# 50.042 FCS Summer 2024
# Lecture 14 – Internet Protocols and Security

Felix LOH
Singapore University of Technology and Design

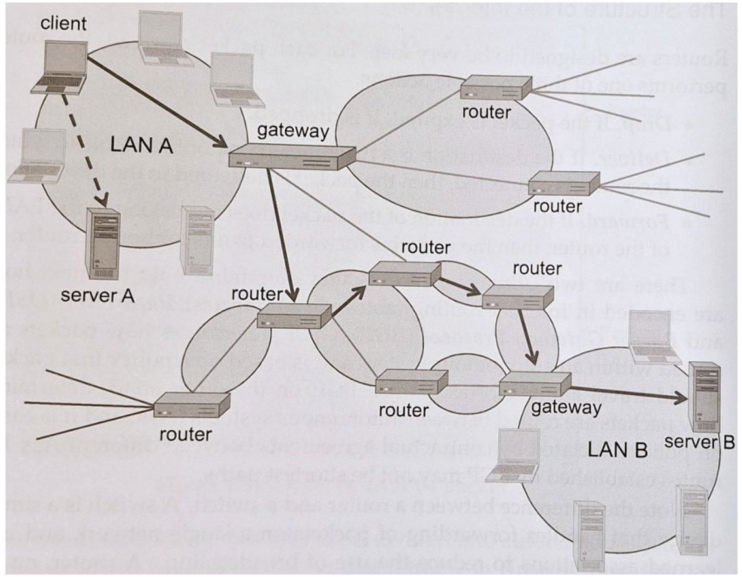
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

With selected materials adapted from: *Introduction to Computer Security (First Edition), by M. Goodrich and R. Tamassia*

## This lecture’s agenda

- In this lecture, we’ll discuss the organization and architecture of the Internet (which is basically a huge computer network), along with several commonly-used protocols
- We’ll also discuss a couple of possible attacks that can be performed on computer networks

# The Internet



- The Internet (or internet) is the global system of interconnected computer networks which uses a set of protocols to communicate between networks and devices
  - It is a network of networks
  - The set of protocols referenced above is the Internet Protocol suite (TCP/IP)

## The Internet

- The internet is a wide area network (WAN) composed of many machine and smaller networks distributed over great distances

- These smaller networks include private networks that are composed of computers located in relatively close proximity to each other
  - The private networks are known as local area networks (LANs)

- Communication occurs through sequences of formatted data units
  - These data units are called *segments/datagrams*, *packets* or *frames*, <u>depending on the layer</u>
  - But in general, these data units are all simply referred to as ‘packets’

## The TCP/IP model

| Layer Number | Layer Name | Some Protocols at this Layer | Protocol Data Unit | Addressing |
|---|---|---|---|---|
| 5 | Application | DNS, HTTP, HTTPS, etc. | Messages | N/A |
| 4 | Transport | TCP, UDP | Segments/Datagrams | Port numbers |
| 3 | Network (or Internet) | IP | Packets | IP addresses |
| 2 | Link (or Data Link) | Ethernet, Wi-Fi, ARP | Frames | MAC addresses |
| 1 | Physical | 10 Base T, 802.11, etc. | Bits | N/A |

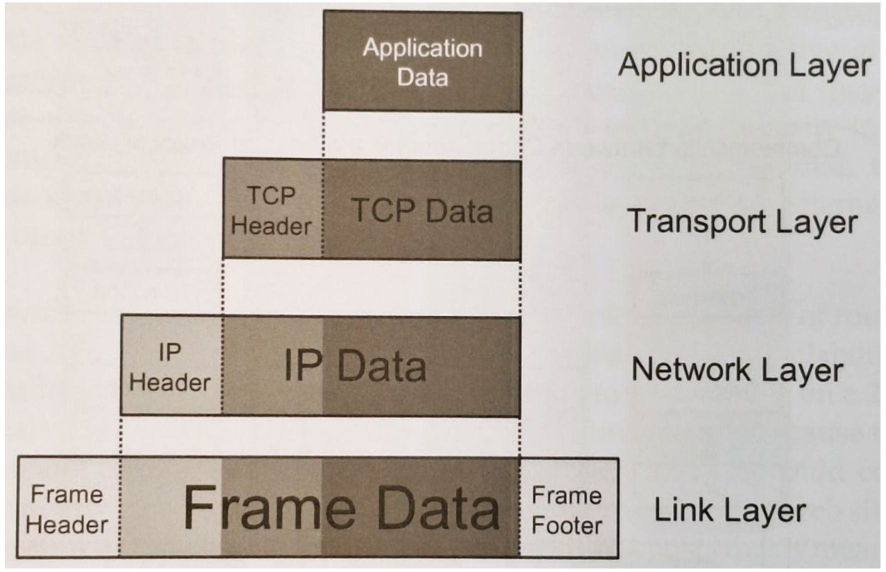
- The architecture of the Internet is modeled conceptually as being partitioned into layers
  - The number of layers depend on the model – in this class, we use the TCP/IP model which has 5 layers (these layers are collectively called the *Internet protocol stack*)
  - The OSI model defines 7 layers

## The TCP/IP model

- Each layer provides a set of service and functionality guarantees for the higher layers
- Where possible, each layer is independent of services and details from the higher layers – this allows for abstraction
- The interface that a lower layer provides to its higher layer is designed to disclose only the essential information of the lower layer that is needed by the higher layer
  - Lower level details are hidden from the higher levels

- This layered model helps system designers to code software that use the appropriate services (and provide the right service guarantees) without worrying about unnecessary implementation details

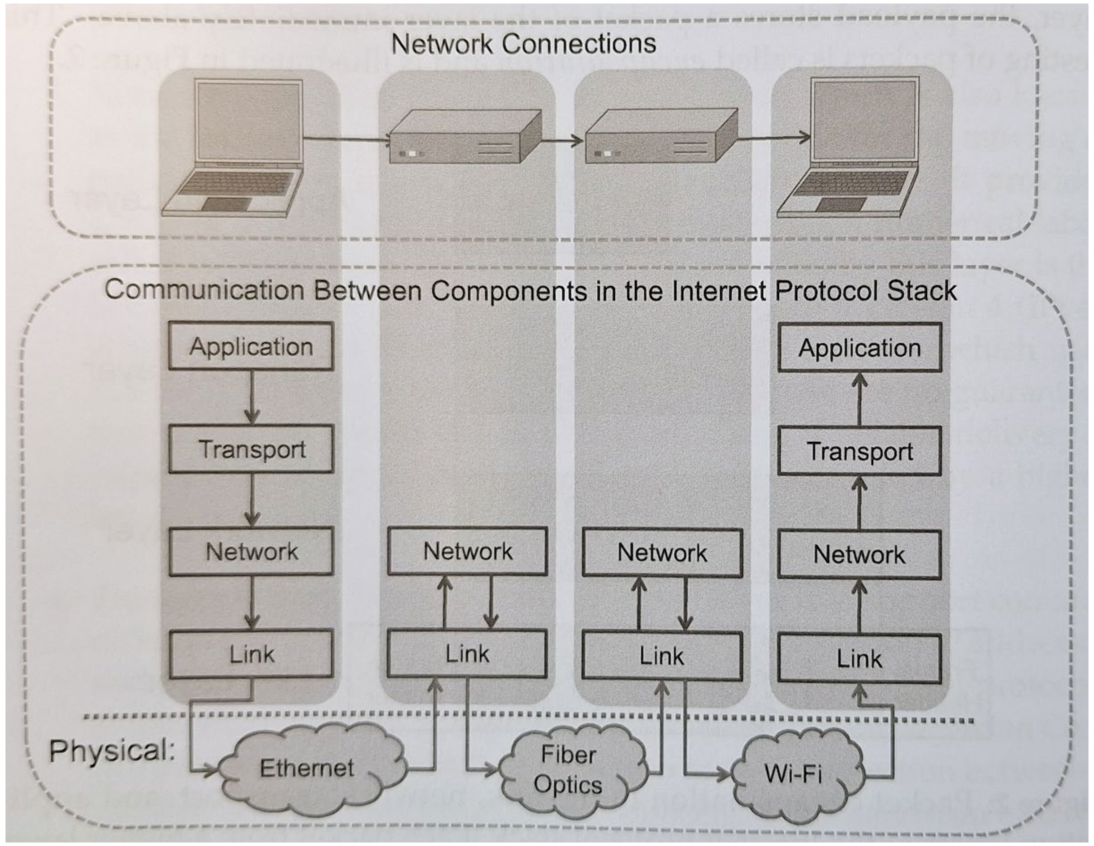
# Transmission of data through the layers



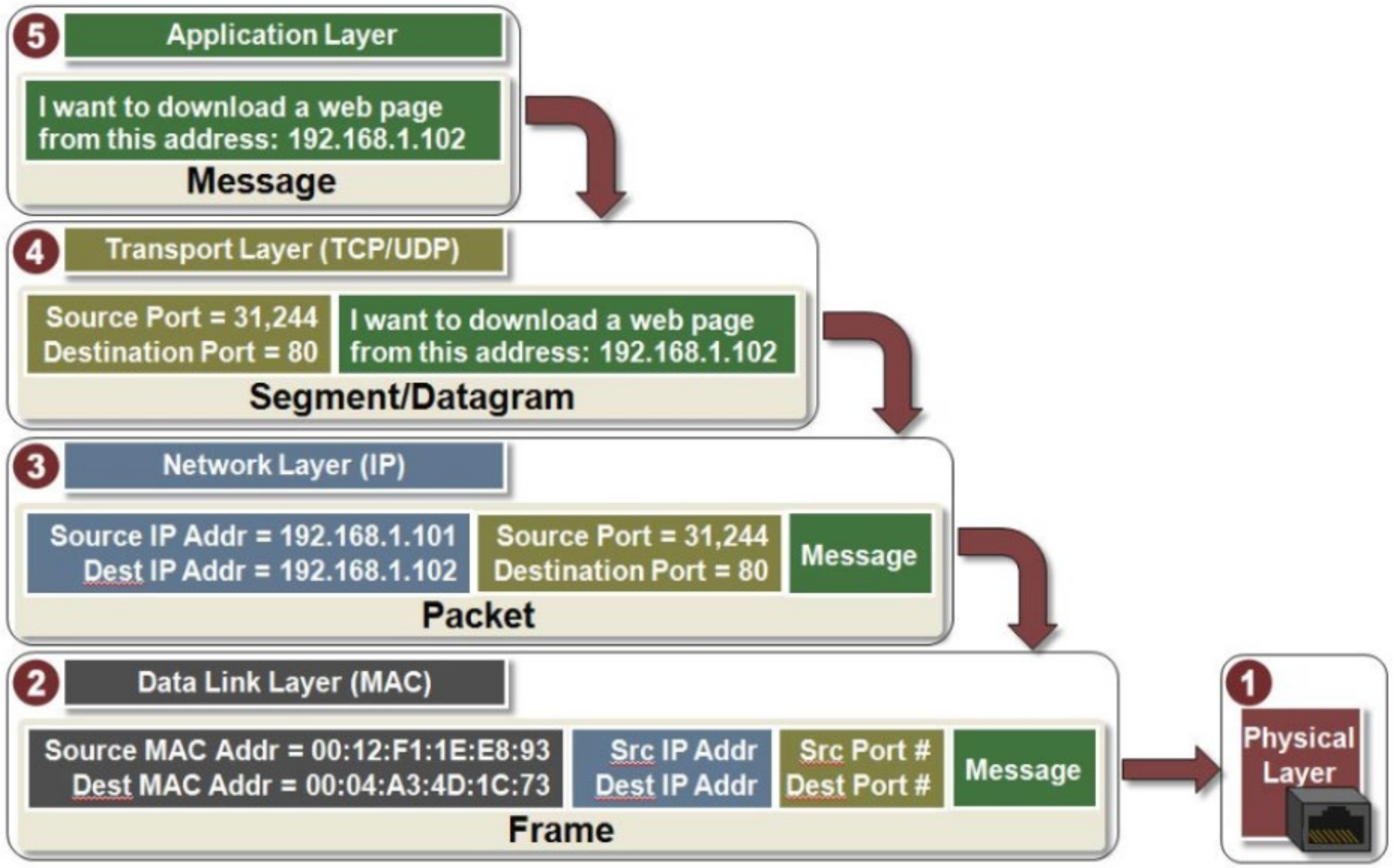
- Generally, each data unit is divided into two parts:
  - Header: metadata that specifies where the data unit is going and other details
  - Payload: the actual information that is being communicated
- For all except the topmost layer, the payload of a data unit stores a data unit of the immediate layer above – this is encapsulation

# Transmission of data through the layers

- For example, a web server transmitting content to the web browser of a client
  - It uses the HTTP application layer protocol
  - The HTTP data is encapsulated in the payload of a TCP transport layer segment
  - The segment is in turn encapsulated in the payload of an IP packet
  - The IP packet is finally contained in a link layer protocol frame (e.g. Ethernet frame), to be transmitted over some physical medium



# Transmission of data – another example


5
Application Layer
I want to download a web page from this address: 192.168.1.102
Message
4
Transport Layer (TCP/UDP)
Source Port = 31,244
Destination Port = 80
I want to download a web page from this address: 192.168.1.102
Segment/Datagram
3
Network Layer (IP)
Source IP Addr = 192.168.1.101
Dest IP Addr = 192.168.1.102
Source Port = 31,244
Destination Port = 80
Message
Packet
2
Data Link Layer (MAC)
Source MAC Addr = 00:12:F1:1E:E8:93
Dest MAC Addr = 00:04:A3:4D:1C:73
Src IP Addr
Dest IP Addr
Src Port #
Dest Port #
Message
Frame
1
Physical Layer

## Physical layer

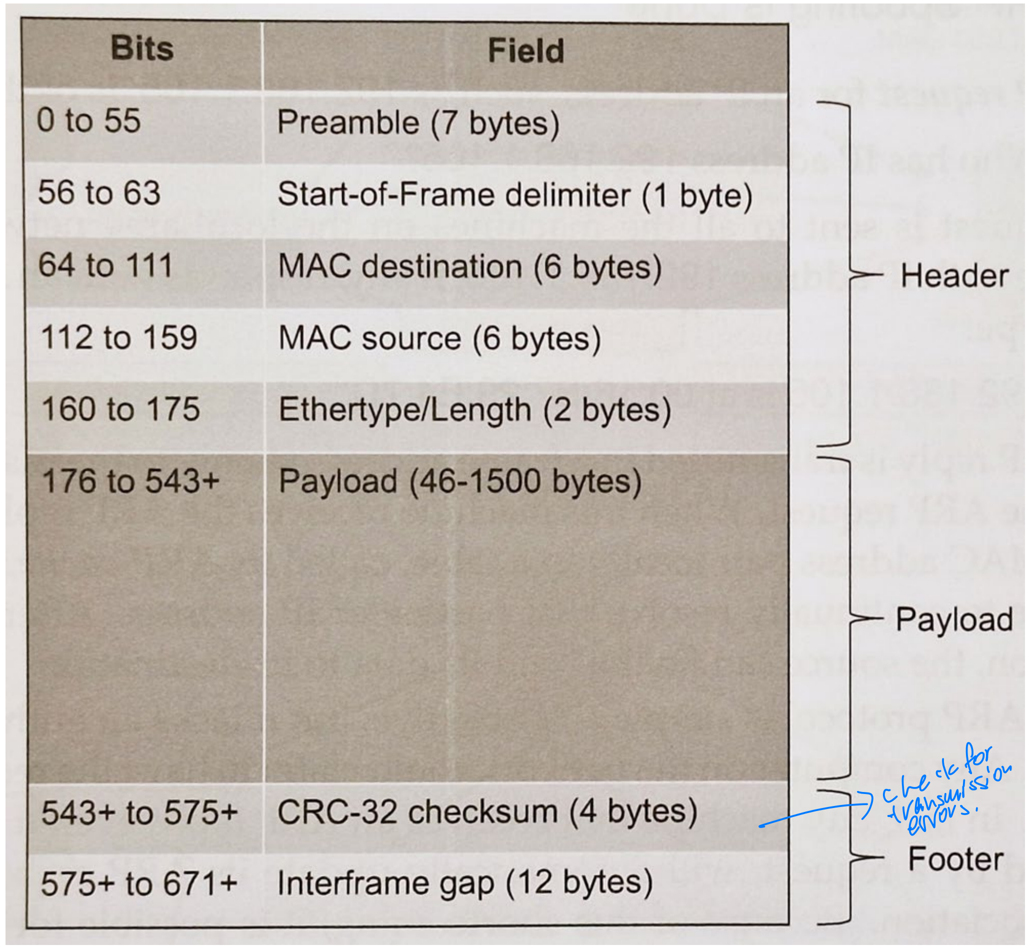
- Task of this layer:
  - Move the actual bits of data between the nodes of the network, on a best-effort basis

- This level deals with implementation details, such as whether the connections are done with copper wires, optical-fiber cables, wireless radio, etc.

## Link layer

- Also known as the data link layer
- Task of this layer:
  - Transfer data between a pair of network nodes, or between nodes in a LAN, and to detect errors that occur at the physical layer
- This level deals with the logical aspects of sending data across network links and how to find good routing paths in a LAN
  - The link layer uses 48-bit addresses, known as MAC addresses (**M**edia **A**ccess **C**ontrol), e.g. 05:23:4A:FD:7B:C8
- Data at this level is grouped into frames
- Includes protocols such as Ethernet, Wi-Fi and the Address Resolution Protocol (ARP)

# Ethernet Frame



| Bits | Field |
|---|---|
| 0 to 55 | Preamble (7 bytes) |
| 56 to 63 | Start-of-Frame delimiter (1 byte) |
| 64 to 111 | MAC destination (6 bytes) |
| 112 to 159 | MAC source (6 bytes) |
| 160 to 175 | Ethertype/Length (2 bytes) |
| 176 to 543+ | Payload (46-1500 bytes) |
| 543+ to 575+ | CRC-32 checksum (4 bytes) |
| 575+ to 671+ | Interframe gap (12 bytes) |

Header

Payload

check for transmission errors.

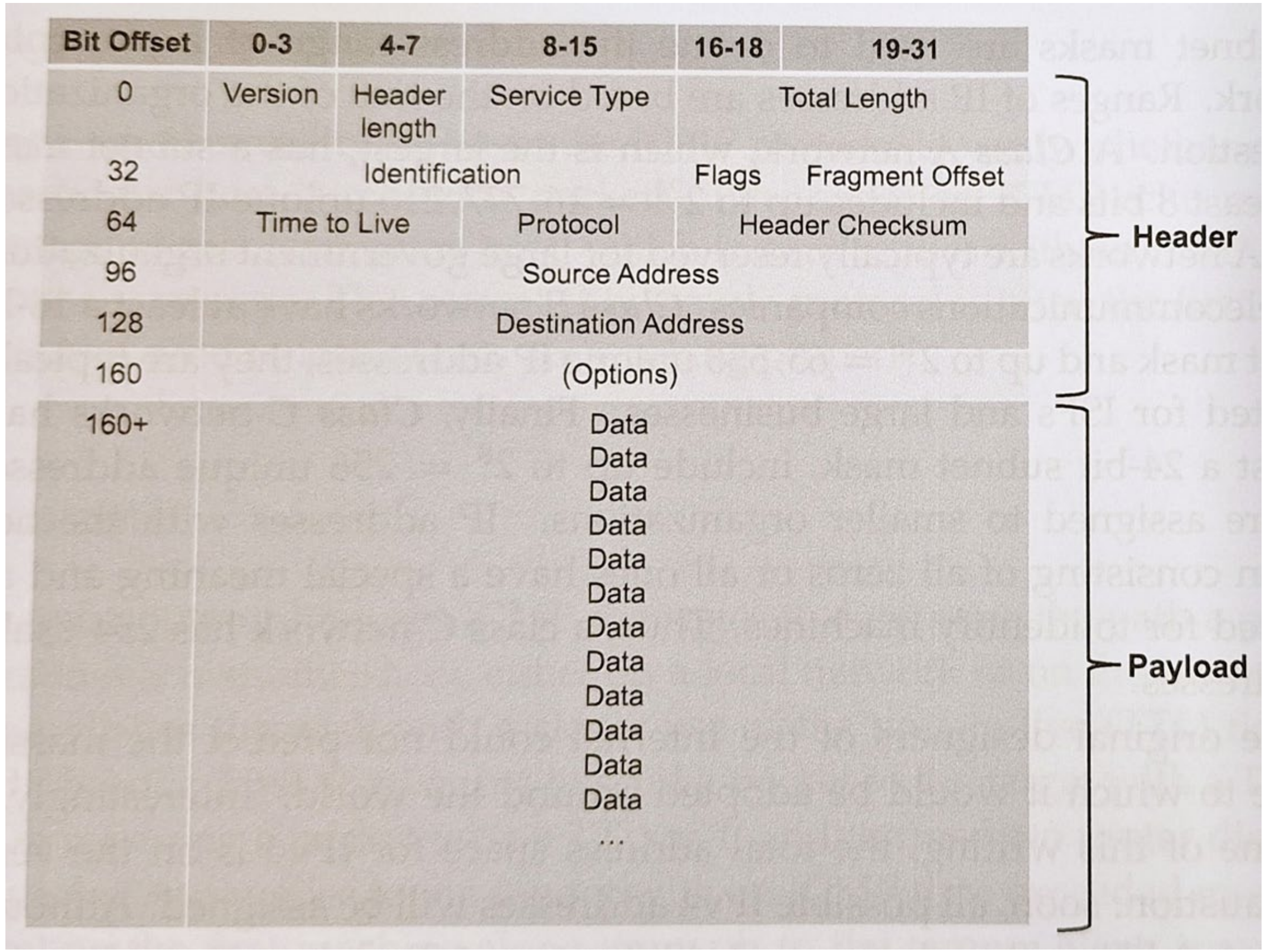
Footer

## Network layer

- Also known as the Internet layer
- Task of this layer:
    - Move data packets between any two hosts, on a best effort basis
    - The hosts can be in two different networks or LANs
- It provides a way to individually address each host, by using a numerical label called the *IP address*
- Data at this level is grouped into packets

- The main protocol used here is the Internet Protocol (IP), with two versions:
    - IPv4: uses 32-bit IP addresses (e.g. 192.168.1.1)
    - IPv6: uses 128-bit IP addresses (e.g. 2001:0db8:85a3:0000:0000:8a2e:0370:7334)

# IPv4 Packet



| Bit Offset | 0-3 | 4-7 | 8-15 | 16-18 | 19-31 |
|---|---|---|---|---|---|
| 0 | Version | Header length | Service Type | Total Length | |
| 32 | Identification | | | Flags | Fragment Offset |
| 64 | Time to Live | | Protocol | Header Checksum | |
| 96 | Source Address | | | | |
| 128 | Destination Address | | | | |
| 160 | (Options) | | | | |
| 160+ | Data<br>Data<br>Data<br>Data<br>Data<br>Data<br>Data<br>Data<br>Data<br>Data<br>Data<br>Data<br>… | | | | |

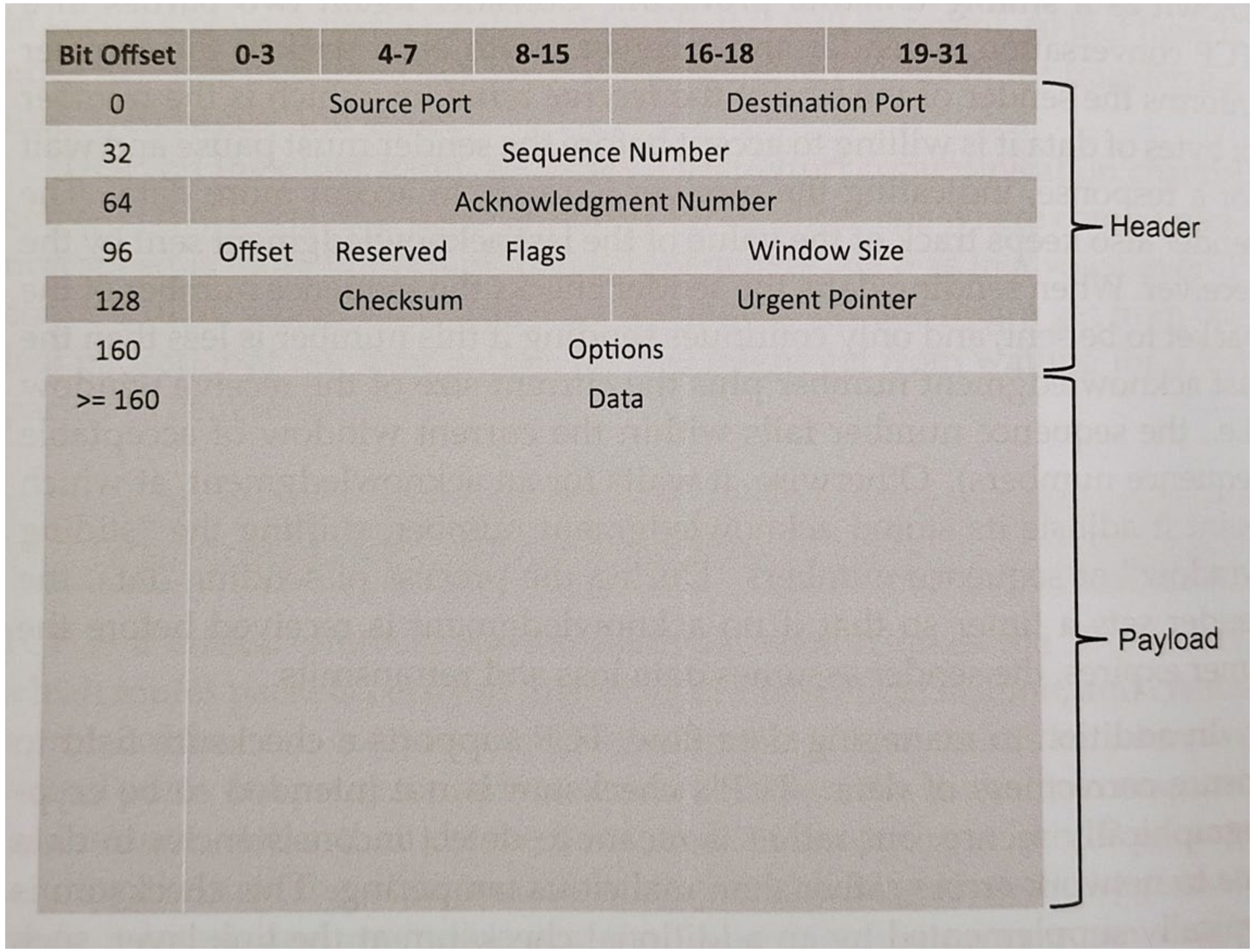
Header
Payload

## Transport layer

- Task of this layer:
  - Support communications and connections between applications, based on IP addresses and *ports*
  - Ports are 16-bit addresses for application-level protocols to use
- Data at this level is grouped into segments or datagrams, depending on the protocol (TCP or UDP)

- This layer provides two protocols:
  - Transmission Control Protocol (TCP) – establishes a virtual connection between a client and a server and guarantees delivery of all segments in an ordered manner
  - User Datagram Protocol (UDP) – a connectionless protocol that does not require prior setup; it delivers datagrams as quickly as possible but with no delivery guarantees

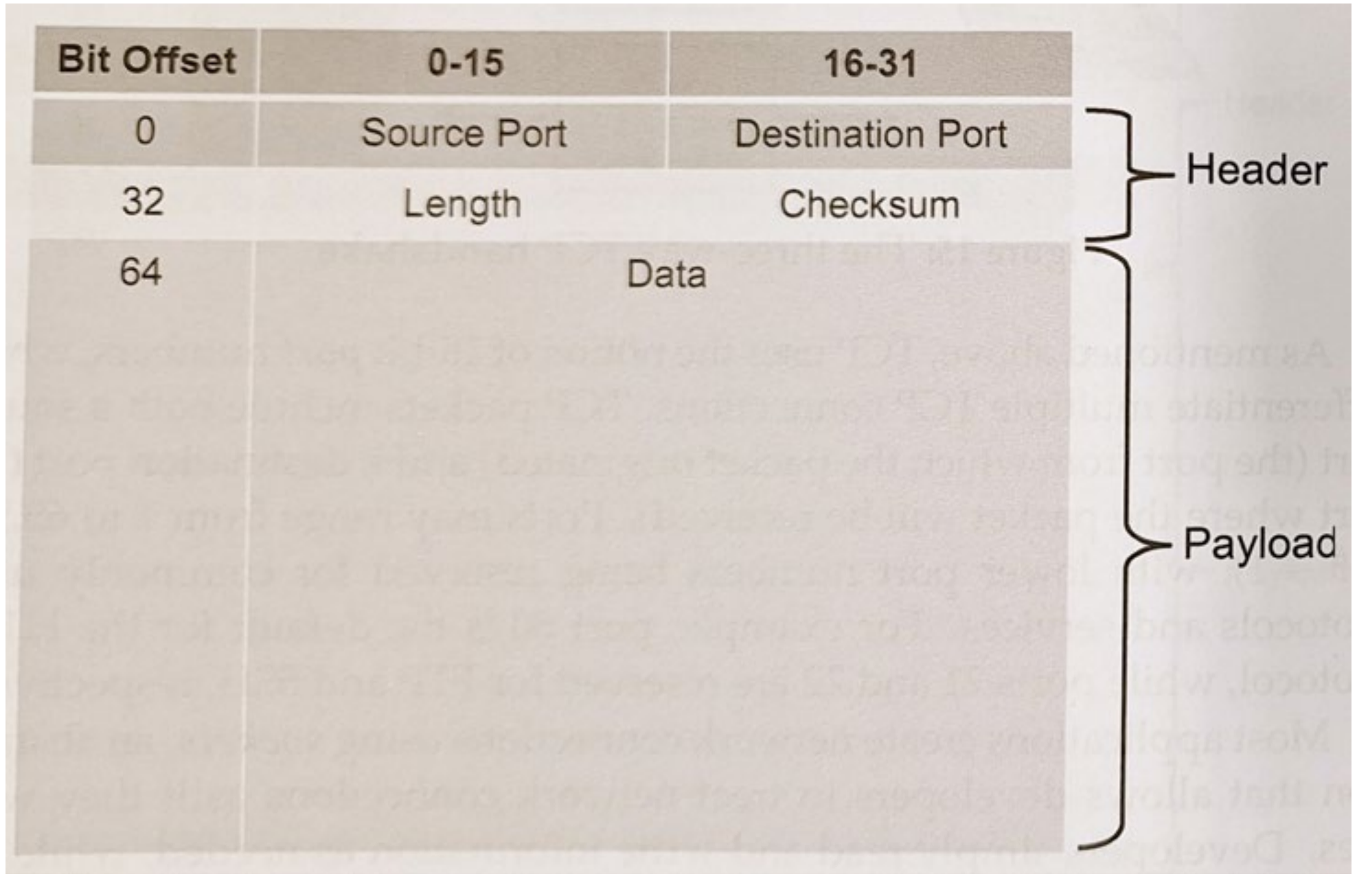
# TCP Segment



| Bit Offset | 0-3 | 4-7 | 8-15 | 16-18 | 19-31 |
|---|---|---|---|---|---|
| 0 | Source Port | | | Destination Port | |
| 32 | Sequence Number | | | | |
| 64 | Acknowledgment Number | | | | |
| 96 | Offset | Reserved | Flags | Window Size | |
| 128 | Checksum | | | Urgent Pointer | |
| 160 | Options | | | | |
| >= 160 | Data | | | | |

Header
Payload

# UDP Datagram



| Bit Offset | 0-15 | 16-31 |
|---|---|---|
| 0 | Source Port | Destination Port |
| 32 | Length | Checksum |
| 64 | Data | |

Header
Payload

## Application layer

- Task of this layer:
  - Provide protocols that support useful functions on the internet, based on the services provided by the transport layer

- Some commonly used protocols at this level, and their associated port numbers:
  - DNS (Domain Name System) — port 53, uses UDP
  - HTTP (Hypertext Transfer Protocol) — port 80, uses TCP
  - IMAP (Internet Message Access Protocol) — port 143, uses TCP
  - HTTPS (Hypertext Transfer Protocol over TLS/SSL) — port 443, uses TCP

## Network security issues

- Connecting computers and devices to a network, like the internet, provides huge benefits to society
- However, computer networking also enables some attacks on computers and information
- We’ll go over a couple of these attacks, but first let’s look at how computer networking impacts our principles of security

# Network security issues: confidentiality

- There is no requirement, for any of the layers, to keep the contents of the data units confidential
  - As a matter of fact, the basic standard protocols for each layer do not encrypt the headers or payloads of their data units

- As a consequence, if one wishes for network connections to be kept confidential, then encryption needs to be done explicitly
  - The encryption can be done at the application layer, e.g. using the HTTPS protocol
  - Alternatively, we can revise a lower layer protocol to include encryption, e.g. the IPsec specification

## Network security issues: integrity

- There are simple checksums in the headers (or footers) of the data units at each layer, for validation of data integrity of the payload and/or header contents
  - However, these checksums are **not** cryptographically secure
  - The checksums are meant for detection of errors during transmission
  - As such, the checksums do **not** provide integrity in the context of security

- So, if true integrity is required, this needs to be done at the application layer or with alternative protocols at the lower layers

## Network security issues: availability

- The internet is designed to tolerate the failures of hosts and networking devices (e.g. routers)
  - However, it is difficult to ensure that some particular networking device or host is available on a 24/7 basis
  - E.g. web servers can become unavailable if they are inundated with many data requests; these requests can come from legitimate sources, or can be generated by an attacker with the intent of executing a denial of service (DoS) attack on a server

- So, we need network applications that can scale with increases in the number of data requests, and/or block requests that originate from illegitimate sources

## Network security issues: authenticity

- The headers (and footers) of the data units used in the standard internet protocols do **not** have a dedicated field to place digital signatures
  - In the internet protocol stack, there is no notion of user identities

- So, if we require identities and digital signatures, we need to do this explicitly at the application layer or with alternative protocols at the lower layers

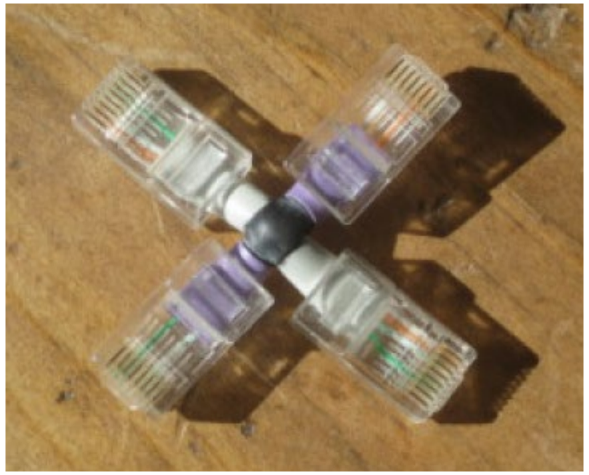
# Packet sniffing

- Packet sniffing is considered a passive attack (eavesdropping)
  - But packet sniffing *can* be legal to perform in certain situations:
  - E.g. packet sniffing on your own closed and separate LAN
  - E.g. as part of a penetration test on an organization’s network, where the organization has contracted you to perform such a test
  - Check your local laws before you attempt

- Packet sniffing works at the link layer and above
  - So strictly speaking, the packets being sniffed are actually *frames*

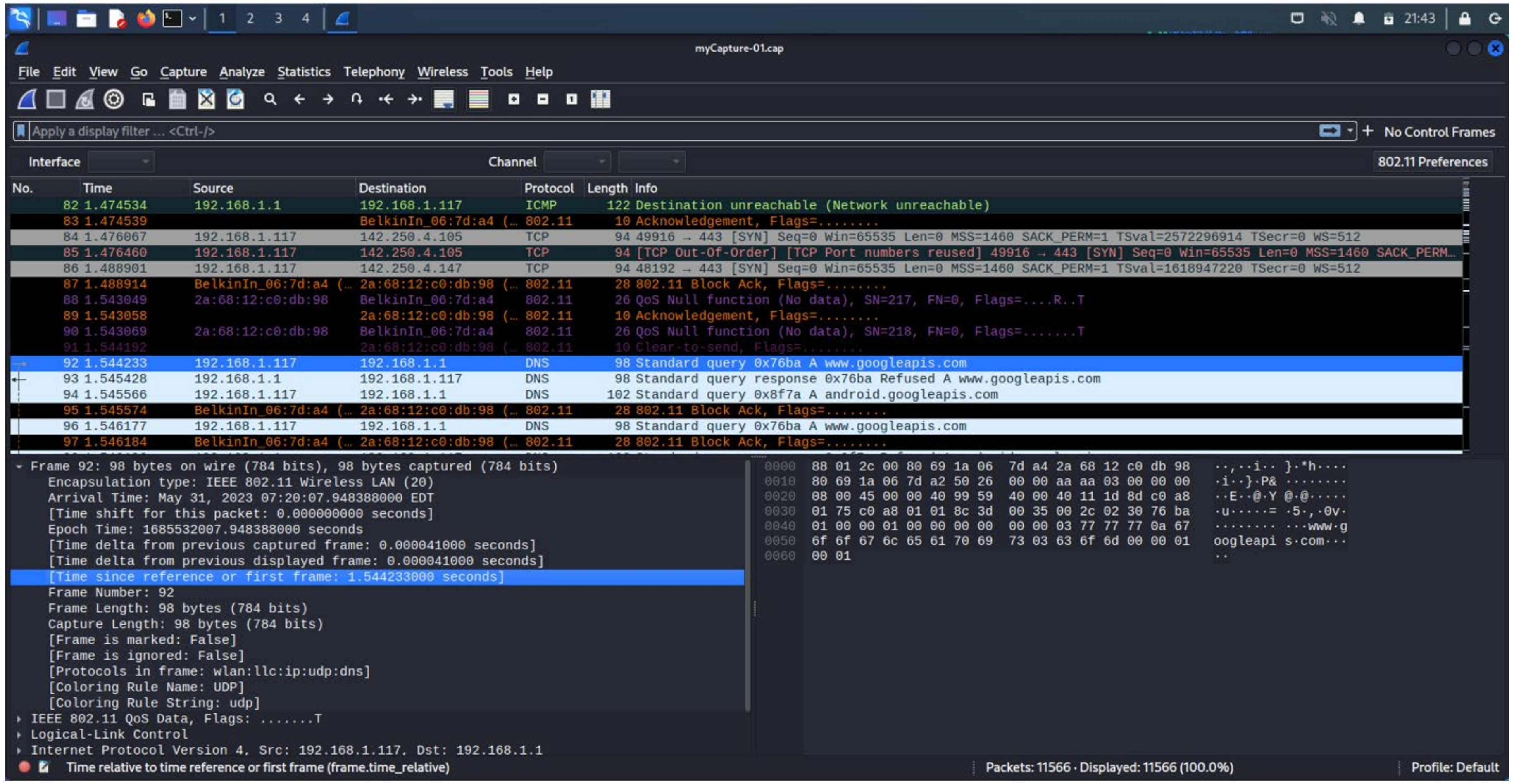
## Packet sniffing

- You can use a network packet analyzer to carry out packet sniffing on a target LAN
  - Well-known packet analyzers include Wireshark, Nmap and tcpdump
  - Use packet analyzers together with a network tap (for wired Ethernet LANs) or a wireless card (for wireless LANs, like WiFi)


A simple network tap

# Packet sniffing

- Example of a packet sniffing session using Wireshark:



myCapture-01.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
No Control Frames
Interface
Channel
802.11 Preferences

```
No.  Time       Source                  Destination             Protocol Length Info
  82 1.474534   192.168.1.1             192.168.1.117           ICMP     122 Destination unreachable (Network unreachable)
  83 1.474539                           BelkinIn_06:7d:a4 (…    802.11    10 Acknowledgement, Flags=........
  84 1.476067   192.168.1.117           142.250.4.105           TCP       94 49916 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1 TSval=2572296914 TSecr=0 WS=512
  85 1.476460   192.168.1.117           142.250.4.105           TCP       94 [TCP Out-Of-Order] [TCP Port numbers reused] 49916 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM…
  86 1.488901   192.168.1.117           142.250.4.147           TCP       94 48192 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1 TSval=1618947220 TSecr=0 WS=512
  87 1.488914   BelkinIn_06:7d:a4 (…    2a:68:12:c0:db:98 (…    802.11    28 802.11 Block Ack, Flags=........
  88 1.543049   2a:68:12:c0:db:98       BelkinIn_06:7d:a4       802.11    26 QoS Null function (No data), SN=217, FN=0, Flags=....R..T
  89 1.543058                           2a:68:12:c0:db:98 (…    802.11    10 Acknowledgement, Flags=........
  90 1.543069   2a:68:12:c0:db:98       BelkinIn_06:7d:a4       802.11    26 QoS Null function (No data), SN=218, FN=0, Flags=.......T
  91 1.544192                           2a:68:12:c0:db:98 (…    802.11    10 Clear-to-send, Flags=........
  92 1.544233   192.168.1.117           192.168.1.1             DNS       98 Standard query 0x76ba A www.googleapis.com
  93 1.545428   192.168.1.1             192.168.1.117           DNS       98 Standard query response 0x76ba Refused A www.googleapis.com
  94 1.545566   192.168.1.117           192.168.1.1             DNS      102 Standard query 0x8f7a A android.googleapis.com
  95 1.545574   BelkinIn_06:7d:a4 (…    2a:68:12:c0:db:98 (…    802.11    28 802.11 Block Ack, Flags=........
  96 1.546177   192.168.1.117           192.168.1.1             DNS       98 Standard query 0x76ba A www.googleapis.com
  97 1.546184   BelkinIn_06:7d:a4 (…    2a:68:12:c0:db:98 (…    802.11    28 802.11 Block Ack, Flags=........
```

```
▾ Frame 92: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
    Encapsulation type: IEEE 802.11 Wireless LAN (20)
    Arrival Time: May 31, 2023 07:20:07.948388000 EDT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1685532007.948388000 seconds
    [Time delta from previous captured frame: 0.000041000 seconds]
    [Time delta from previous displayed frame: 0.000041000 seconds]
    [Time since reference or first frame: 1.544233000 seconds]
    Frame Number: 92
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: wlan:llc:ip:udp:dns]
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]
▸ IEEE 802.11 QoS Data, Flags: .......T
▸ Logical-Link Control
▸ Internet Protocol Version 4, Src: 192.168.1.117, Dst: 192.168.1.1
```

```
0000  88 01 2c 00 80 69 1a 06  7d a4 2a 68 12 c0 db 98   ··,··i·· }·*h····
0010  80 69 1a 06 7d a2 50 26  00 00 aa aa 03 00 00 00   ·i··}·P& ········
0020  08 00 45 00 00 40 99 59  40 00 40 11 1d 8d c0 a8   ··E··@·Y @·@·····
0030  01 75 c0 a8 01 01 8c 3d  00 35 00 2c 02 30 76 ba   ·u·····= ·5·,·0v·
0040  01 00 00 01 00 00 00 00  00 00 03 77 77 77 0a 67   ········ ···www·g
0050  6f 6f 67 6c 65 61 70 69  73 03 63 6f 6d 00 00 01   oogleapi s·com···
0060  00 01                                              ··
```

```
Time relative to time reference or first frame (frame.time_relative)      Packets: 11566 · Displayed: 11566 (100.0%)      Profile: Default
```

## Packet sniffing: mitigation

- It is not possible to completely protect all data in a network from packet sniffing
- But we can mitigate its effects by encrypting the data in the traffic, where possible
  - We can use TLS (more on this protocol later) to encrypt, say, web browser traffic – this is known as HTTPS
  - Most data in the packets will be encrypted, particularly the payload
  - The headers of the packets **cannot** be encrypted

## The Address Resolution Protocol (ARP)

- The Address Resolution Protocol is a link layer protocol that provides services to the network layer
- It is used to find a computer's (or host's) hardware address in a LAN, given the computer's network layer address
    - In other words, ARP determines a computer's MAC address, given the computer's IP address
- Machines (or more accurately, network interfaces), are identified by their MAC address
    - It is a 48-bit address, e.g. 00:2A:B7:34:F2:5B
    - Every device connected to the LAN will have a unique MAC address

# The Address Resolution Protocol (ARP)

- How it works:
- Suppose a source computer wants to send a data unit to a destination computer on the LAN; the source machine knows the IP address of the destination machine
- However, the sending of the data unit is delegated to the link layer, so the source machine needs to identify the <u>MAC address</u> of the destination machine
- In the ARP protocol, the translation of IP addresses to MAC addresses is accomplished using a broadcast message (an *ARP request*) that queries all machines on a LAN, so that the proper machine (i.e. the destination computer) can respond

## The Address Resolution Protocol (ARP)

- For example:

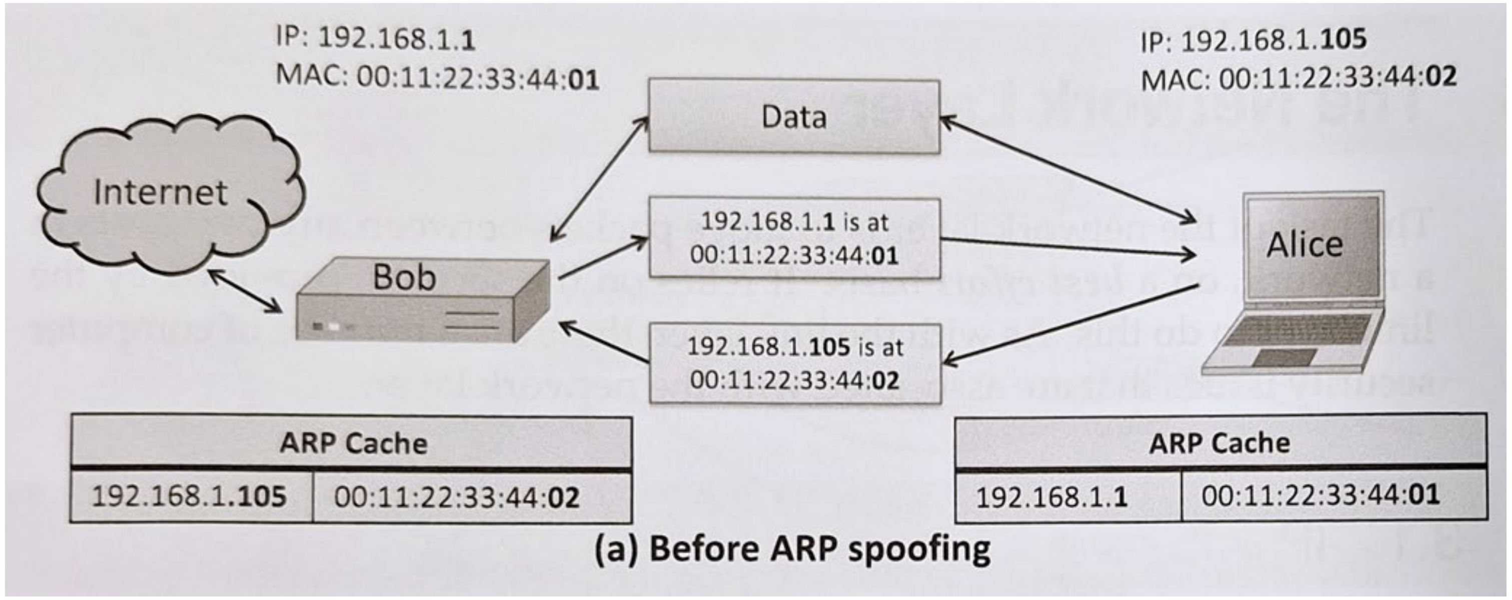
1. The source machine wishes to send some data to a destination machine whose IP address is 192.168.1.10
2. The source machine broadcasts an *ARP request*, in the form of “Who has IP address 192.168.1.10 ?”
    - This request is sent to all machines in the LAN (it is a broadcast)
3. The machine with IP address 192.168.1.10, if it exists, responds with an *ARP reply* in the form of: “192.168.1.10 is at 00:24:3A:FD:7E:B7”
    - This reply is sent in a frame addressed to only the machine that made the ARP request
4. When the source machine receives the ARP reply, the machine stores the IP-MAC address pair in a table located in its memory, called its *ARP cache*
5. After this ARP resolution, the source machine can finally send its data to the destination machine

## The Address Resolution Protocol (ARP)

- The ARP is a simple and effective protocol, but it lacks an authentication scheme
  - Any machine on the LAN can claim to have the requested IP address
  - Not only that, any machine that receives an ARP reply will automatically update its ARP cache with the new IP-MAC address pair (even if that reply was **not** preceded by an ARP request)

- Consequently, it is possible for malicious parties on a LAN to perform an *ARP spoofing* attack
  - This is a man-in-the-middle attack, which if successful, allows an attacker to eavesdrop on traffic in a LAN, and more

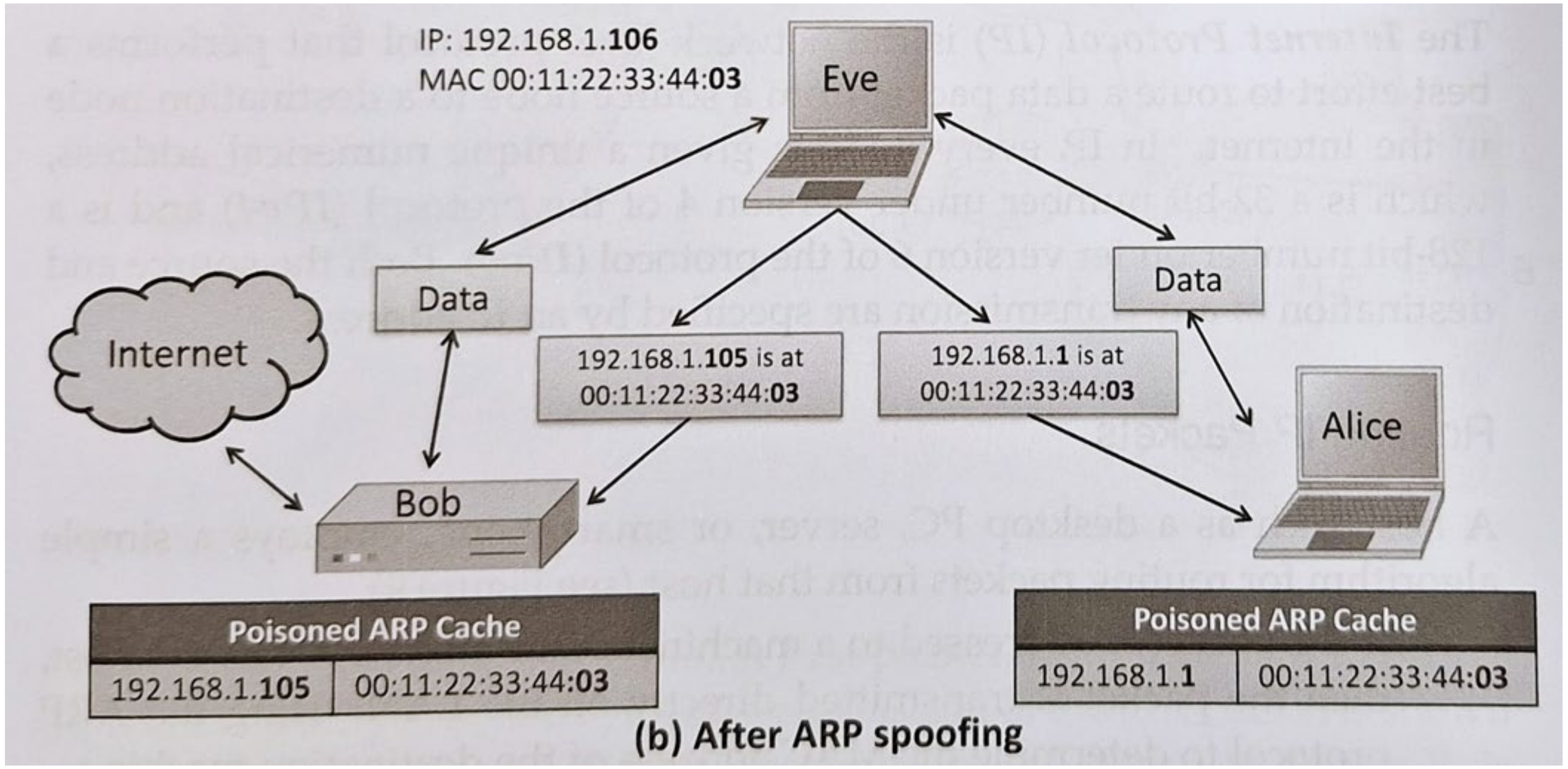
# ARP spoofing attack



(a) Before ARP spoofing

1. Prior to the attack, Alice and Bob have both established each other's respective IP-MAC address pairs in their respective ARP caches, after broadcasting ARP requests on their LAN

# ARP spoofing attack



(b) After ARP spoofing

2. Eve (who is also in the LAN) now sends an ARP reply to Alice, associating Bob's IP address with Eve's MAC address, and an ARP reply to Bob, associating Alice's IP address with Eve's MAC address

# ARP spoofing attack

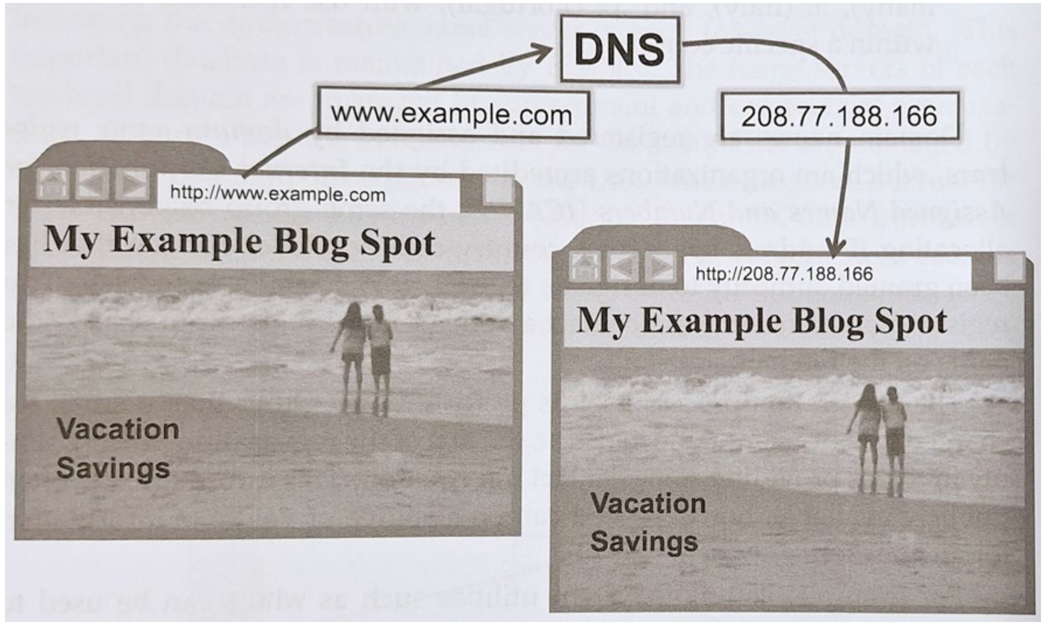
3. This results in the *ARP cache poisoning* of both Alice's and Bob's machines
    - Alice now thinks Bob's IP address is associated with Eve's MAC address
    - Bob now thinks Alice's IP address is associated with Eve's MAC address
4. As a result, all traffic between Alice and Bob are now routed through Eve
5. This allows Eve to control the traffic between Alice and Bob
    - Eve can passively observe this traffic
    - Or she can actively tamper with the traffic
    - Eve can also stop traffic between Alice and Bob (a denial-of-service attack)

## ARP spoofing attack: prevention

- One simple technique is to check for multiple occurrences of the same MAC address on the LAN, which is an indicator of an ARP spoofing attack
- Another solution is to use *static ARP tables*, which requires a network administrator to manually specify a device’s (typically a router) ARP cache to assign certain MAC addresses to specific IP addresses
    - ARP requests to that device are ignored

# The DNS protocol

- The DNS protocol allows internet users to utilize human-friendly computer hostnames in a web browser, instead of IP addresses
- Basic concept is as follows:
  - Suppose you want to access the website with the hostname “www.example.com”
  - Your web browser first contacts a DNS server to find the web server’s actual IP address (in this case, 208.77.188.166)
  - Your web browser can then attempt to reach that web server using the web server’s IP address 208.77.188.166



## The TLS protocol

- TLS: **T**ransport **L**ayer **S**ecurity
- First version defined in 1999, current version (TLS 1.3) defined in 2018
  - It is the current standard used for establishing a secure channel over the internet
  - TLS replaces the older and insecure SSL protocol
- TLS works at the application and transport layers
  - Note that TLS does not fit neatly into any single layer of the TCP/IP model
- TLS uses asymmetric (public-key) crypto, and the data transmitted using this protocol is confidential, has integrity and is authenticated
- This protocol is widely used in applications such as email and VoIP, and it is most commonly associated with the HTTPS protocol

## The TLS protocol

- TLS allows for a variety of methods that provide confidentiality, integrity and authenticity
- It also offers a selection of key establishment methods
- Note that TLS 1.3 **removed** a sizable number of block ciphers, hash functions and key exchange methods that were available in TLS 1.2
    - As a result, a significant number of websites are incompatible with TLS 1.3
    - Most websites are compatible with TLS 1.2

- TLS starts with a *handshake*, where the client and server exchange information about their respective cryptographic capabilities, decide on the cryptographic methods to use, as well as certificates

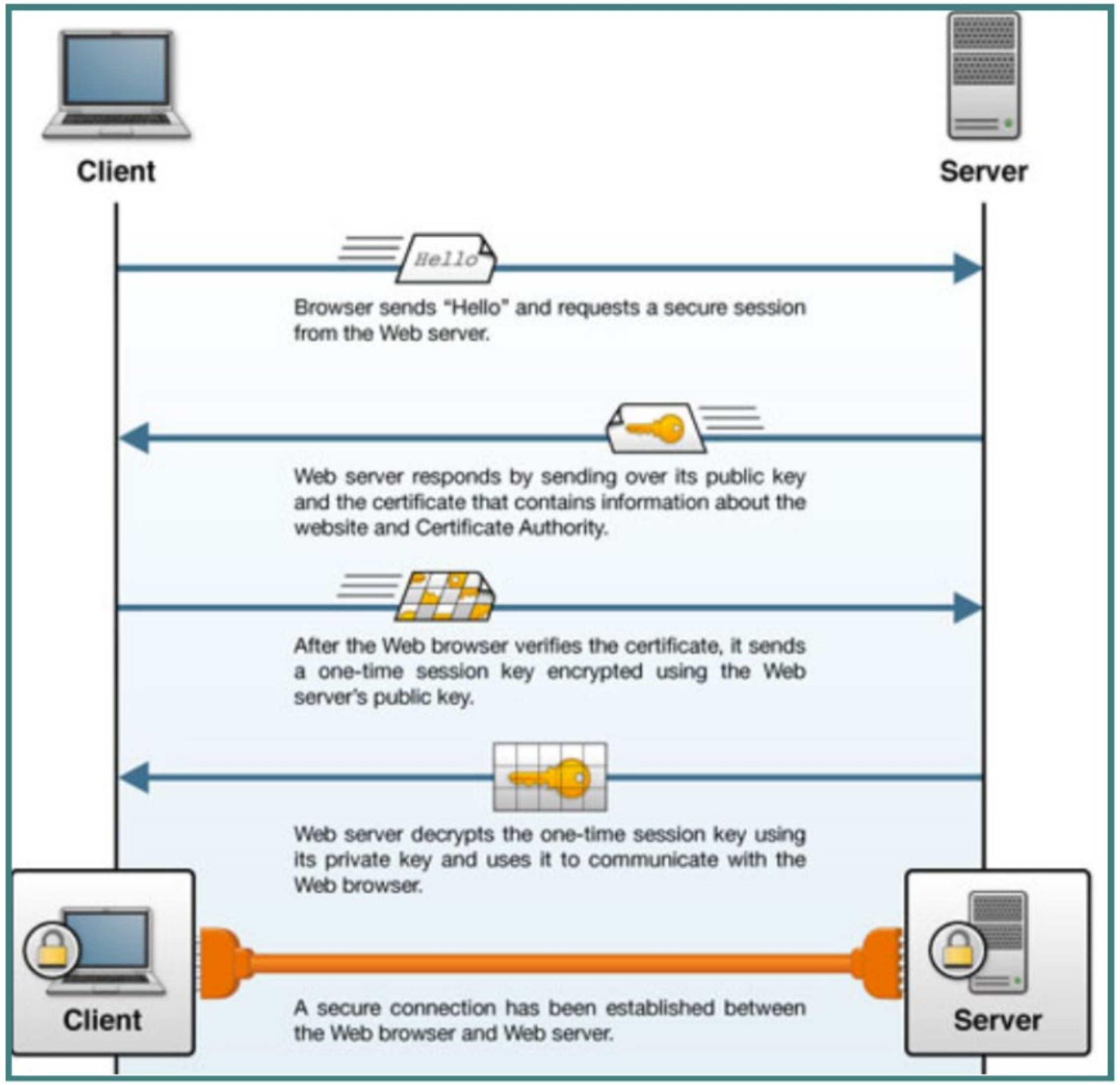
# The TLS protocol: authentication options

- Some of the key establishment methods supported by TLS 1.3 include:
- Ephemeral DHKE
    - Here, the client and server each dynamically create a respective one-time **public-private key pair** for DHKE; the key pairs are deleted after the session is over – this guarantees a truly fresh session key each time DHKE is executed
        - The reason is for *perfect forward secrecy*: if the private key of the server is ever leaked, **past** enciphered messages are still protected
        - Note: TLS 1.3 **removed** the option for static DHKE (this is the version taught in our lectures, where the client and server utilize static public-private key pairs)
- RSA digital signature protocol
    - Used for signing of the certificates
    - Note: TLS 1.3 **removed** RSA as an option as a key establishment method
- Elliptic curve DHKE
- Pre-shared key (PSK)

# The TLS protocol: encryption options

- Some of the ciphers supported by TLS 1.3 include:
- Block ciphers:
  - AES in CTR mode, with polynomial arithmetic in the extension field $GF(2^{128})$ to compute an authentication tag – this is also known as AES in Galois/Counter Mode (AES-GCM)
  - AES-GCM provides **both** confidentiality and integrity
  - Note: TLS 1.3 **removed** the options for a number of ciphers, including AES in CBC mode and 3DES in CBC mode
- Stream ciphers:
  - ChaCha20

# The TLS protocol: operating procedure


Client
Server
Hello
Browser sends "Hello" and requests a secure session from the Web server.
Web server responds by sending over its public key and the certificate that contains information about the website and Certificate Authority.
After the Web browser verifies the certificate, it sends a one-time session key encrypted using the Web server's public key.
Web server decrypts the one-time session key using its private key and uses it to communicate with the Web browser.
Client
Server
A secure connection has been established between the Web browser and Web server.

## The TLS protocol: operating procedure

1. The client (e.g. a web browser) sends a *client hello* message to the server (e.g. a web server) to request a secure connection
2. The server responds with a *server hello* message, along with a digital certificate containing its public key (and CA information)
3. The client verifies the server's certificate by using the appropriate CA's public key. If the verification is successful, the client proceeds to the next step
4. The client creates a secret session key, then encrypts it with the server's public key and sends the resulting ciphertext message over to the server

## The TLS protocol: operating procedure

5. The server decrypts the enciphered session key using its private key and sends an acknowledgment message to the client
6. A secure connection is now established; the client and the server can exchange enciphered messages using the shared session key

- Note: the TLS protocol also allows for the verification of the client’s certificate, but this is optional

## The TLS protocol: comments

- Each application needs to set up its own TLS session
- It’s up to the application to determine what data to send through the TLS connection
- E.g. in the HTTPS protocol, the web browser sends the HTTP traffic through a TLS connection
- TLS is a convenient and relatively fool-proof way to improve the security of data sent over the internet