

## Module 4

### Middleboxes

Prepared by Dr. Manal Mizher

### Lecture 8

## جامعة الجميع الذكية

تسعى لتكون الكيان الأول الذي يتبنى مبدأ المجتمع التعليمي  
التعاوني ويقدم العلم مقابل العمل لتنمية وبناء المجتمعات.

## Network programming

Course ID: CYB615



جامعة الجميع الذكية  
EVERYONE'S SMART UNIVERSITY

1. TCP Flags, Inbound, Outbound
2. Write/Read to .pcap file in Scapy
3. Middleboxes

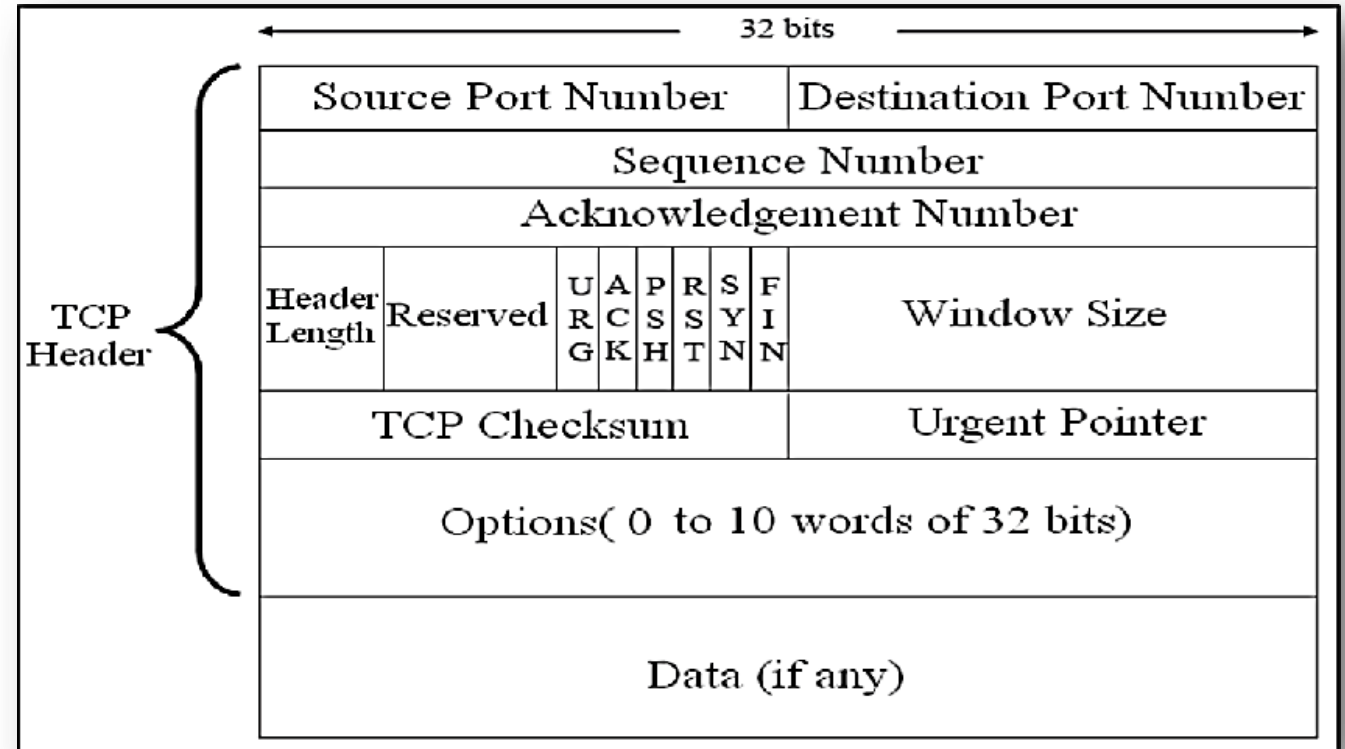
(Definition, Network Functions (NFs), Benefits, How is use it? , Criticism and challenges, Categories)

## 4. Examples of Middleboxes

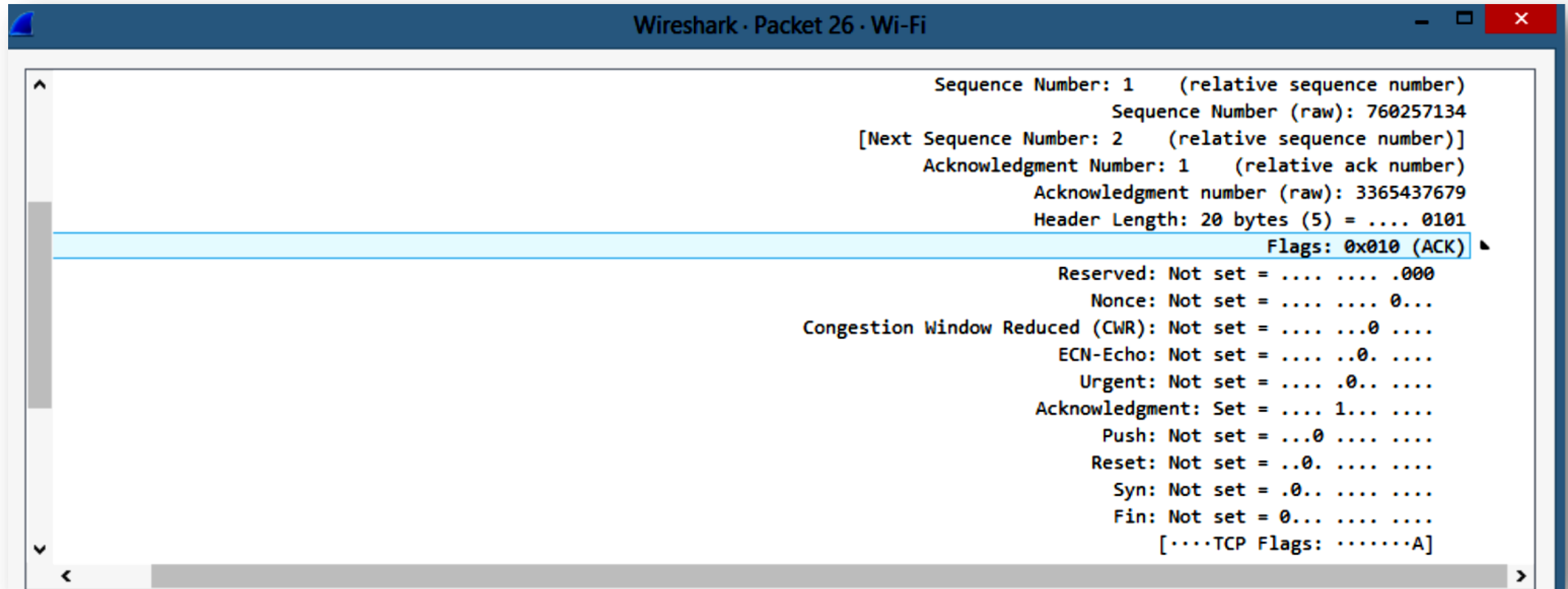
1. Firewalls
2. Intrusion detection systems (IDSs)
3. Intrusion prevention systems (IPSs)
4. Network address translators (NATs) → Port-Translating NAT (PAT)
5. WAN optimizers
6. Proxy server
7. Load balancers

## ■ In TCP connection:

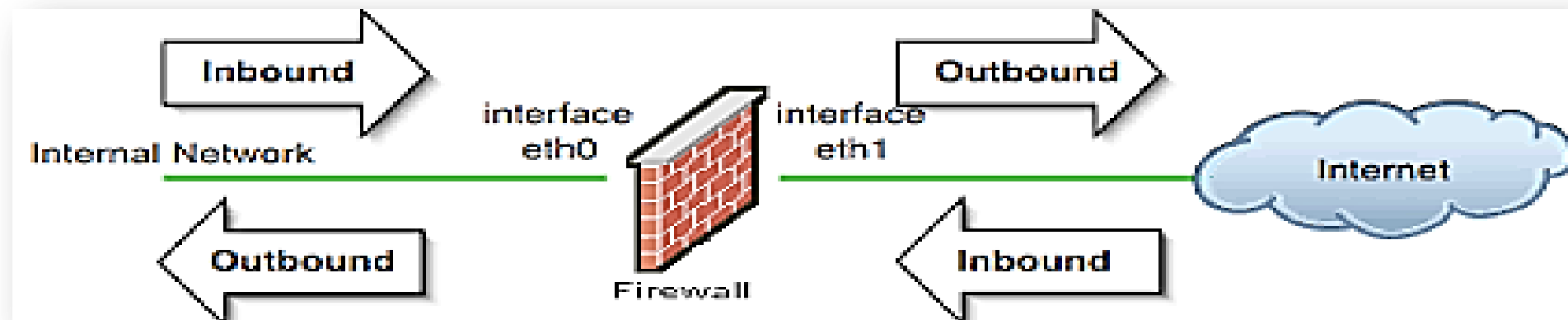
- Flags are used to indicate a particular state of connection or to provide some additional useful information like troubleshooting purposes or to handle a control of a particular connection.
- Most commonly used flags are “SYN”, “ACK” and “FIN”. Each flag corresponds to 1 bit information. Where set flag = 1, unset = 0



| Flag |                           | Binary |   |   |   |   |   |   |   | Decimal |
|------|---------------------------|--------|---|---|---|---|---|---|---|---------|
| CWR  | Congestion Window Reduced | 1      | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 128     |
| ECE  | ECN-Echo                  | 0      | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 64      |
| URG  | Urgent                    | 0      | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 32      |
| ACK  | Acknowledgement           | 0      | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 16      |
| PSH  | Push                      | 0      | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 8       |
| RST  | Reset                     | 0      | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 4       |
| SYN  | Syn                       | 0      | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2       |
| FIN  | Fin                       | 0      | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1       |



- **Inbound** means someone else from outside of your computer initiate the connection to your computer, so the traffic starts flowing inward to your machine.
- **Outbound** means you initiate the connection and the traffic starts flowing outward of your computer to the destination you intended. Example you connect to a server.
- You can stop inbound and outbound packets and modify them, before they will be processed by Windows network stack.



## Write:

```
from scapy.all import *

data = 'Everyone s Smart University'
a = IP(dst='129.132.2.21') / UDP() / Raw(load=data)

send(a)
wrpcap('ESU_udp.pcap', a, append=True)
a.show()
print(a)

data2 = 'Network programming class'
packet = Ether()/IP(src='129.132.2.1', dst='129.132.2.21')/TCP(sport=50, dport=100, flags='A')/Raw(load=data2)
sendp(packet)
wrpcap('ESU_tcp.pcap', packet, append=True)
packet.show()
print(packet)
```

## Read:

```
from scapy.all import *

packets = rdpcap('ESU_udp.pcap')
print('packets =', packets)

for pkt in packets:
    if TCP in pkt:
        pdataTCP = bytes(pkt[TCP].payload)
        print('TCP')
        print('pdataTCP =', pdataTCP)
    if Raw in pkt:
        pdataRaw = bytes(pkt[Raw].payload)
        print('Raw')
        print('pdataRaw =', pdataRaw)
    if UDP in pkt:
        pdataUDP = bytes(pkt[UDP].payload)
        print('UDP')
        print('pdataUDP =', pdataUDP)
    else:
        pdataNone = ""
        print('None')
        print('pdataNone =', pdataNone)
```

- Computer networking device (Or software): that transforms, inspects, filters, and manipulates traffic for purposes other than packet forwarding.
- Any intermediary network device performing functions other than standard functions of an IP forwarding between two end hosts.
- These extraneous functions have interfered with application performance and have been criticized for violating "important architectural principles" such as the end-to-end principle.
- Examples of middleboxes include firewalls, network address translators (NATs), load balancers, and deep packet inspection (DPI) boxes.

- **Special-purpose operations on packets:**
  - Service Providers: DPI, load-balancer, ...
  - Enterprises: Firewalls, NAT, ...
- **Physical appliances (middle-boxes) are expensive.**
- **Virtual appliances (VNFs) difficult to scale.**



- Middleboxes are widely deployed across both private and public networks.
- Cellular networks use middleboxes to ensure scarce network resources are used efficiently as well as to protect client devices.
- Dedicated middlebox hardware is widely deployed in enterprise networks to improve network security and performance, however, even home network routers often have integrated firewall, NAT, or other middlebox functionality (*has some middlebox capabilities!*).
- Service Providers (ISPs)

**Primarily deployed for security and performance benefits.**

Firewalls

Application Firewalls

Intrusion Detection Systems (IDS)

Intrusion Prevention Systems (IPS)

Proxy/Caches

WAN Optimizers

Protocol Accelerators

**Many other uses too!**

Billing and usage monitoring, asset tracking, Network Address Translation, protocol converters (6to4/4to6)...

- **Reasons driving the deployment of middleboxes come in two main categories:**
  - (1) Security to enhance the visibility of network traffic and enable the enforcement of security policies.
  - (2) Performance enhancements through traffic shaping, caching and transparent proxying.

- Compared to forwarding devices such as switches and routers, middleboxes are complex.
- Indeed, they operate on flows of packets at multiple layers of the network stack, from the network layer to the application layer, and do so at line rate.
- Middleboxes interfere with end-to-end packet transmission, application functionality, and restricting or preventing end host applications from functioning properly.

### Key differences between middleboxes and routers:

Middleboxes are often **stateful**. They remember fine-grained data that is updated as frequently as every packet or every connection.

Middleboxes perform **complex and varied operations** on packets. There are new categories of middleboxes on the market every year.

## Middlebox interference can be categorized into three types:

1. Middleboxes intentionally drop or filter packets according to policies:  
For example, network administrators filter P2P file sharing traffic to avoid the legal implications of copyrighted content .
2. Middleboxes modify the content of packets:  
Some web proxies modify HTTP headers to control meta information between client and server (e.g., cache preferences).
2. Middleboxes also inject forged packets, e.g., for blocking purposes:  
A notorious example is the Great Firewall of China (GFC) that blocks specific sites by injecting spoofed DNS responses, with obvious consequences in terms of Internet censorship

- 1) Firewalls
- 2) Intrusion detection systems (IDSs)
- 3) Intrusion prevention systems (IPSs)
- 4) Network address translators (NATs) → Port-Translating NAT (PAT)
- 5) WAN optimizers
- 6) Proxy server
- 7) Load balancers

### 1) Firewalls:

- Filter traffic based on a set of predefined security rules defined by a network administrator.
- IP firewalls reject packets "based purely on fields in the IP and transport headers (e.g., disallow incoming traffic to certain port numbers, disallow any traffic to certain subnets etc.)".
- Other types of firewalls may use more complex rule sets, including those that inspect traffic at the session or application layer.

<https://docs.rackspace.com/support/how-to/best-practices-for-firewall-rules-configuration/>

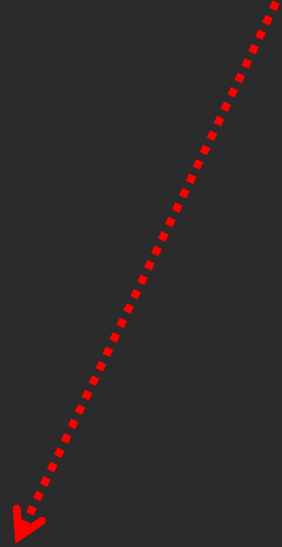
- **So, Firewall filters packet-by-packet, based on:**
  - *Source and destination IP addresses and port #'s*
  - *TCP SYN and ACK bits; ICMP message type*
  - *Deep packet inspection on packet contents (DPI)*
- **Firewall applies a set of rules to each packet**
  - *To decide whether to permit or deny the packet*
- **Each rule is tested on the packet**
  - *Comparing headers, deciding whether to allow/deny*
- **Order matters**
  - *Once packet matches a rule, the decision is done*

- **Block all packets with IP protocol field = 17 and with either src port or dst port = 23**
  - *All incoming and outgoing UDP flows blocked*
  - *All Telnet connections are blocked*
- **Block inbound TCP packets with SYN but no ACK**
  - *Prevents external clients from making TCP connections with internal clients*
  - *But allows internal clients to connect to outside*
- **Block all packets with TCP port of *Fortnite***
  - *Ex: 80, 443 and 5222 all TCP*

```
from scapy.all import *
global my_keyword1, my_keyword2
my_keyword1 = TCP
my_keyword2 = UDP

def pkt_callback(pkt):
    global my_keyword1, my_keyword2
    if my_keyword1 in pkt:
        pkt.show()
        print("recognized TCP packet")
        print('Destination port =', pkt[TCP].dport)
        print('Source port =', pkt[TCP].sport)
    elif my_keyword2:
        pkt.show()
        print("recognized UDP packet")
        print('Destination port =', pkt[UDP].dport)
        print('Source port =', pkt[UDP].sport)
    else:
        pkt.show()
        print("This packet is neither TCP nor UDP")

# Spkt = sniff(iface='Qualcomm Atheros AR9485WB-EG Wireless Network Adapter', count=4, filter="udp sport 80 or dport 80 ", prn=lambda x:x.summary())
Spkt = sniff(iface='Qualcomm Atheros AR9485WB-EG Wireless Network Adapter', filter=" tcp or udp", count=2, prn=pkt_callback)
Spkt.summary()
```



- The sniff() function listens for an infinite period of time until the user interrupts.
- To restrict the number of packets to be captured sniff() allows a **count** parameter.
- You can also filter packets while sniffing using the filter parameter. It uses a Berkeley Packet Filter (**BPF**) syntax.
- You can filter any packet on the basis of source/destination IP address, port number, protocol and lot more by using the **BPF** syntax.
- We can also sniff packets offline from pcap files by running the following command:  
**sniff(offline="<file name>")** , ex: "**<file name>** = 1.pcap
- When scapy sniffs packets, it generally sniffs from all of your network interfaces. However, we can explicitly mention the interfaces that we would like to sniff on using the **iface** parameter.

- sniff() function has another interesting parameter called **prn** that allows you to pass a function that executes with each packet sniffed. This allows us to do some custom actions with each packet sniffed. *sniff(prn=lambda x:x.summary(), count=5)*
- In order for your program/script to format and return the packet info as you wish, the sniff function passes the packet object as the one and only argument into the function you specify in the sniff's prn argument. This gives us the option to do some fun stuff (not just formatting) with each packet sniffed

- Here are some conditional statements to very clearly tell me what ARP traffic in my computer is seeing.
- Address Resolution Protocol (ARP) is a protocol or procedure that connects an ever-changing Internet Protocol (IP) address to a fixed physical machine.

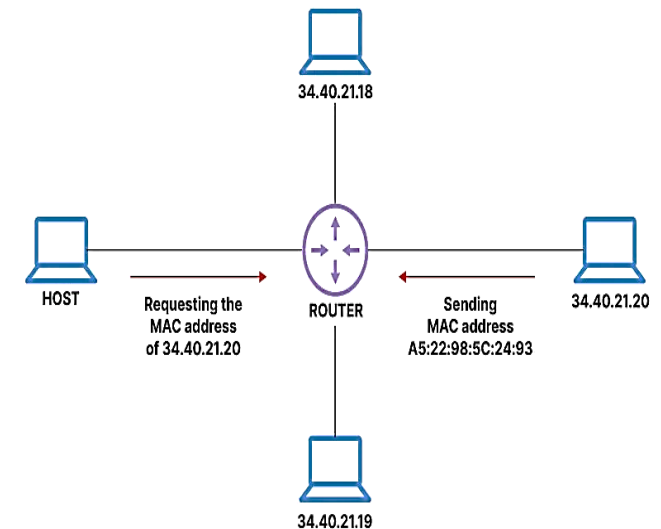
Small letters, ex: ip, arp, tcp, udp, icmp

```
from collections import Counter
from scapy.all import ARP, sniff

def arp_display(pkt):
    if pkt[ARP].op == 1: #who-has (request)
        return f"Request: {pkt[ARP].psrc} is asking about {pkt[ARP].pdst}"
    if pkt[ARP].op == 2: #is-at (response)
        return f"Response: {pkt[ARP].hwsrc} has address {pkt[ARP].psrc}"

sniff(prn=arp_display, filter="arp", store=0, count=10)
```

## How Address Resolution Protocol (ARP) Works



```
from scapy.all import *

def filter_TCP_flags(pkt):
    if pkt.haslayer(TCP):
        # print('Packet = ', pkt)
        # packetstr = "".join(map(chr, bytes(pkt)))
        # print('Packet AS string= ', packetstr)
        print('TCP flag is ', pkt[TCP].flags)
        if 'P' in pkt[TCP].flags:
            print('Could you drop this packet?')

pkts = sniff(filter='tcp and (tcp.flags.syn or tcp.flags.ack)', prn=filter_TCP_flags, count=30)
```

If you wonder why the packets cannot be dropped: The software does not intercept the traffic, it instead shares a view on the network data through promiscuous mode.

- You can use *haslayer()* method:

```
def pkt_callback(pkt):  
    if pkt.haslayer(TCP):  
        pkt.show()
```

Q: Filter packets of a particular website in Python (using Scapy)? Suppose that you have a list of possible IPs (used for load balancing) of the website. And, you want to filter packets for all those IPs. How can you do that?

<https://biot.com/capstats/bpf.html>

```
from scapy.all import *

global addresses, addressesBlocked

addresses = ["239.255.255.250", "216.58.212.110", "142.251.37.163", "142.251.37.35", "142.250.200.238"]
addressesBlocked = []

def pkt_callback(pkts):
    global addresses, addressesBlocked
    print(pkts[IP].dst)
    if pkts[IP].dst in addresses:
        if pkts[IP].dst in addressesBlocked:
            a = 4
        else:
            addressesBlocked.append(pkts[IP].dst)

Spkt = sniff(iface='Qualcomm Atheros AR9485WB-EG Wireless Network Adapter', count=3, prn=pkt_callback, store=0)

if len(addressesBlocked):
    print('Blocked IPs : ', addressesBlocked)
else:
    print('*** NO Blocked IPs! ***')

# store=0 ---> discard sniffed packets
```

### 2) Intrusion detection systems (IDSs):

- Monitor traffic and collect data for offline analysis for security anomalies.
- Unlike firewalls, IDSs do not filter packets in real time, as they are capable of more complex inspection and must decide whether to accept or reject each packet as it arrives.

<https://www.dnsstuff.com/network-intrusion-detection-software>

```
def detect_TCPflood(self, packet):
    if packet.haslayer(TCP):
        pkt_src=packet[IP].src
        pkt_dst=packet[IP].dst
        stream = pkt_src + ':' + pkt_dst

        if stream in type(self).__ip_cnt_TCP:
            type(self).__ip_cnt_TCP[stream] += 1
        else:
            type(self).__ip_cnt_TCP[stream] = 1

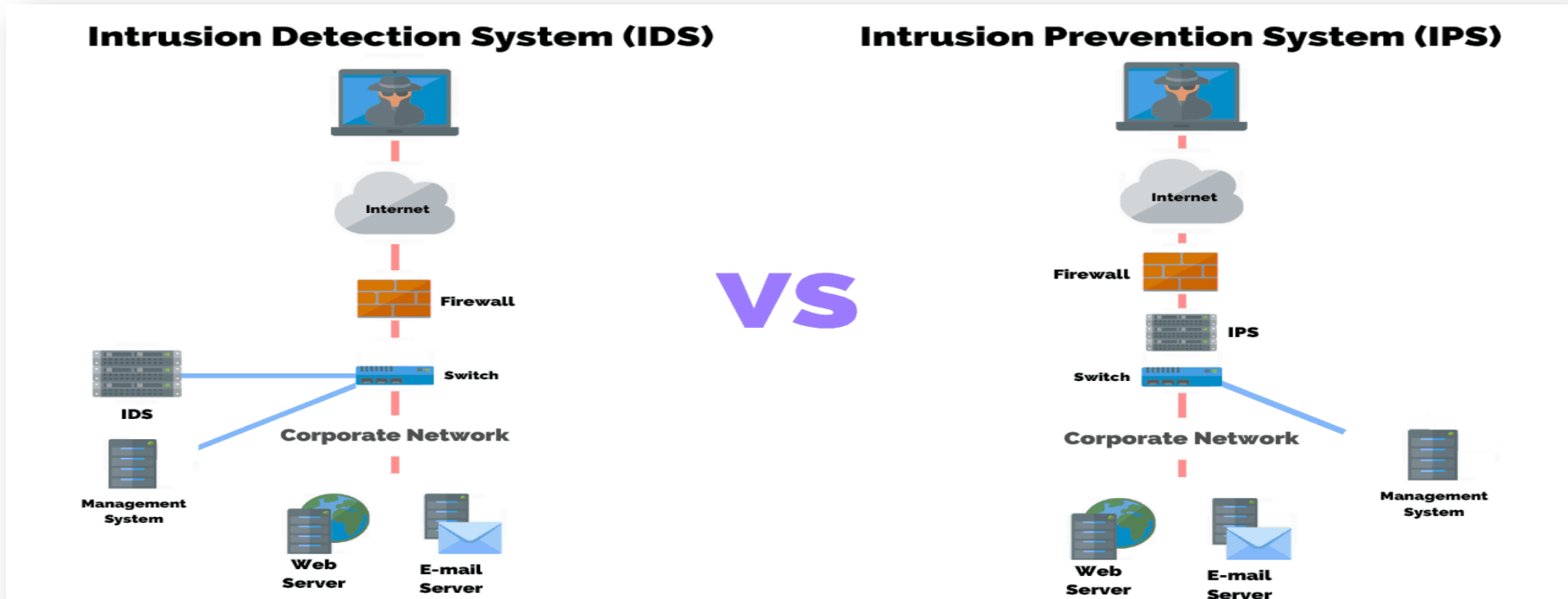
        for stream in type(self).__ip_cnt_TCP:
            pkts_sent = type(self).__ip_cnt_TCP[stream]
            if pkts_sent > type(self).__THRESH:
                src = stream.split(':')[0]
                dst = stream.split(':')[1]
                print("Possible Flooding Attack from %s --> %s"%(src, dst))
```

```
Terminal: Local x Local (2) x +
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35798, ['PSH', 'ACK']
IP Packet: 192.168.1.19 ==> 138.201.34.122 , 2022-07-18 18:49:47 , Port: 35798 --> 443, ['ACK']
IP Packet: 192.168.1.19 ==> 138.201.34.122 , 2022-07-18 18:49:47 , Port: 35798 --> 443, ['PSH', 'ACK']
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['ACK']
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['ACK']
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['ACK']
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['ACK']
IP Packet: 192.168.1.19 ==> 138.201.34.122 , 2022-07-18 18:49:47 , Port: 35802 --> 443, ['ACK']
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['ACK']
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['ACK']
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['ACK']
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['ACK']
Possible Flooding Attack from 138.201.34.122 --> 192.168.1.19
IP Packet: 192.168.1.19 ==> 138.201.34.122 , 2022-07-18 18:49:47 , Port: 35802 --> 443, ['ACK']
Possible Flooding Attack from 138.201.34.122 --> 192.168.1.19
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['ACK']
Possible Flooding Attack from 138.201.34.122 --> 192.168.1.19
IP Packet: 192.168.1.19 ==> 138.201.34.122 , 2022-07-18 18:49:47 , Port: 35802 --> 443, ['ACK']
Possible Flooding Attack from 138.201.34.122 --> 192.168.1.19
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['ACK']
Possible Flooding Attack from 138.201.34.122 --> 192.168.1.19
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['ACK']
Possible Flooding Attack from 138.201.34.122 --> 192.168.1.19
IP Packet: 138.201.34.122 ==> 192.168.1.19 , 2022-07-18 18:49:47 , Port: 443 --> 35802, ['PSH', 'ACK']
```

### 3) Intrusion prevention systems (IPSs):

- is a network security tool (which can be a hardware device or software) that continuously monitors a network for malicious activity and takes action to prevent it, including reporting, blocking, or dropping it, when it does occur.
- Intrusion prevention systems expand on the capabilities of intrusion detection systems (IDS), which serve the fundamental purpose of monitoring network and system traffic.
- What makes intrusion prevention systems more advanced than intrusion detection systems is that IPS are located in-line (directly in the path in which the source and destination communicate) and have the capability to prevent or block the malicious activity that is occurring.

<https://digitalguardian.com/dskb/intrusion-prevention-system>



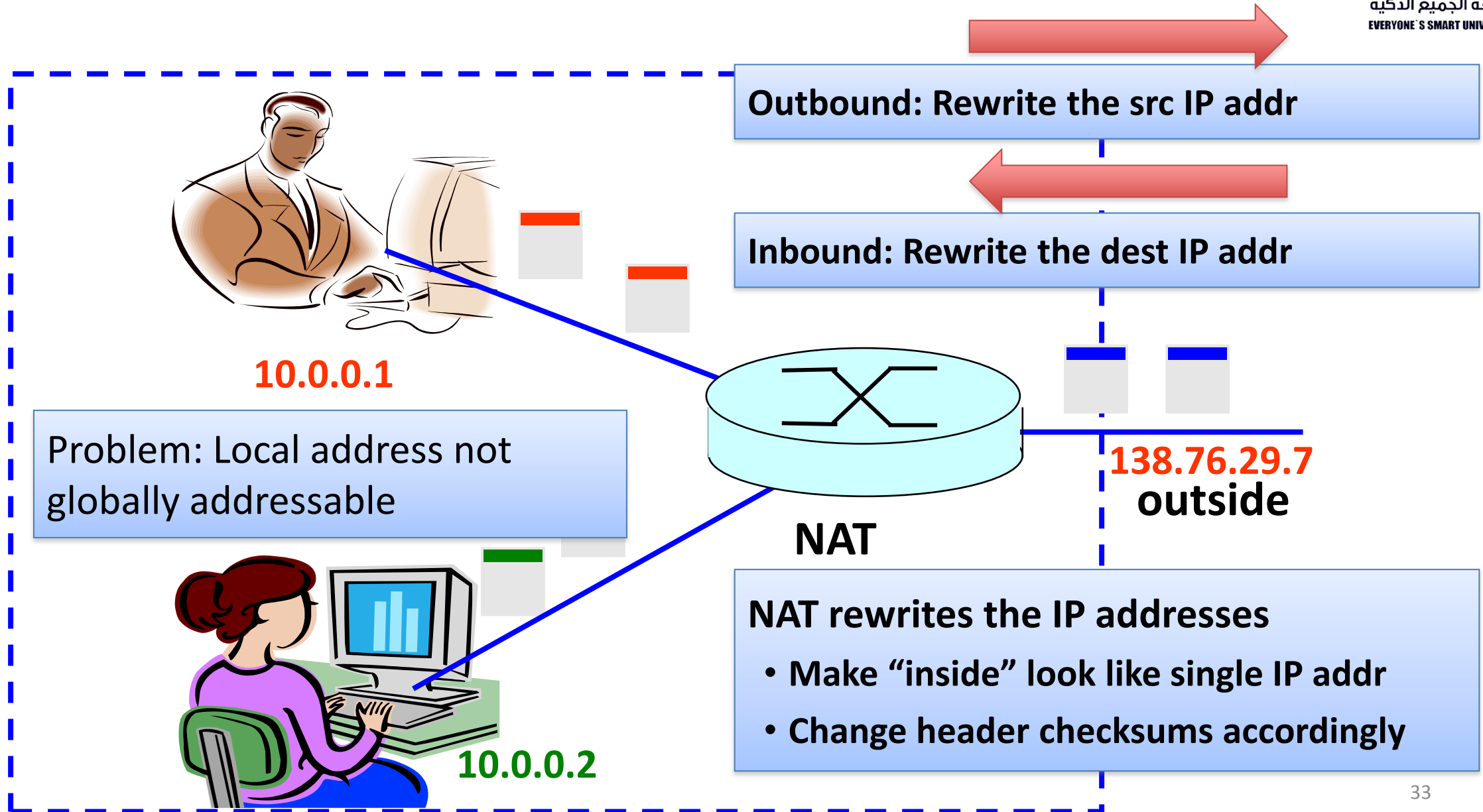
- Intrusion prevention systems are usually located behind a firewall to function as another filter for malicious activity.

<https://purplesec.us/intrusion-detection-vs-intrusion-prevention-systems/>

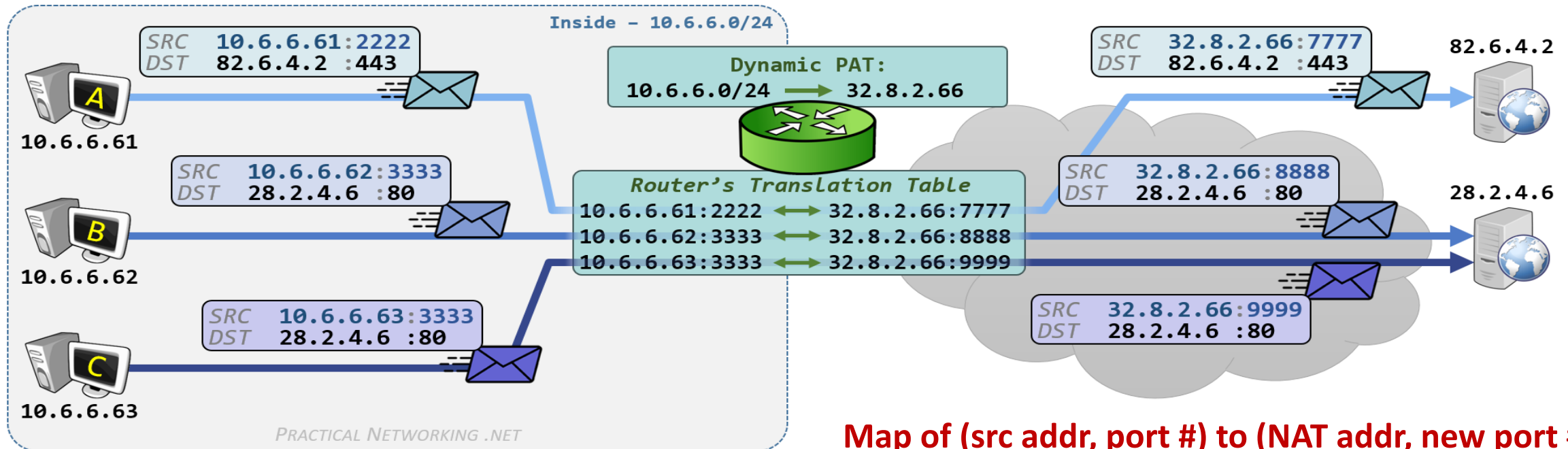
### 4) Network address translators (NATs):

- replace the source and/or destination IP addresses of packets that traverse them.
- Typically, NATs are deployed to allow multiple end hosts to share a single IP address: hosts "behind" the NAT are assigned a private IP address and their packets destined to the public Internet traverse a NAT, which replaces their internal private address with a shared public address. These are widely used by cellular network providers to manage scarce resources.

# Examples of Middleboxes (NAT)



**Port Address Translation (PAT):** is an extension of Network Address Translation (NAT) that permits multiple devices on a LAN to be mapped to a single public IP address to conserve IP addresses.



### 5) WAN optimizers:

- Improve bandwidth consumption and perceived latency between endpoints.
- Typically deployed in large enterprises, WAN optimizers are deployed near both sending and receiving endpoints of communication; the devices then coordinate to cache and compress traffic that traverses the Internet.

### 6) Proxy server:

- is a system or router that provides a gateway between users and the internet.
- Therefore, it helps prevent cyber attackers from entering a private network.
- It is a server, referred to as an “intermediary” because it goes between end-users and the web pages they visit online.

<https://www.varonis.com/blog/what-is-a-proxy-server>

- A proxy server is basically a computer on the internet with its own IP address that your computer knows. When you send a web request, your request goes to the proxy server first. The proxy server then makes your web request on your behalf, collects the response from the web server, and forwards you the web page data so you can see the page in your browser.
- When the proxy server forwards your web requests, it can make changes to the data you send and still get you the information that you expect to see. A proxy server can change your IP address, so the web server doesn't know exactly where you are in the world. It can encrypt your data, so your data is unreadable in transit. And lastly, a proxy server can block access to certain web pages, based on IP address.

### 5 REASONS WHY YOU SHOULD USE A **PROXY SERVER**



**1**

**Control internet usage** of employees or children.



**2**

**Bandwidth savings** and improved speeds.



**3**

**Privacy benefits.**



**4**

**Improved security.**

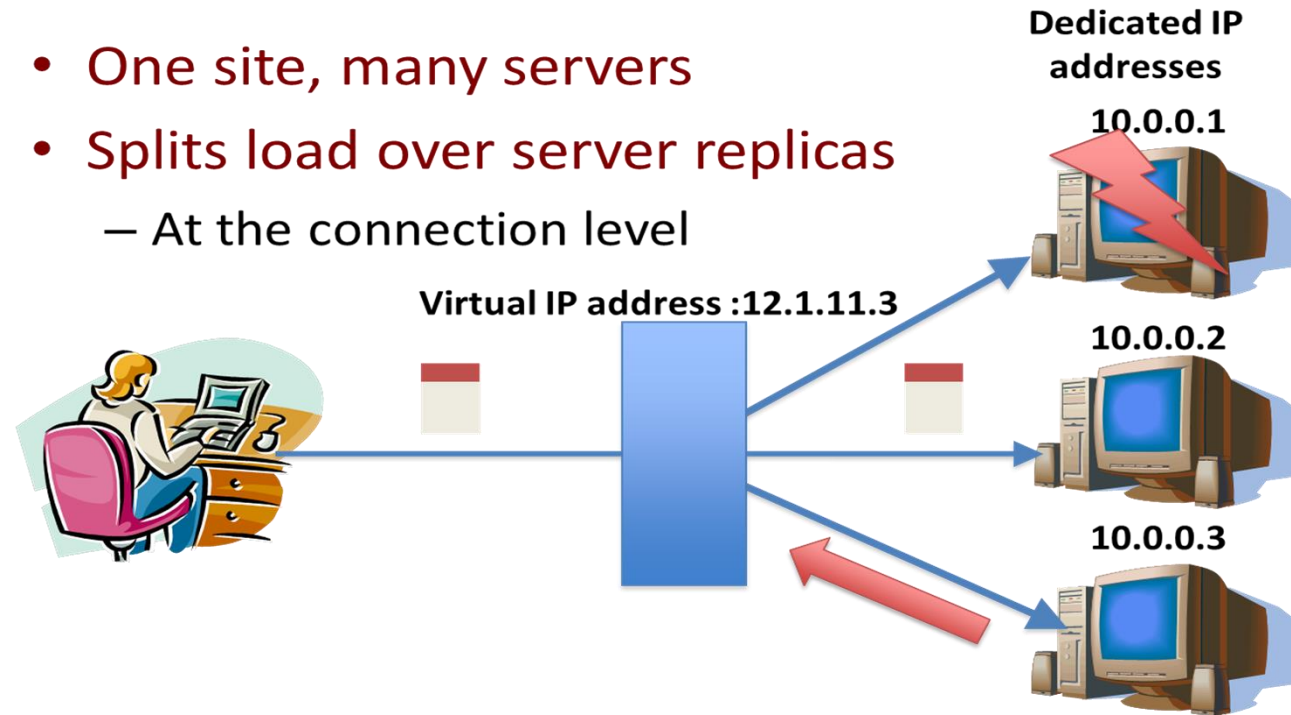


**5**

**Access to blocked resources.**

7) **Load balancers:** provide one point of entry to a service, but forward traffic flows to one or more hosts that actually provide the service.

- One site, many servers
- Splits load over server replicas
  - At the connection level



- Apply load balancing policies

## ■ What is load balancers in network? (Traffic cop)

A load balancer is a device (Or software) that acts as a reverse proxy and distributes network or application traffic across a number of servers. Load balancers are used to increase capacity (concurrent users) and reliability of applications.

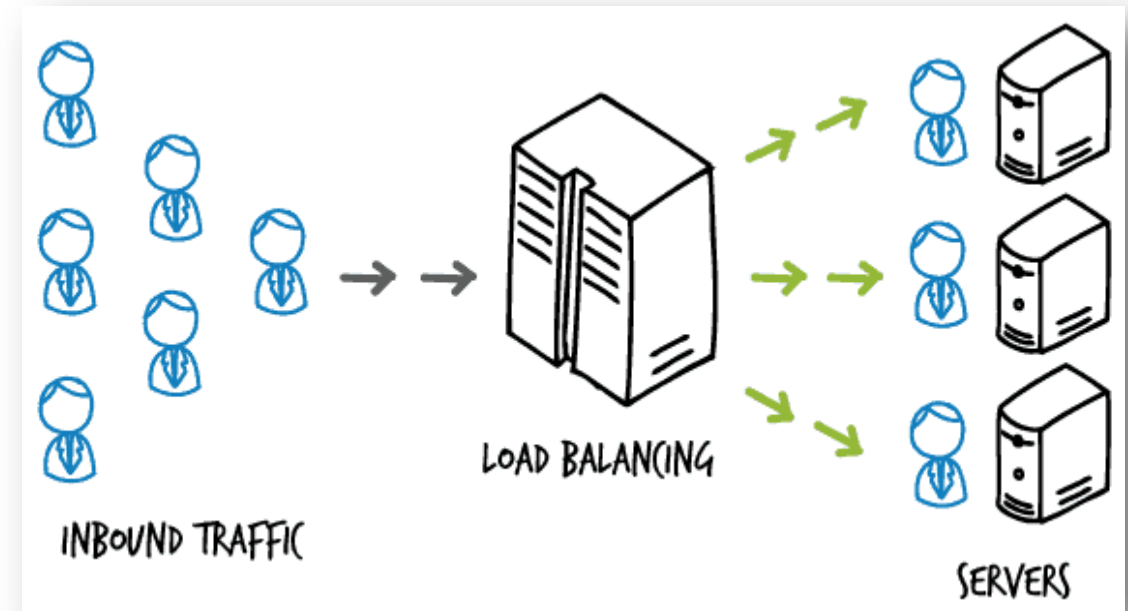
## ■ Benefits of Load Balancing:

1. Reduced downtime
2. Scalable
3. Redundancy
4. Flexibility
5. Efficiency



## Load balancer performs the following functions:

1. Distributes client requests or network load efficiently across multiple servers.
2. Ensures high availability and reliability by sending requests only to servers that are online.
3. Provides the flexibility to add or subtract servers as demand dictates.



## Load balancers typically come in two flavors:

- hardware-based and software-based.
- Vendors of hardware-based solutions load proprietary software onto the machine they provide, which often uses specialized processors.
- To cope with increasing traffic at your website, you have to buy more or bigger machines from the vendor.
- Software solutions generally run on **commodity hardware**, making them less expensive and more flexible.
- You can install the software on the hardware of your choice or in cloud environments like AWS EC2.

**\*\* Commodity hardware refers to cheap, standardized servers that are easy to buy off the shelf from any vendor.**