress filtering to drop spoofed packets



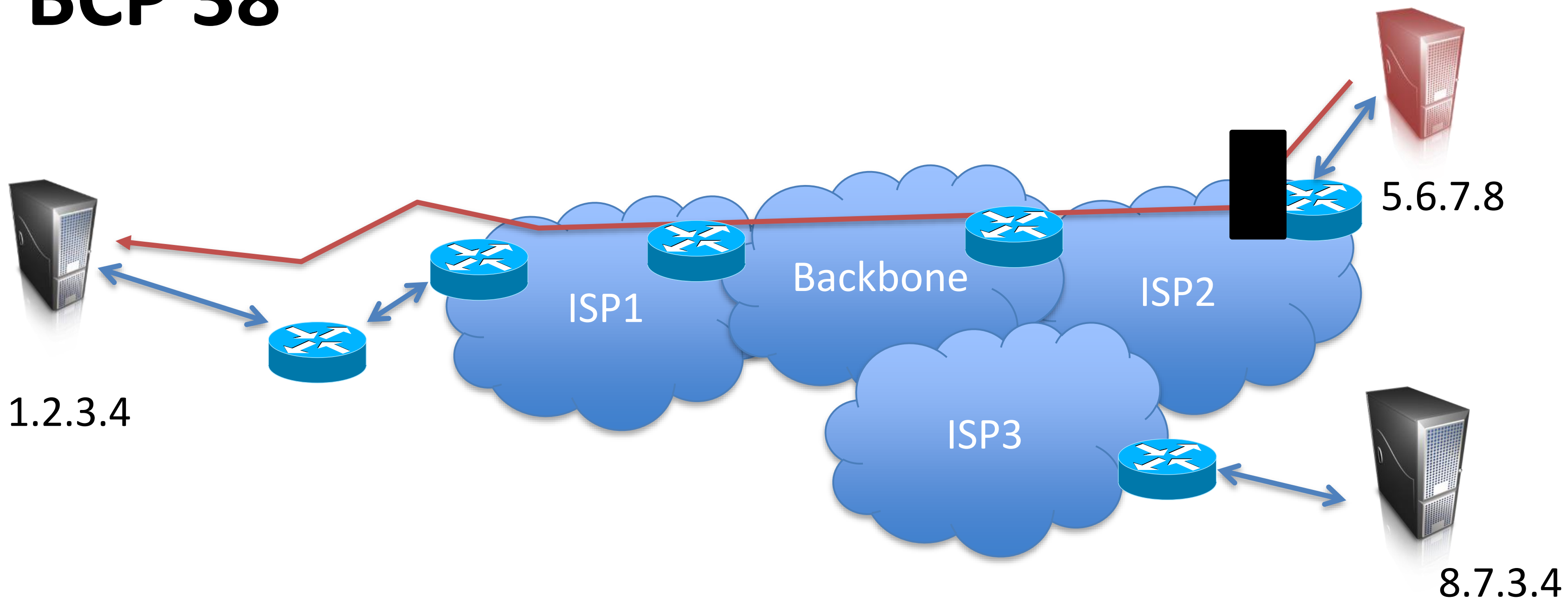
[Docs] [txt|pdf]

Network Working Group
Request for Comments: 2827
Obsoletes: 2267
BCP: 38
Category: Best Current Practice

P. Ferguson
Cisco Systems, Inc.
D. Senie
Amaranth Networks Inc.
May 2000

Network Ingress Filtering:
Defeating Denial of Service Attacks which employ
IP Source Address Spoofing

BCP 38



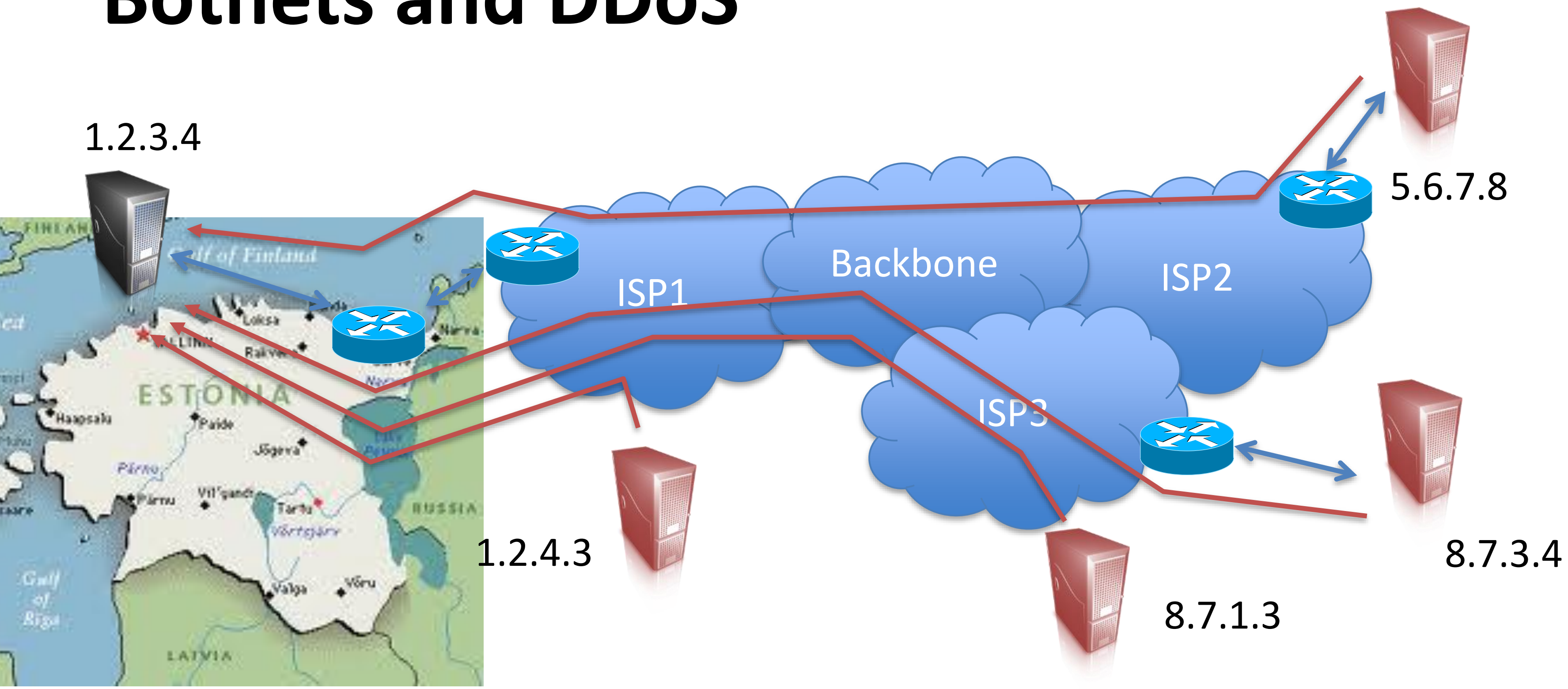
Before forwarding on packets, **check at ingress that source IP legitimate**

- Easier to do on ISP’s side
- 80% of networks adopt ingress filtering in some form

Does this stop DoS attacks?

- Requires widespread adoption and compliance
  - Often small incentives for network operator
- More and more DoS-attacks do not use spoofing
  - Botnets and distribute DoS (DDoS) attacks

Botnets and DDoS



April 27, 2007

Continued for weeks, with varying levels of intensity

Government, banking, news, university websites

Government shut down international Internet connections

Other IPv4 issues

Protocol implementation vulnerabilities

- Certain application environments demand complex ways of handling IP packets
- Fertile ground for mistakes
- Can lead to vulnerabilities

Prototypical example: **Packet fragmenting**

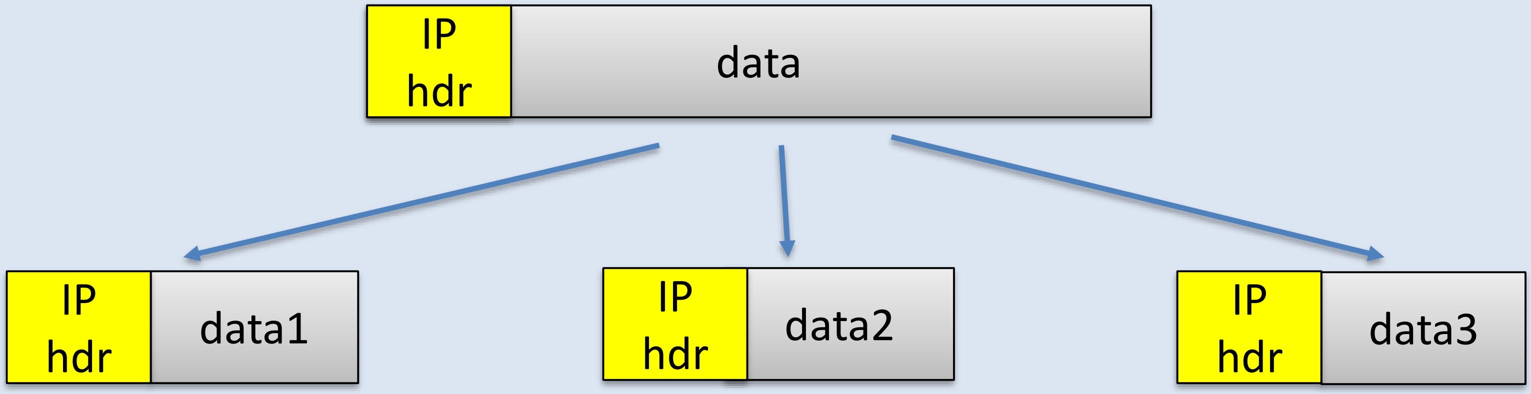
IPv4 fragmenting

IP allows datagrams of size from: 20 bytes - 65535 bytes

Problem: Some link layers only allow smaller MTU (maximum transmission unit)

- Ethernet: 1500 bytes (up to 9198 bytes with jumbo frames)
- WLAN: 2304 bytes

Solution: IP figures out MTU of next link, and fragments packet if necessary into smaller chunk (or refuses to relay!)



Path MTU discovery: Technique to discover least MTU between two IPs to avoid fragmenting.

IPv4 fragmenting

Fragmentation is controlled in the IP header

<table>
<tr><td>4-bit
version</td><td>4-bit
hdr len</td><td>8-bit
type of service</td><td colspan="2">16-bit
total length (in bytes)</td></tr>
<tr><td colspan="3">16-bit
identification</td><td>3-bit
flags</td><td>13-bit
fragmentation offset</td></tr>
<tr><td colspan="2">8-bit
time to live (TTL)</td><td>8-bit
protocol</td><td colspan="2">16-bit
header checksum</td></tr>
<tr><td colspan="5">32-bit
source IP address</td></tr>
<tr><td colspan="5">32-bit
destination IP address</td></tr>
<tr><td colspan="5">options (optional)</td></tr>
</table>

IPv4 fragmenting

16-bit identification	3-bit flags	13-bit fragmentation offset

Source-specified “unique” number identifying datagram

Fragment offset in 8-byte units

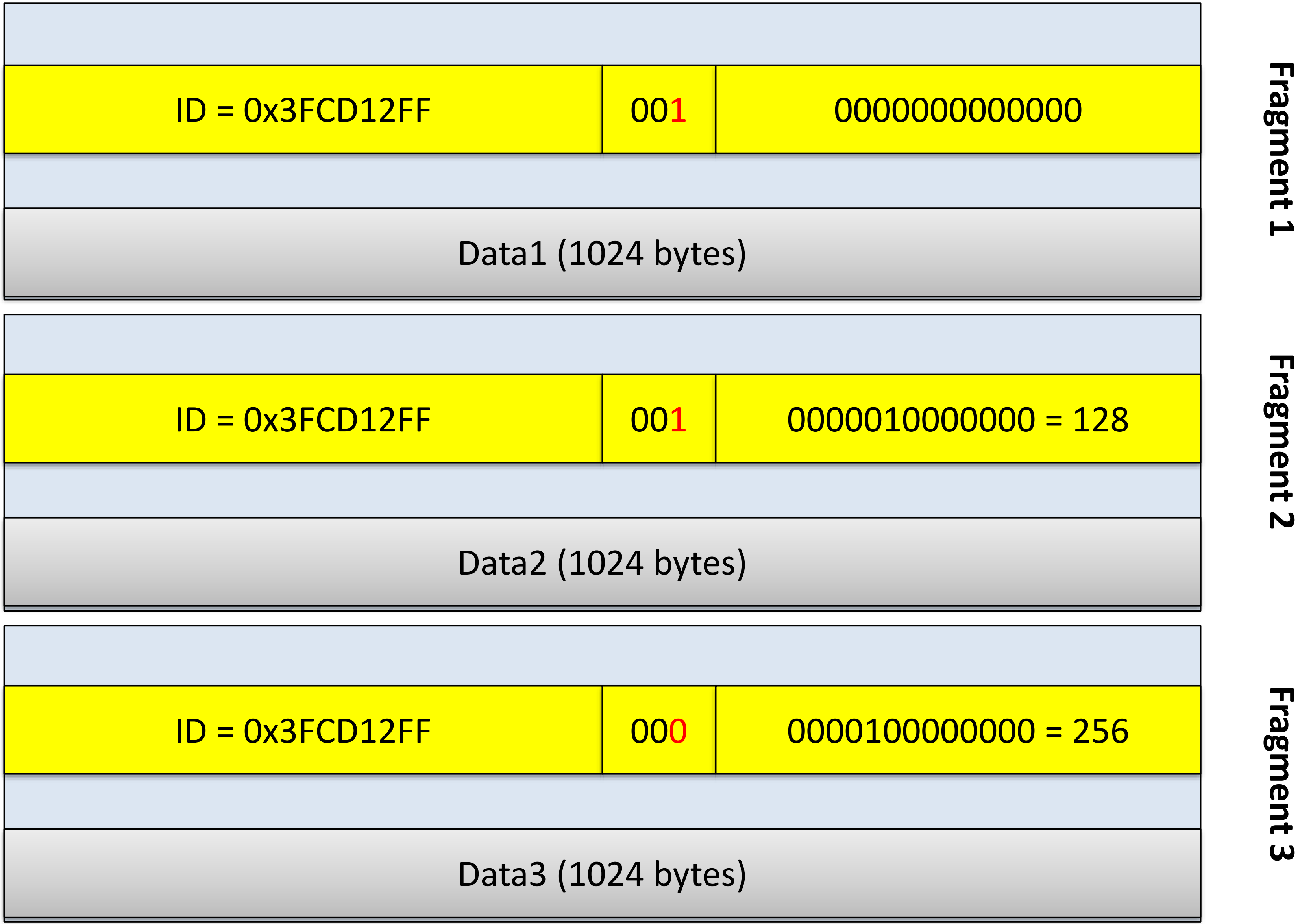
Flags:
0 b1 b2

where b1 = May Fragment (0) / Don’t Fragment (1)
where b2 = Last Fragment (0) / More Fragments (1)

Reassembly process: Receiver keeps large buffer, and re-assembles fragments into original packet size!

Possible implementation mistakes when receiving unexpected values!

IPv4 fragmenting – Example



Fragment 1

ID = 0x3FCD12FF	001	0000000000000

Data1 (1024 bytes)

Fragment 2

ID = 0x3FCD12FF	001	0000010000000 = 128

Data2 (1024 bytes)

Fragment 3

ID = 0x3FCD12FF	000	0000100000000 = 256

Data3 (1024 bytes)

Fragmentation attacks

Fragmentation assembly can be abused if done incorrectly:

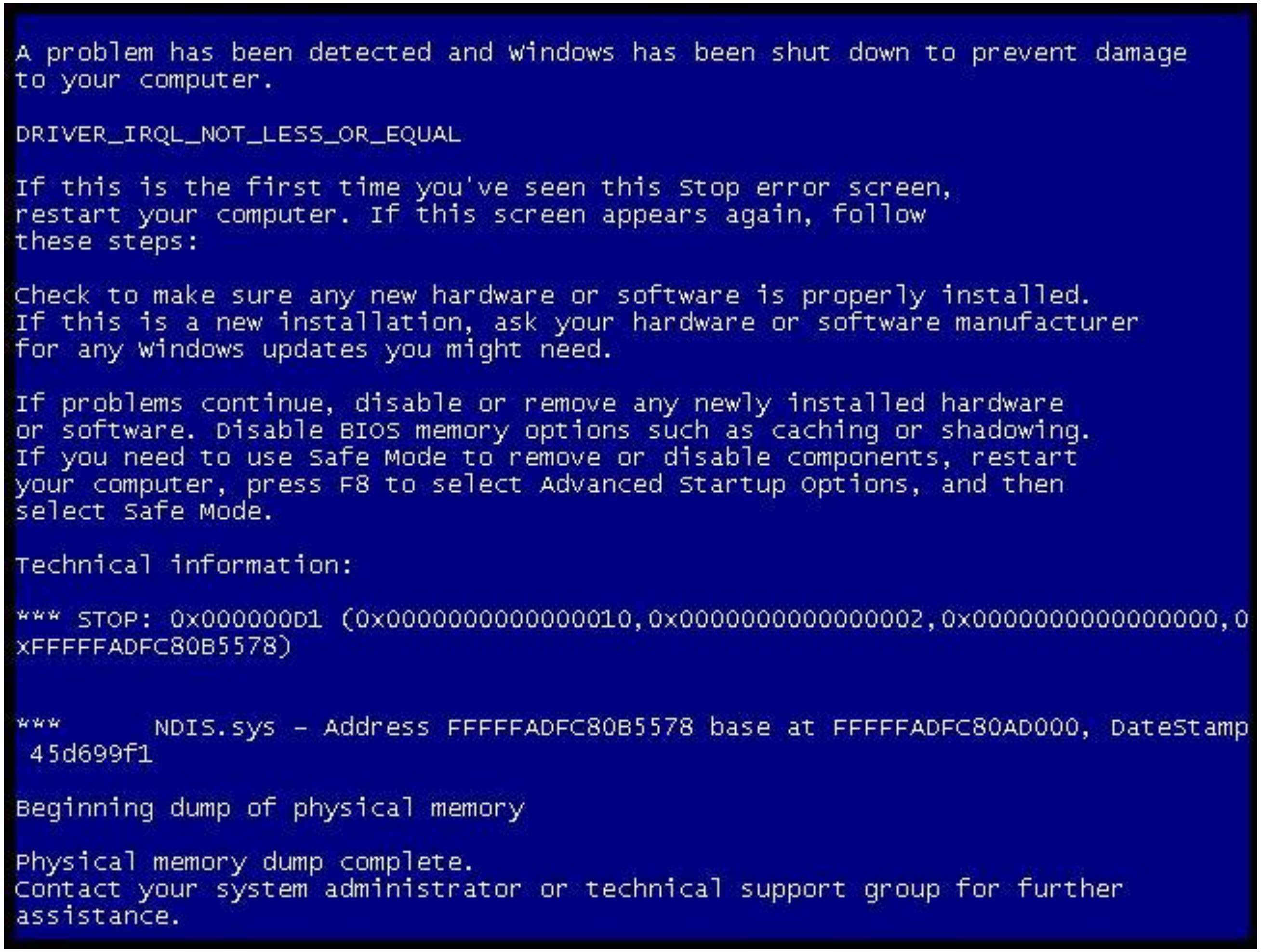
- “Ping of death”: allows sending > 65,536 byte packet, *overflows buffer*.

  This is because max offset is $65528 = (2^{13} - 1) \cdot 8$ *but IP does not prevent us from including more than 8B of data*

  Example: Last offset = 1111111111111, followed by 16 bytes of data.

- “Teardrop” DoS: mangled fragmentation crashes re-assembly code
  - Set offsets so that two packets have overlapping data!
  - Modify above example so that Data1 is 2048 bytes, leave rest unchanged!

Typical Ping-of-death outcome (1990s)



```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

DRIVER_IRQL_NOT_LESS_OR_EQUAL

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x000000D1 (0x0000000000000010,0x0000000000000002,0x0000000000000000,0
xFFFFFADFC80B5578)


***      NDIS.sys - Address FFFFFADFC80B5578 base at FFFFFADFC80AD000, DateStamp
 45d699f1

Beginning dump of physical memory

Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.
```

Agenda

1. Link Layer Issues
2. Network Layer Issues
3. **Transport Layer Issues**
4. Application Layer Issues

TCP (transport control protocol)

- Connection-oriented
  - state initialized during handshake and maintained
- Reliability is a goal
  - generates segments
  - timeout segments that aren’t ack’d
  - checksums headers,
  - reorders received segments if necessary
  - flow control

TCP Protocol

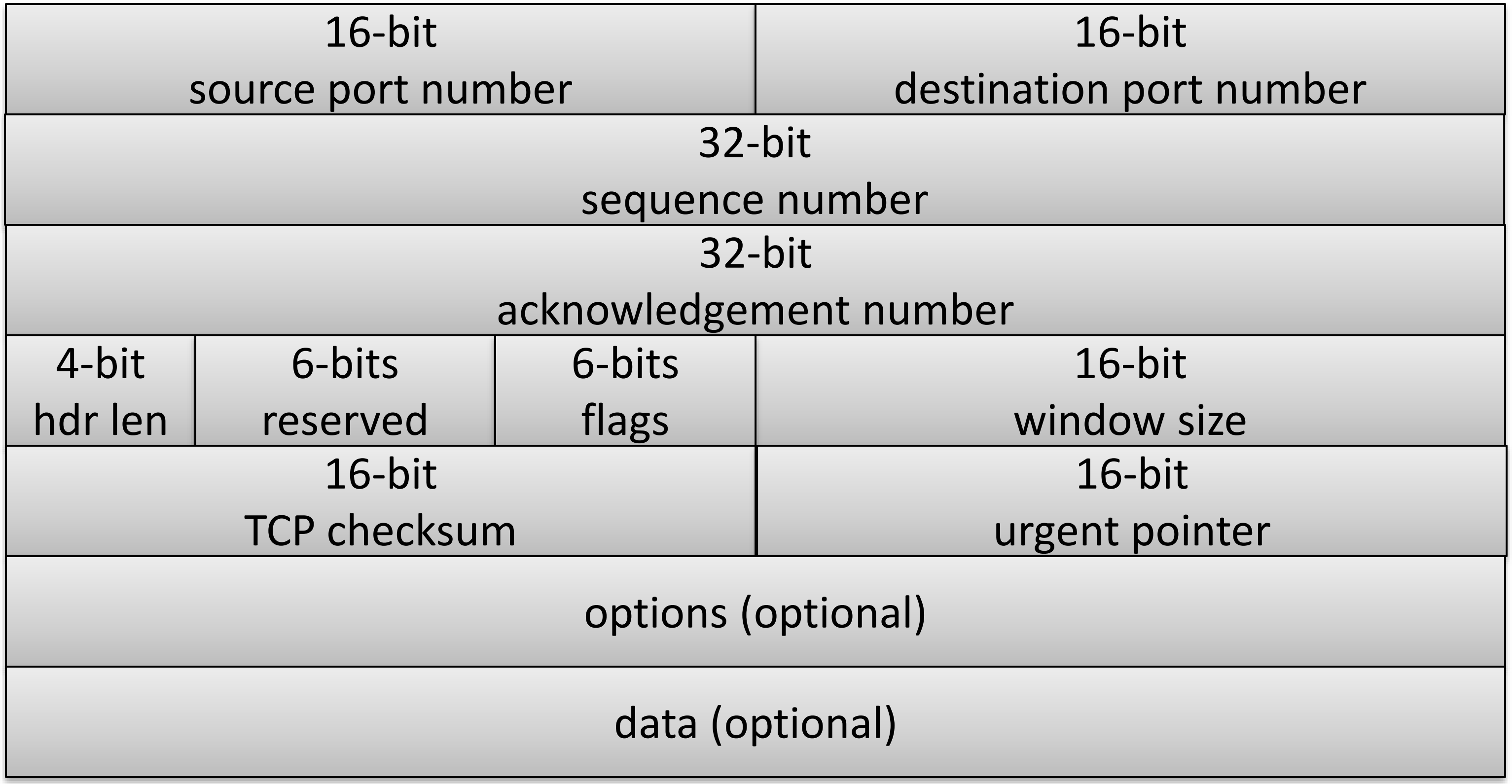
- Establishes a connection between *IP1:port1* and *IP2:port2*
- End-point is established through an Internet Socket
- Can be in one of many states:
  - **LISTEN / ESTABLISHED / CLOSED + many more**

TCP (transport control protocol)

IP hdr

TCP hdr

data



16-bit source port number

16-bit destination port number

32-bit sequence number

32-bit acknowledgement number

4-bit hdr len

6-bits reserved

6-bits flags

16-bit window size

16-bit TCP checksum

16-bit urgent pointer

options (optional)

data (optional)

TCP (transport control protocol)

TCP flags:

URG	urgent pointer valid
ACK	acknowledgement number valid
PSH	pass data to app ASAP
RST	reset connection
SYN	synchronize sequence #’s
FIN	finished sending data

TCP Connections

- Every connection is labeled by ClientIP:ClientPort and ServerIP:ServerPort
- When new connection created by client (new socket), typically client chooses random ClientPort
- Server must be listening on ServerPort, creating a passive socket
  - New connections handled by separate thread

TCP Connection Logic

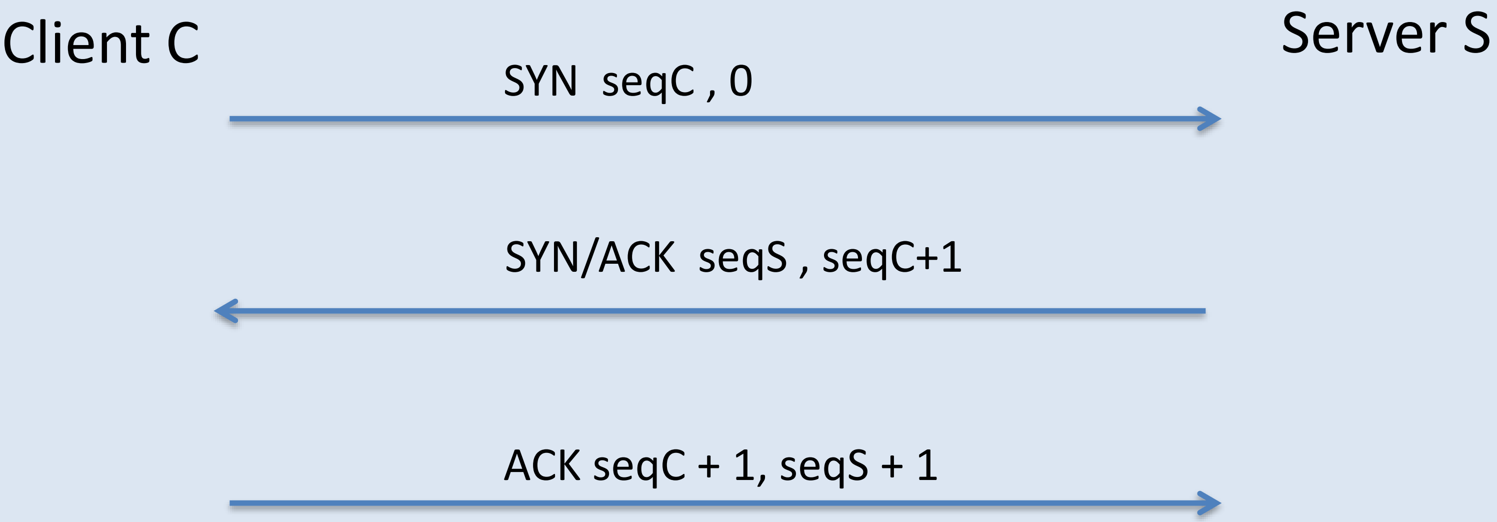
- Packets sent from client / server are assigned increasing **sequence numbers** seqC and seqS, initialized when establishing connection
  - Sequence number are per byte
- Also each packet contains the **acknowledgment number** to acknowledge received bytes
- TCP protocol handles missing messages / re-sent / etc

Abstractly, socket simply looks like a file with read/write interface once connection is open

TCP handshake

Protocol establishes a TCP session between Client C and Server S

Connection will be labeled by ClientIP:ClientPort and ServerIP:ServerPort



Client C

Server S

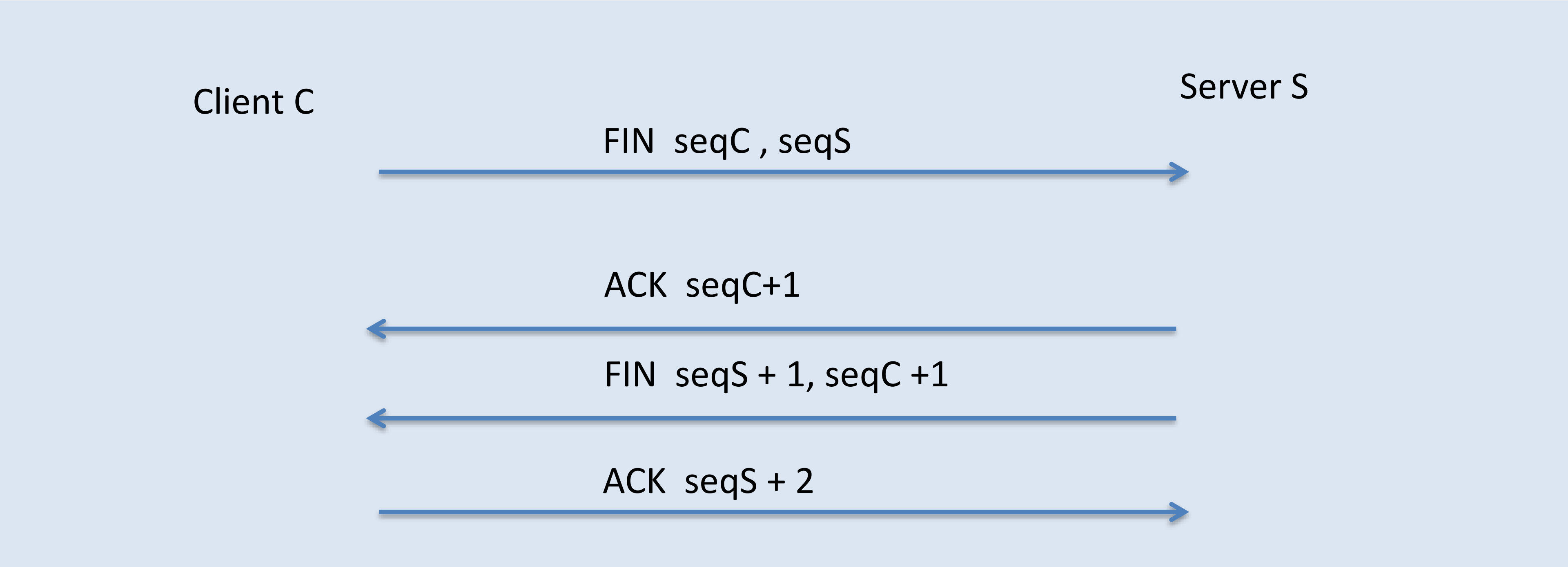
SYN seqC , 0

SYN/ACK seqS , seqC+1

ACK seqC + 1, seqS + 1

SYN = syn flag set ACK = ack flag set x,y = x is sequence #, y is acknowledge #

TCP teardown



Client C

Server S

FIN seqC , seqS

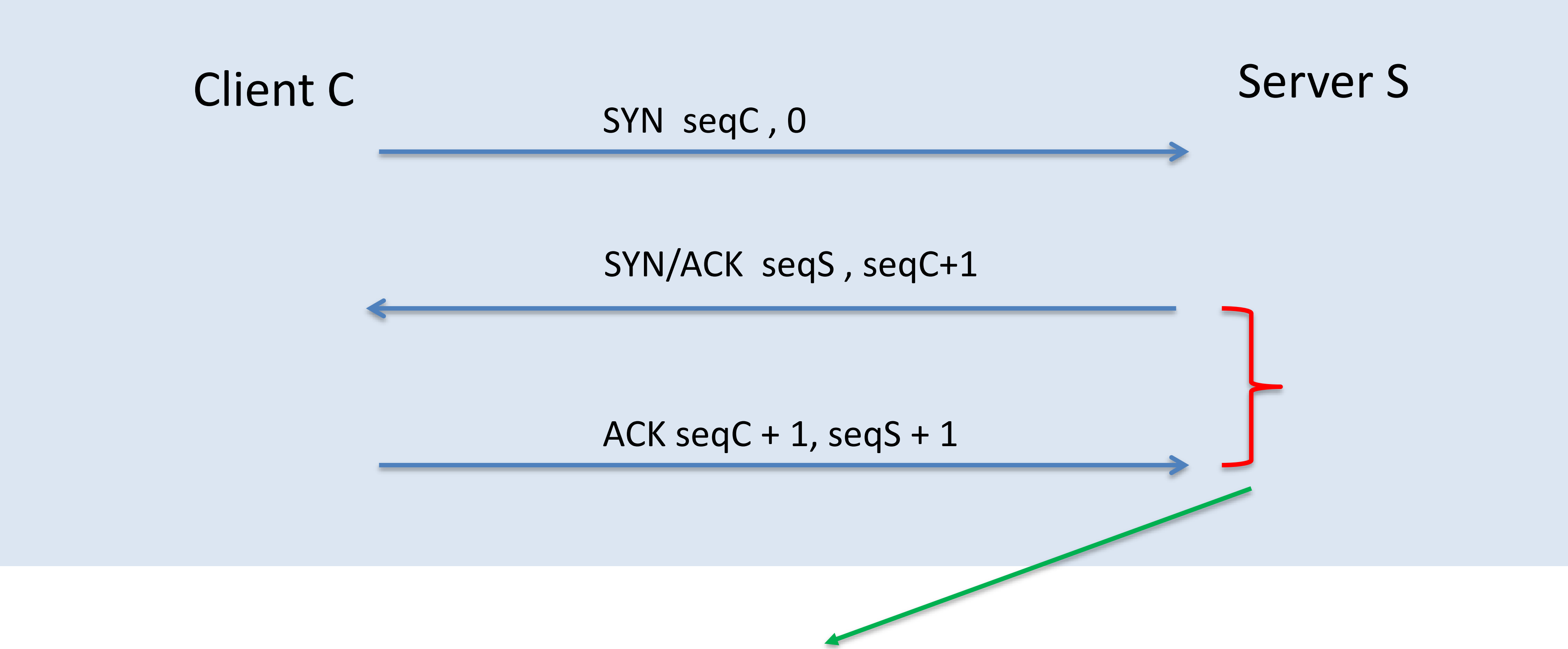
ACK seqC+1

FIN seqS + 1, seqC +1

ACK seqS + 2

SYN = syn flag set
ACK = ack flag set
x,y = x is sequence #, y is acknowledge #

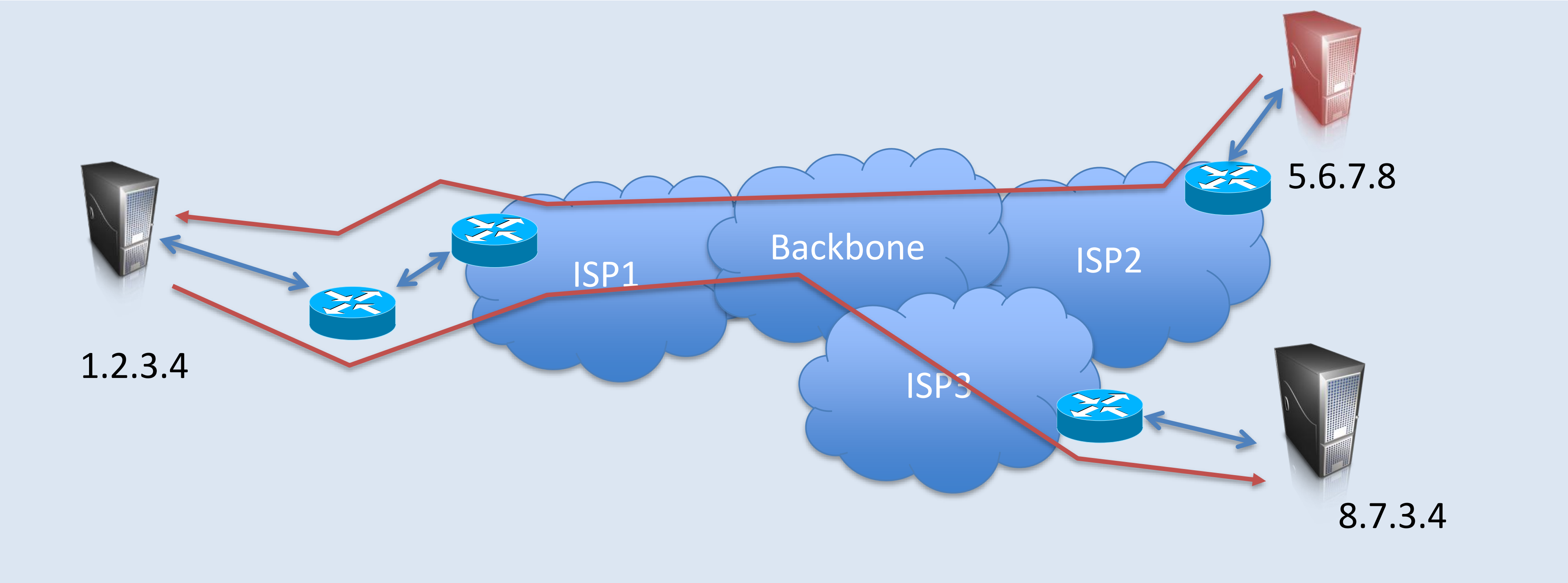
TCP handshake



Server needs to remember that a SYN/ACK message was sent back to client! This costs some memory

Q: How can this be abused?

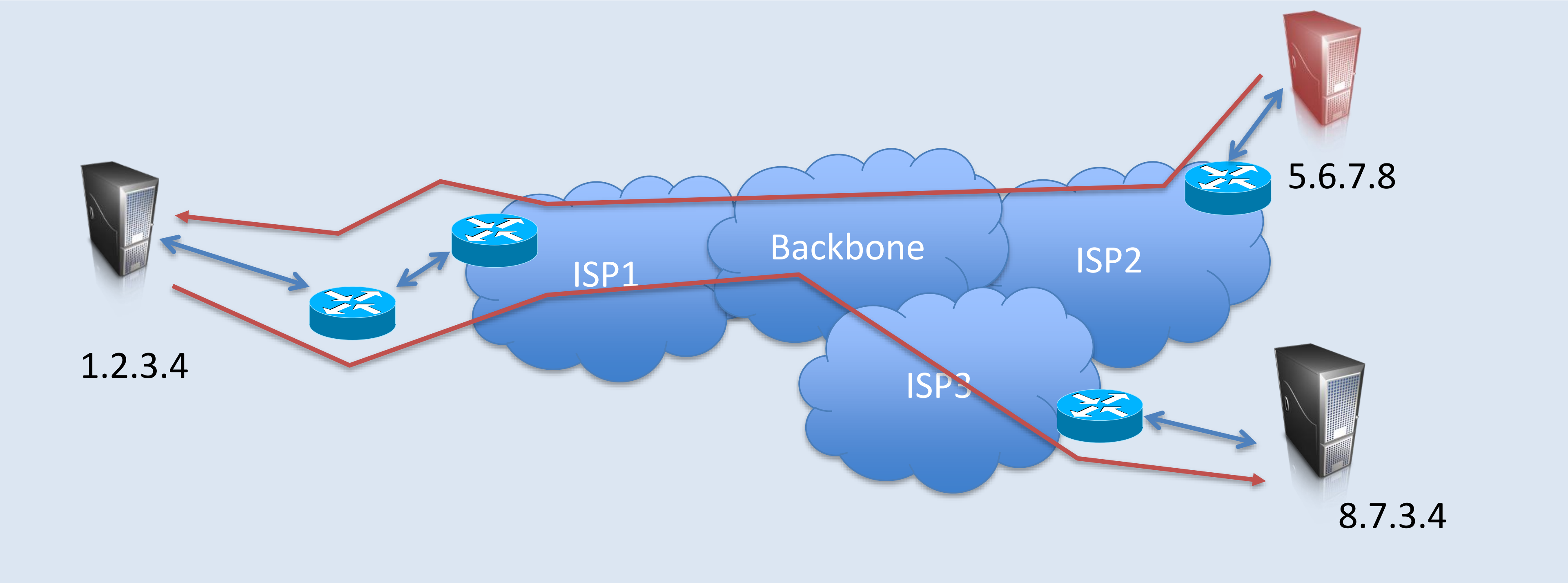
TCP SYN floods



Send lots of TCP SYN packets to 1.2.3.4

- 1.2.3.4 maintains state for each SYN packet for some amount window of time
- Side question: If 5.6.7.8 sets SRC IP to be 8.7.3.4, what does 8.7.3.4 receive?

TCP SYN floods

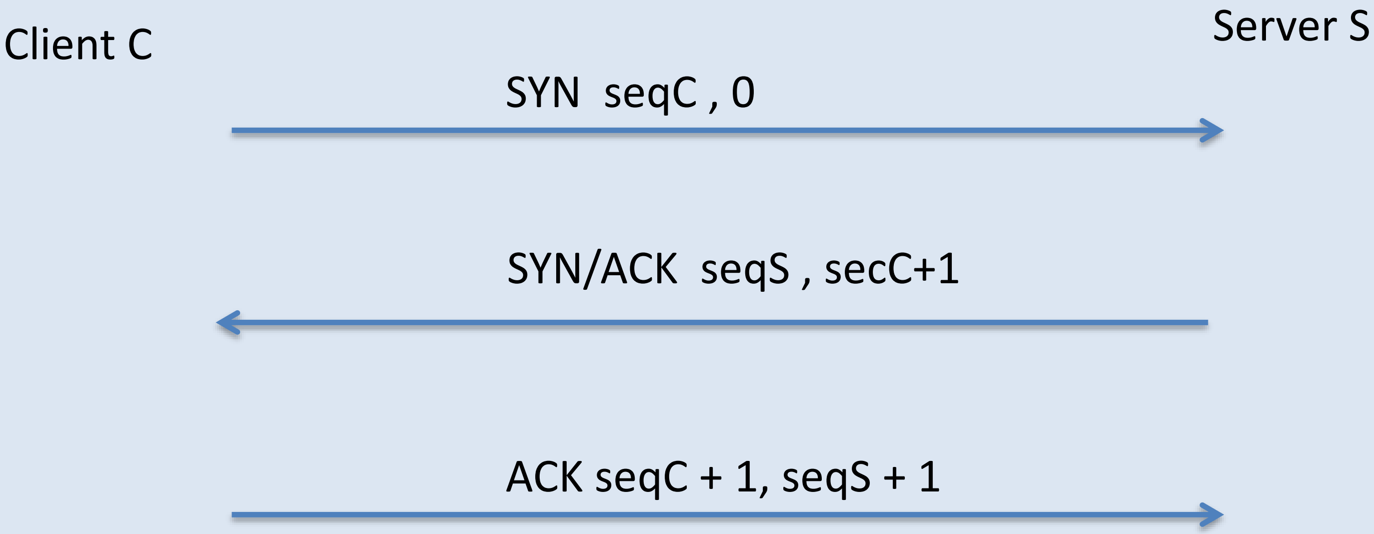


Send lots of TCP SYN packets to 1.2.3.4

- Why is this a denial of service attack?

Answer: 1.2.3.4 runs out of memory (if not cleverly implemented!)

TCP handshake


Client C

Server S

SYN seqC , 0

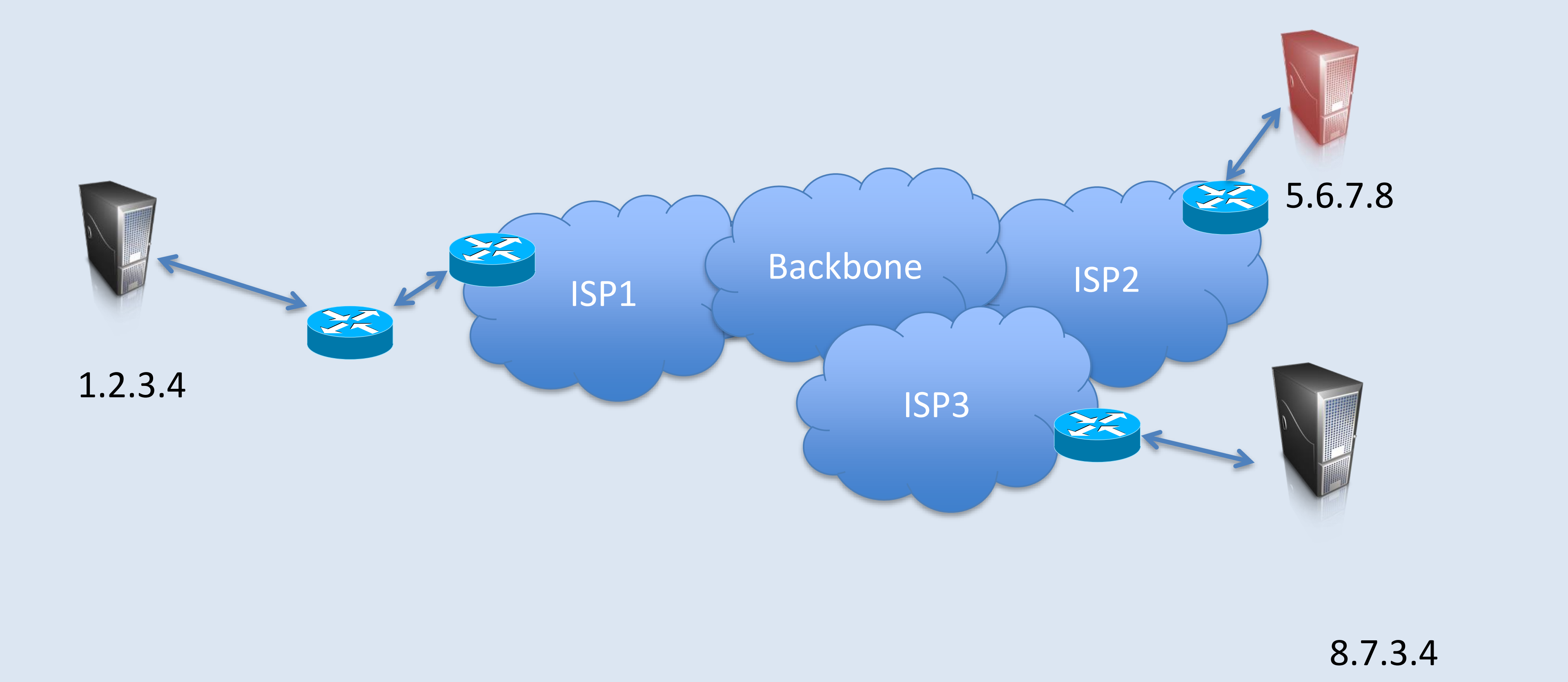
SYN/ACK seqS , secC+1

ACK seqC + 1, seqS + 1

How are secC and seqS selected?

Sequence numbers are the main mechanism for reliability allowing us to know how packets are to be ordered!

Predictable sequence numbers

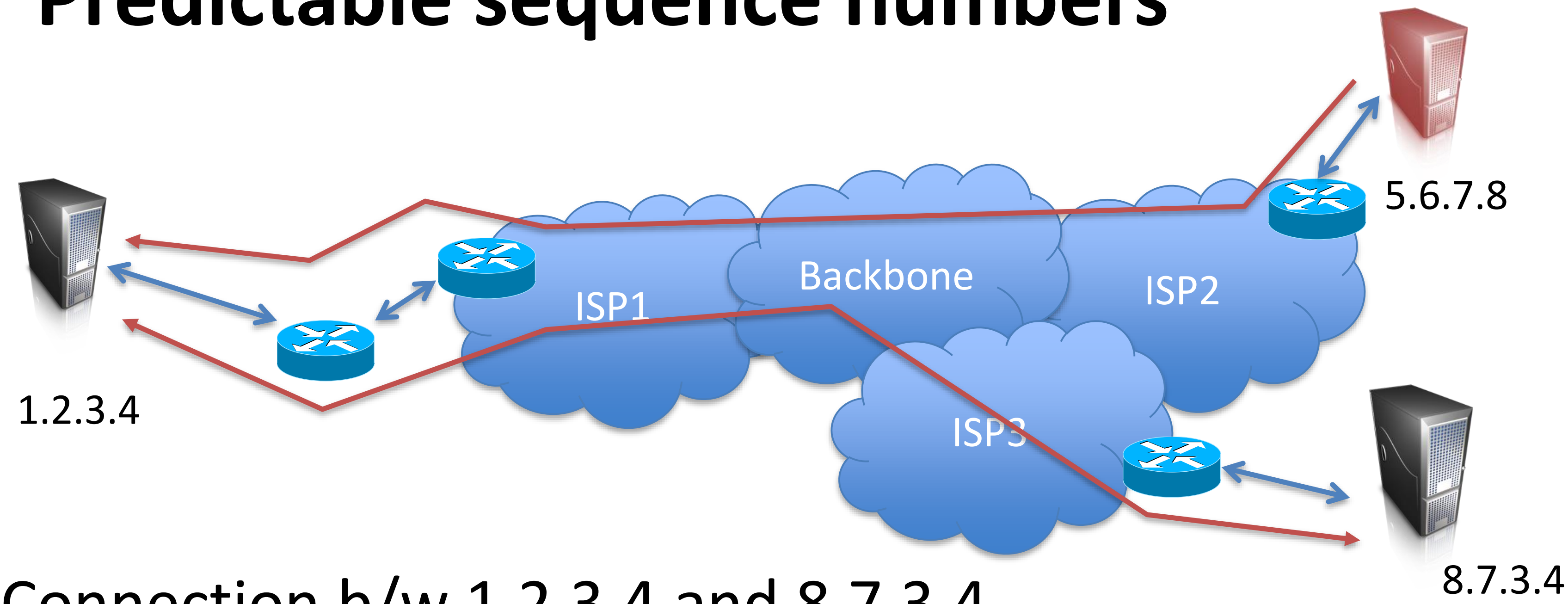


4.4BSD used predictable **initial sequence numbers** (ISNs)

- At system initialization, set ISN to 1
- Increment ISN by 64,000 every half-second

What can a clever attacker do? [Assume spoofing is possible]

Predictable sequence numbers



Connection b/w 1.2.3.4 and 8.7.3.4

Forge a FIN packet from 8.7.3.4 to 1.2.3.4

src: 8.7.3.4
dst: 1.2.3.4

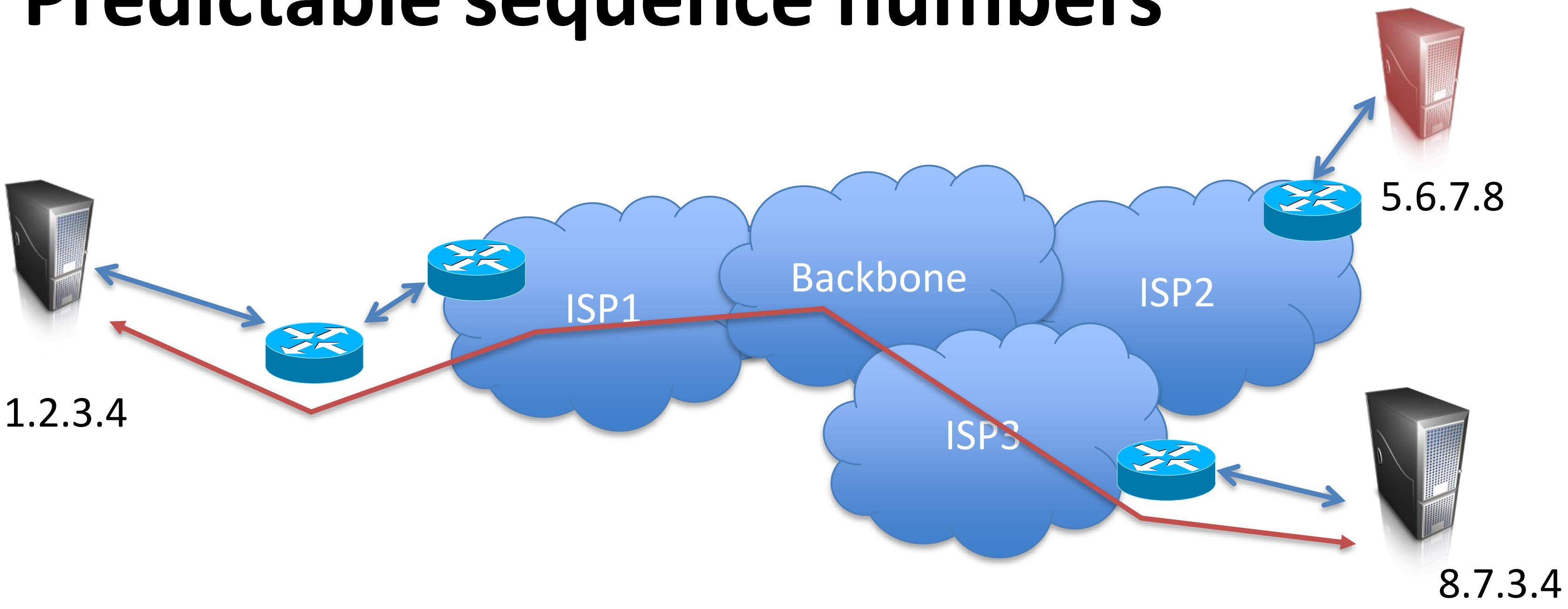
seq#(8.7.3.4)
FIN

Forge some application-layer packet from 8.7.3.4 to 1.2.3.4

src: 8.7.3.4
dst: 1.2.3.4

seq#(8.7.3.4)
“rsh rm –rf /”

Predictable sequence numbers



Fix idea 1:

- Random ISN at system startup
- Increment by 64,000 each half second

Better fix:

- Random ISN for every connection

Also:

- Cryptography at higher level should prevent injection

Agenda

1. Link Layer Issues
2. Network Layer Issues
3. Transport Layer Issues
4. **Application Layer Issues**

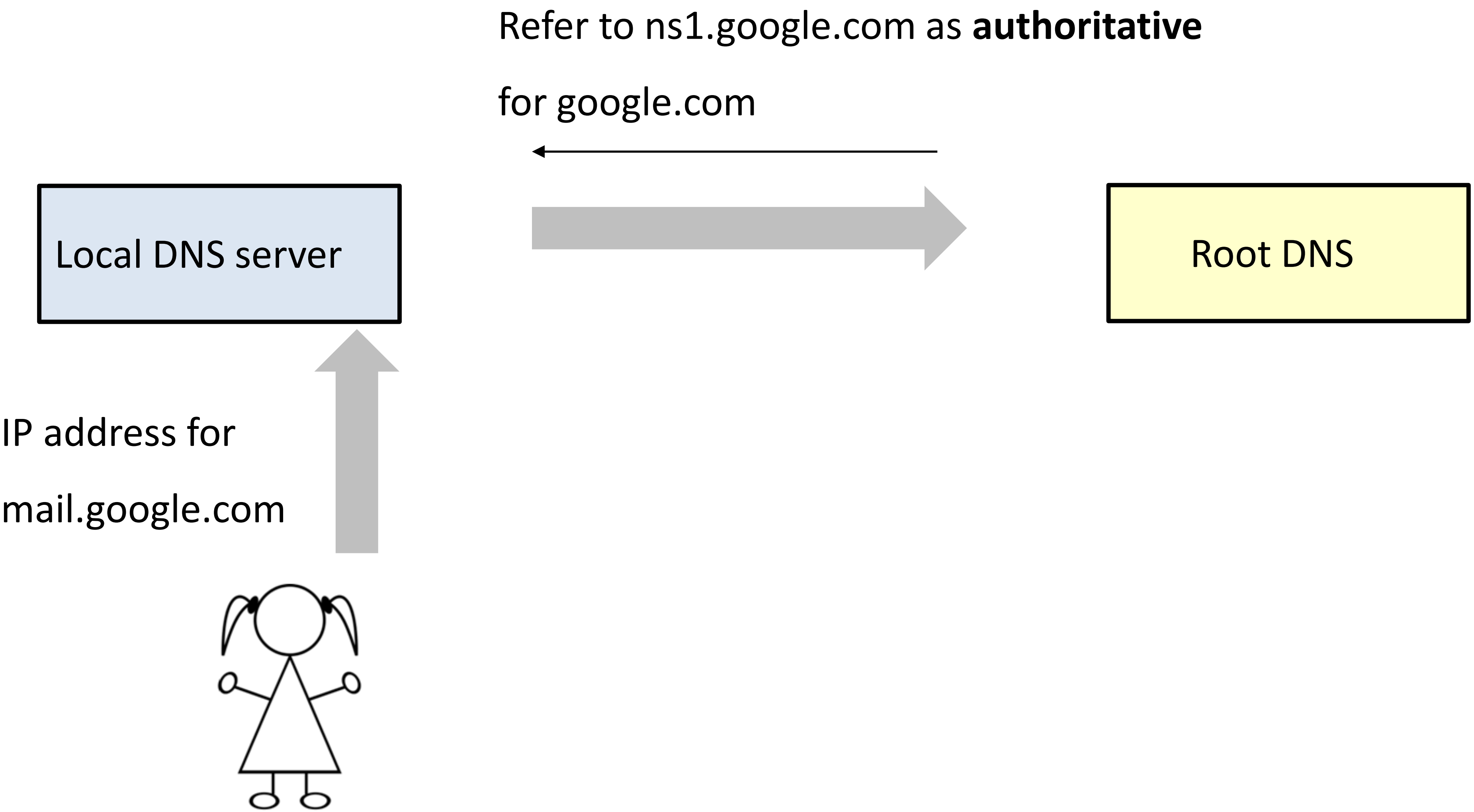
DNS: Hosts $\longrightarrow$ IP

We don’t want to have to remember IP addresses

Early days of ARPANET: manually managed hosts.txt served from single computer at SRI

Today’s solution: DNS system (Domain Name Service)

DNS Recap



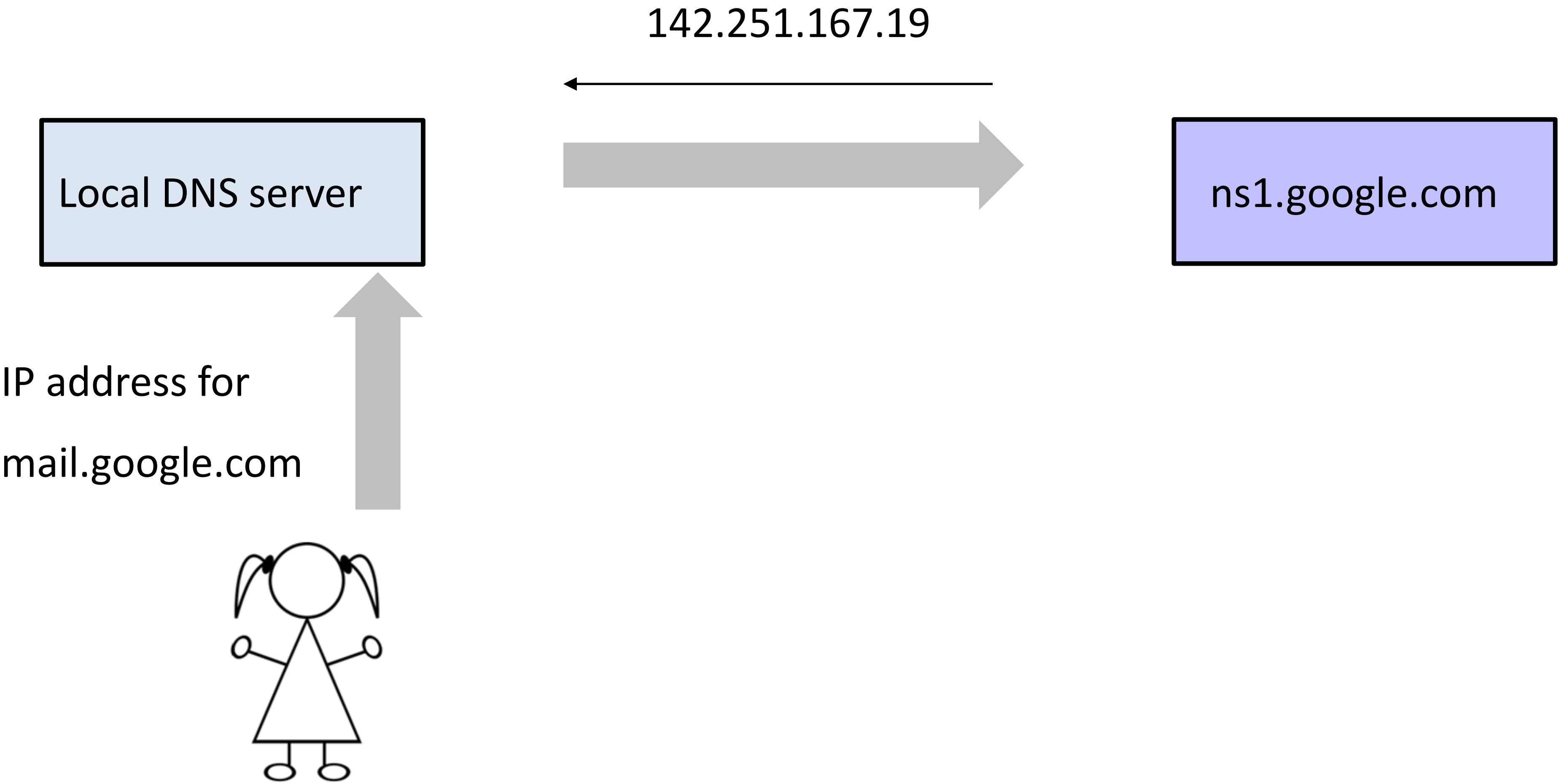
Refer to ns1.google.com as **authoritative** for google.com

Local DNS server

Root DNS

IP address for mail.google.com

DNS Recap



142.251.167.19

Local DNS server

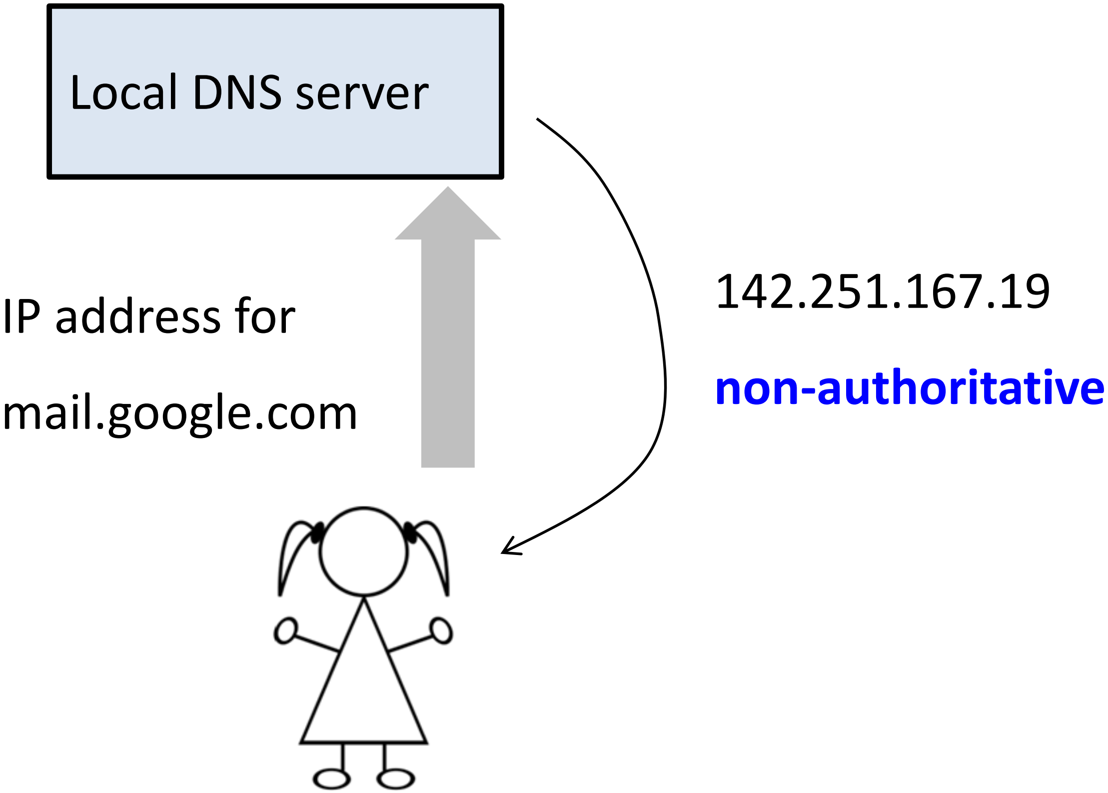
ns1.google.com

IP address for

mail.google.com

DNS Recap

Cache info for future queries



Local DNS server

IP address for

mail.google.com

142.251.167.19

non-authoritative

Root name servers – attacks

SOFTWARE // ENTERPRISE APPLICATIONS

2/7/2007
05:52 PM

Sharon Gaudin
News

Secrets Of The DoS Root Server Attack Revealed

Security experts say possibly millions of zombie computers were used in Tuesday's attack on the Internet's 13 root servers. But the attack didn't work because people had been planning for it for years.

Do you know what your computer was doing the other night?

That's the question a lot of security professionals and analysts would like to put to users. On Tuesday, the 13 servers that help manage worldwide Internet traffic were hit by a denial-of-service attack that nearly took down three of them. Analysts say the hackers' used possibly millions of zombie computers to wage the attack -- and they expect that army is populated with the desktops and laptops of unknowing users around the world.

"Individuals have contributed to this problem without knowing it," says Graham Cluley, a senior technology consultant with Sophos. "People heard about hackers doing these things, but guess what? It may have been your computer doing part of the hacking. ... People need to take more responsibility over the cleanliness of their PCs."

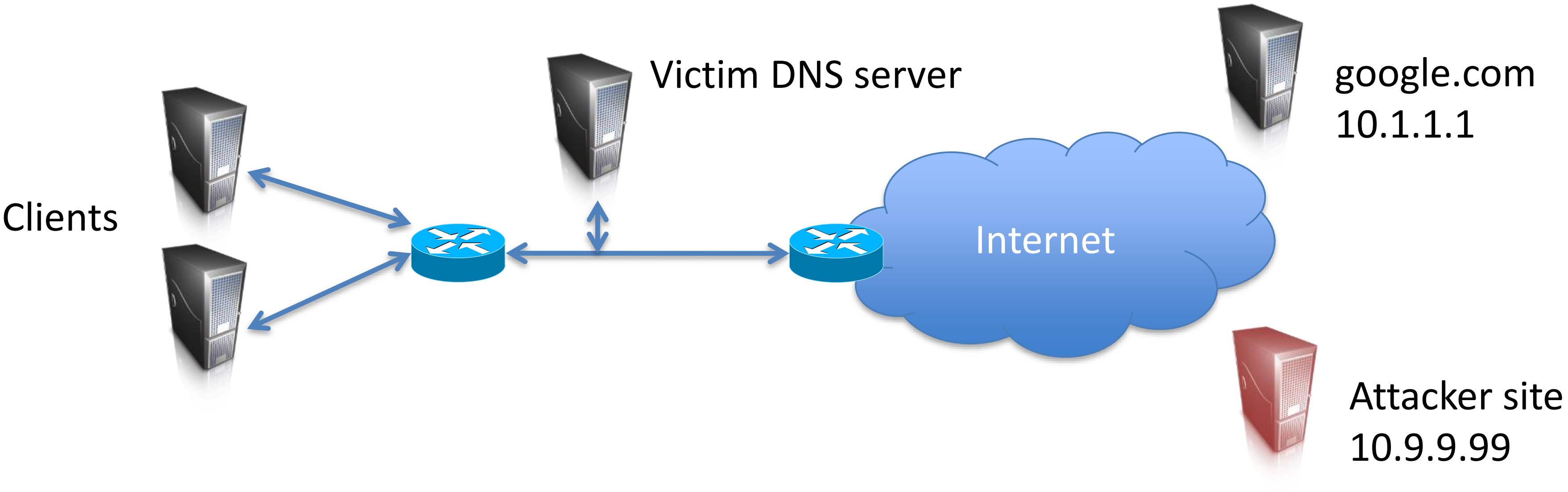
Building A Mobile

Among 688 respondents, 46
apps, with an additional 24%
Soon all apps will look like m
for those with no plans to ge

Caching

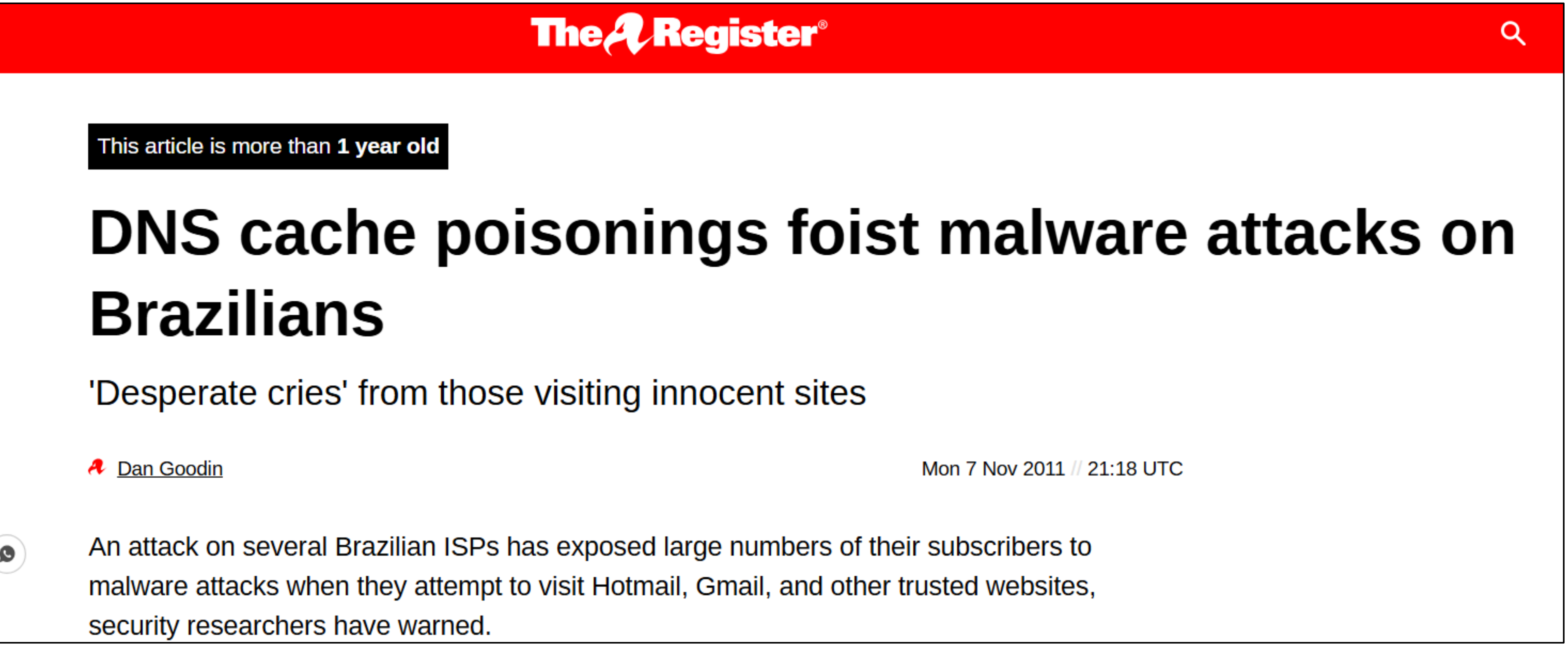
- DNS servers will cache responses
  - Both negative and positive responses
  - Speeds up queries
  - periodically times out. TTL set by data owner

DNS cache poisoning



Goal: Redirect traffic meant for google.com to 10.9.9.99 by abusing victim’s DNS server

An example of DNS poisoning attack



This article is more than **1 year old**

DNS cache poisonings foist malware attacks on Brazilians

'Desperate cries' from those visiting innocent sites

Dan Goodin

Mon 7 Nov 2011 // 21:18 UTC

An attack on several Brazilian ISPs has exposed large numbers of their subscribers to malware attacks when they attempt to visit Hotmail, Gmail, and other trusted websites, security researchers have warned.

DNS Cache Poisoning Attack

Kaminsky, 2008

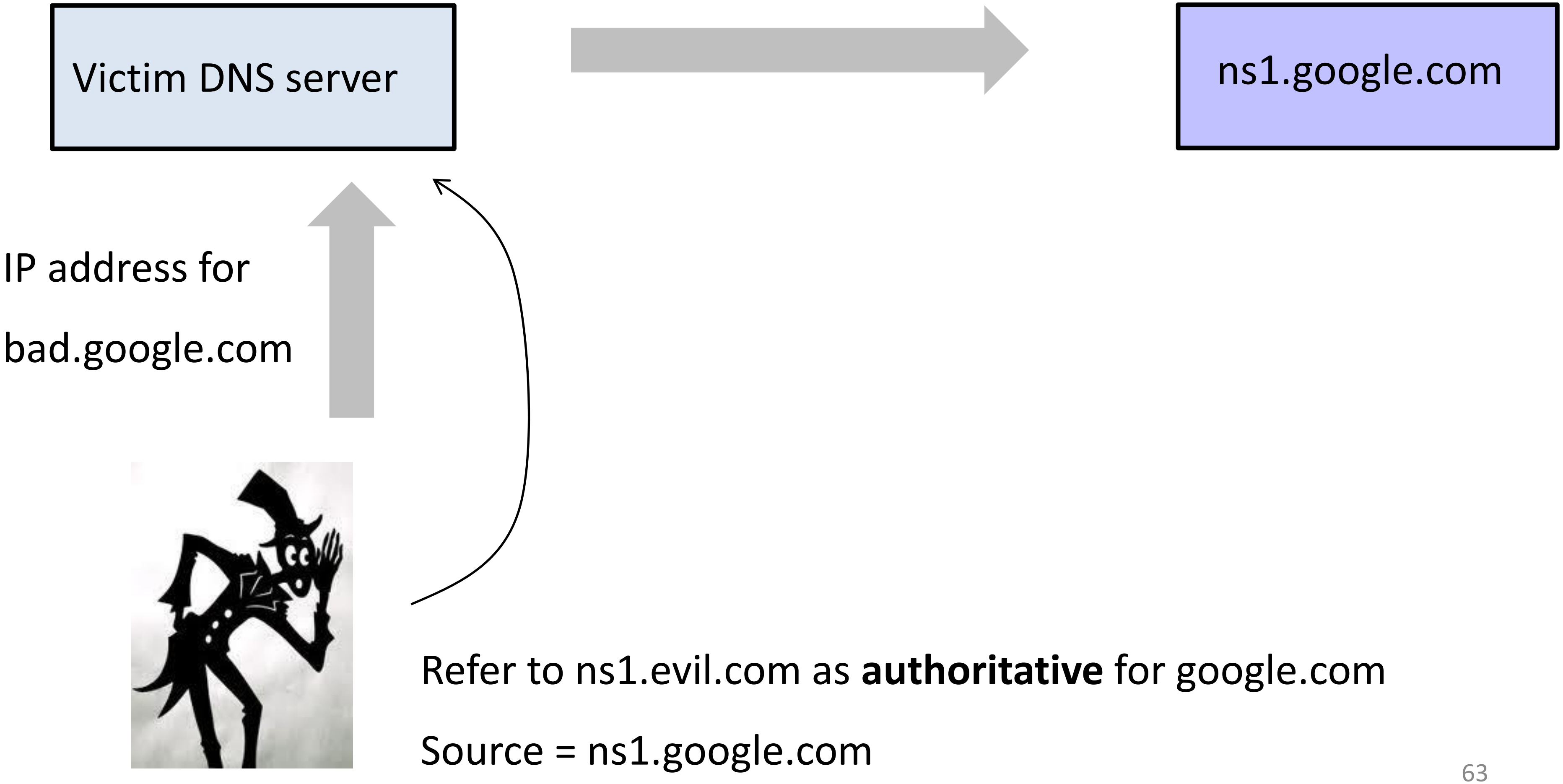
Victim DNS server

IP address for

bad.google.com

DNS Cache Poisoning Attack

Kaminsky, 2008



Victim DNS server

ns1.google.com

IP address for

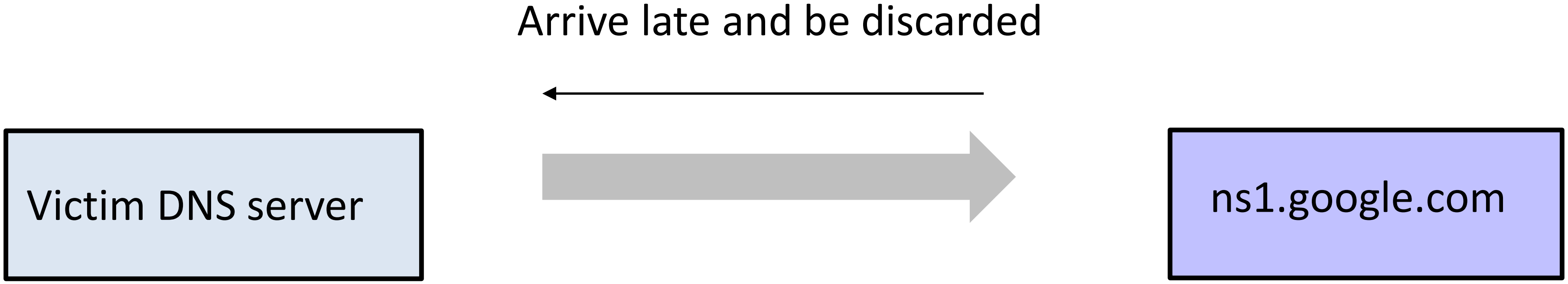
bad.google.com

Refer to ns1.evil.com as **authoritative** for google.com

Source = ns1.google.com

DNS Cache Poisoning Attack

Kaminsky, 2008



Arrive late and be discarded

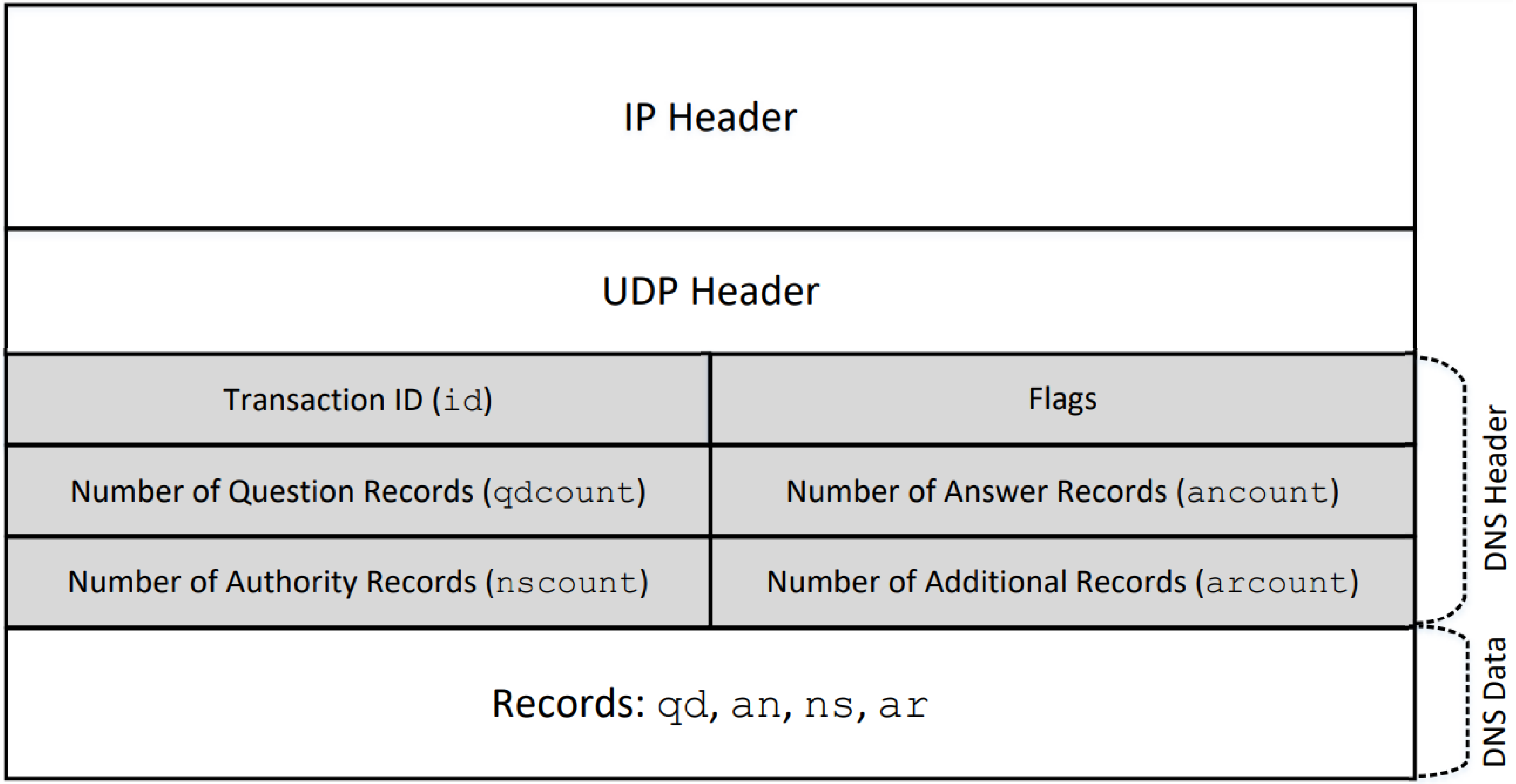
Victim DNS server

ns1.google.com

Cache: ns1.evil.com is

authoritative for google.com

Crafting Spoofed DNS Reply: Structure of DNS



IP Header

UDP Header

Transaction ID (`id`) | Flags

Number of Question Records (`qdcount`) | Number of Answer Records (`ancount`)

Number of Authority Records (`nscount`) | Number of Additional Records (`arcount`)

Records: `qd`, `an`, `ns`, `ar`

Flags: aa = 1 (authoritative answer), qr= 1 (response)

DNS Record Type

Question Record

Name	Record Type	Class
www.example.com	“A” Record 0x0001	Internet 0x0001

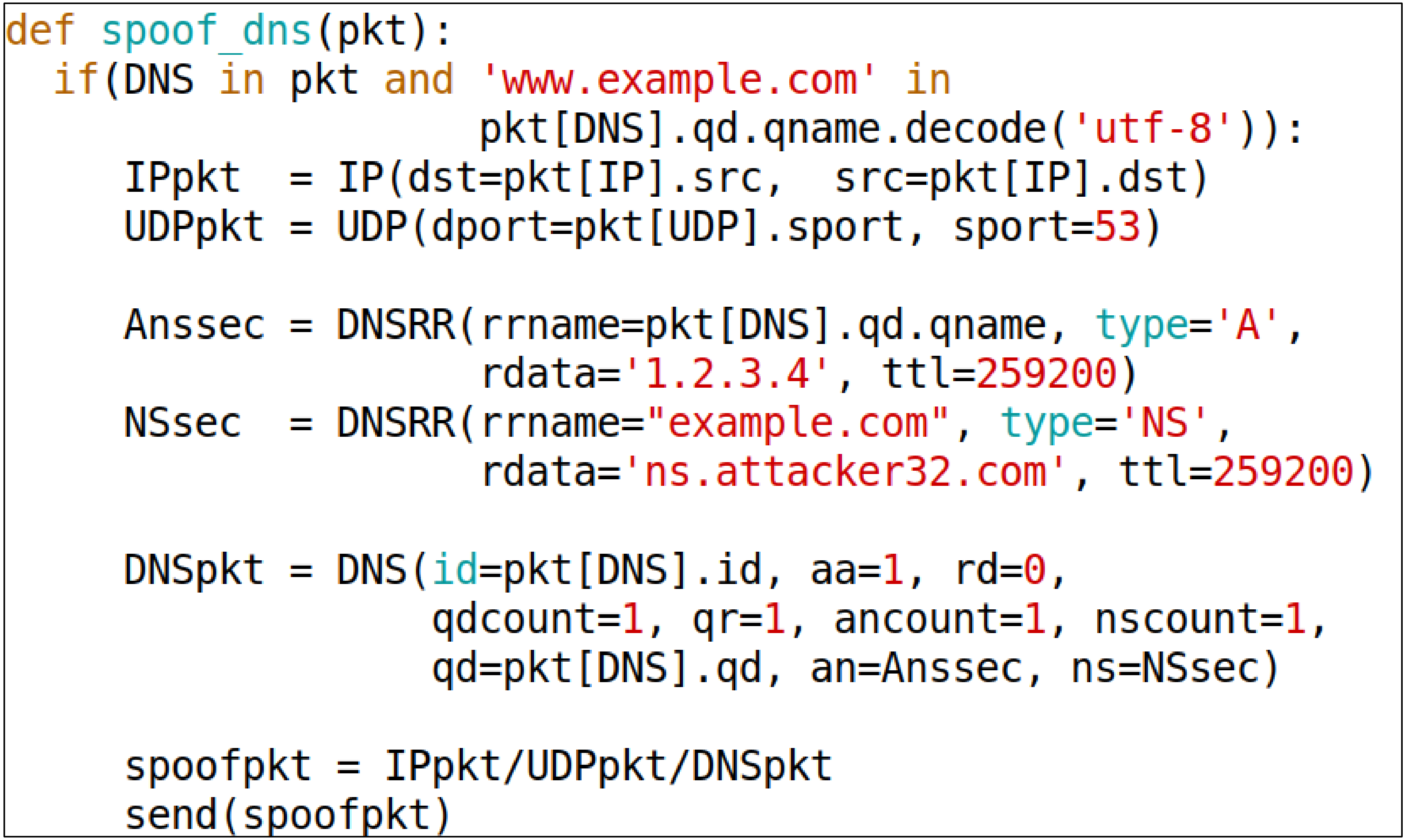
Answer Record

Name	Record Type	Class	Time to Live	Data Length	Data: IP Address
www.example.com	“A” Record 0x0001	Internet 0x0001	0x00002000 (seconds)	0x0004	1.2.3.4

Authority Record

Name	Record Type	Class	Time to Live	Data Length	Data: Name Server
example.com	“NS” Record 0x0002	Internet 0x0001	0x00002000 (seconds)	0x0013	ns.example.com

Code Example: Poisoning Local DNS



```
def spoof_dns(pkt):
  if(DNS in pkt and 'www.example.com' in
                    pkt[DNS].qd.qname.decode('utf-8')):
     IPpkt  = IP(dst=pkt[IP].src,  src=pkt[IP].dst)
     UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

     Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                    rdata='1.2.3.4', ttl=259200)
     NSsec  = DNSRR(rrname="example.com", type='NS',
                    rdata='ns.attacker32.com', ttl=259200)

     DNSpkt = DNS(id=pkt[DNS].id, aa=1, rd=0,
                  qdcount=1, qr=1, ancount=1, nscount=1,
                  qd=pkt[DNS].qd, an=Anssec, ns=NSsec)

     spoofpkt = IPpkt/UDPpkt/DNSpkt
     send(spoofpkt)
```

Flags: aa = 1 (authoritative answer), qr= 1 (response)