TCP/IP

- **Model and Layers**
- **Bits and Number Bases**
- **IPv4 Addressing**
- **Subnetting**
- **Classless Interdomain Routing**
- **IPv6**

Model and Layers

- At the beginning of the course, we discussed two primary conceptual models of networking: **OSI** and **TCP/IP**.

- The OSI model consists of seven layers:
  - **7:** Application
  - **6:** Presentation
  - **5:** Session
  - **4:** Transport
  - **3:** Network
  - **2:** Data Link
  - **1:** Physical

Model and Layers

- The OSI model has a protocol stack, but we use it primarily as a conceptual reference.
- In contrast, the ***TCP/IP protocol stack*** has been much more widely adopted.
- It is divided into four layers -- **`Application`**, **`Transport`**, **`Internet`**, and **`Network Interface`** (a.k.a. **`Link`**) -- that more or less "map onto" the OSI layers.

<u>Model and Layers</u>

- TCP/IP Layers:
  - **4)** **`Application`**: Deals with the applications that process network requests, along with their associated ports.
    - A **`port`** is an address to which you send data to be received...
      - by a particular application...
      - for processing.
    - You might think of it as a *<u>transport-layer address</u>*.
    - Transport protocols like *<u>TCP</u>* and *<u>UDP</u>* use **65,536** different ports, which can be grouped into three categories...

Model and Layers

- ***Port types:***
  - **"Well-known":** 1-1023
  - **Registered:** 1024-49151
  - **Private:** 49152-65535
- The **well-known ports** are used by some of the more common networking applications, such as…
  - **Port 22:** Secure Shell (SSH)
  - **Port 25:** Simple Mail Transfer Protocol (SMTP)
  - **Port 80:** Hypertext Transfer Protocol (HTTP)
  - **Port 443:** Secure HTTP
- See ***Table 6-4*** for more such examples.

Model and Layers

- **3) Transport**: Responsible for type of connection between hosts and acknowledgments of data sent/received.
  - The two main **transport-layer protocols** are TCP and UDP, which are connection-oriented and connectionless, respectively
  - **Transport Control Protocol (TCP)** is **connection-oriented**, where it initiates/confirms a connection, manages transfer, and closes said connection.
  - This begins with a 3-packet sequence (each is a type of packet):
    - **SYN**: *Synchronizing*
      - From Host A to B, attempting connection
      - Sequence number (x) for tracking packets (SEQ#)
      - Length of zero because it contains no data (LEN)

Model and Layers

- **SYN ACK**: *Synchronizing Acknowledgement*
  - From Host B to A, acknowledging package from A
  - Sequence number (y)
  - Acknowledgement number (x+1), also called ACK#
- **ACK**: *Acknowledgement*
  - From Host A to B
  - Sequence number (x+1)
  - Acknowledgement number (y+1)

- This is called a "*handshake*", after which point data packets are transferred.

Model and Layers

- Connection is *terminated* via a 4-packet sequence, where each host:
  - Sends a **`FIN`** packet…
  - …and receives an **`ACK`** packet.
- The **`User Datagram Protocol (UDP)`** is connectionless:
  - A packet is sent from a source to a destination.
  - There is *no acknowledgement* from the other side.
  - Transfer continues until the source stops sending or the destination stops accepting.

Model and Layers

- 2) **Internet**: The addressing and routing of data packets
  - *Internet Protocol* (**IP**): Defines addressing scheme for sources and destinations of data packets sent within or between networks
  - *Address Resolution Protocol* (**ARP**): Associating IP addresses with MAC addresses
  - *Internet Control Message Protocol* (**ICMP**): Data flow control and diagnostics.
- 1) **Network Interface**, or *Link*
  - LAN segments
  - WAN connections

Bits and Number Bases

- Numbers are expressed in **bases**, where…
  - The base is the number of possible values a digit can have.
  - The range of values for a digit will be zero through the base minus one.
- ***Examples:***
  - **Decimal:** **0 - 9**
  - **Binary:** **0 - 1**

Bits and Number Bases

- ***Conversion:*** You calculate the value of the number by multiplying each digit by exponents of the base.
  - Generally, you start where the *right-most* digit
  - ***Binary-to-Decimal:*** 10011

Digit	1	0	0	1	1
Exponent	* 2^4	* 2^3	* 2^2	* 2^1	* 2^0
Product	16	0	0	2	1
SUM	16	16	16	18	19

Bits and Number Bases

- ***Decimal-to-Binary:*** 719
  - Divide the number by two
  - Place the remainder on the end
  - Repeat with the quotient, placing the remainder before the previous digit.
  - Do this until you get a quotient of ***zero***.

Value	Quotient	Remainder
719	359	1
359	179	1
179	89	1
89	44	1
44	22	0
22	11	0
11	5	1
5	2	1
2	1	0
1	0	1

1 0 1 1 0 0 1 1 1 1

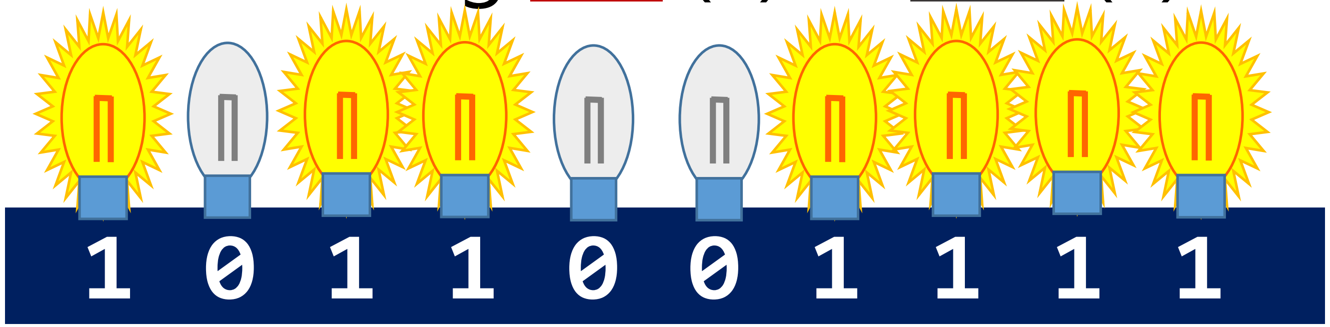
Bits and Number Bases

- ***Hexadecimal:***
  - Values are `0` through `15`
  - Digits are `0 – 9`, with `10 – 15` represented by `A` through `F`
  - A hex digit is equivalent to a *quartet* (`4 bits`)
  - ***Example:*** `719` → 10 1100 1111 → 2 c f
  - This way, you can easily convert *back and forth* between the two

Hex	Binary	Hex	Binary
0	0000	4	0100
1	0001	5	0101
2	0010	6	0110
3	0011	...	

Bits and Number Bases

- A number expressed in binary digits is a *bit string*, and you can think of them as being **ON** (1) or **OFF** (0)
- ***For example:***



- *Selecting bits*:
  - Sometimes, you will want to "turn" some bits on or off
  - This will be the case in scenarios where individual bits or bit sequences in the string have meaning, *in their own right*.

Bits and Number Bases

- This can be accomplished by using a **bit mask**, along with **bitwise operations**.
  - A *bit mask* is simply a bit string, where the different bits or bit sequences have special meaning
  - A *bitwise operation* acts upon a bit pair to produce `0` or `1`, and we will look at two of them:
    - **OR** is used to turn bits **on**
    - **AND** is used to turn bits **off**

Bits and Number Bases

- OR operation:
  - Any bit ***or*** 1 is turned/left **ON**
  - In contrast, any bit ***or*** 0 is simply left **unchanged**

Bit		Mask	Result	
1	OR	1	1	Turned ON (if zero, would have been)
0	OR	1	1	Turned ON
1	OR	0	1	Unchanged
0	OR	0	0	Unchanged

  - If you use a bit mask with OR , it will turn some bits on while keeping the others as they were.

Bits and Number Bases

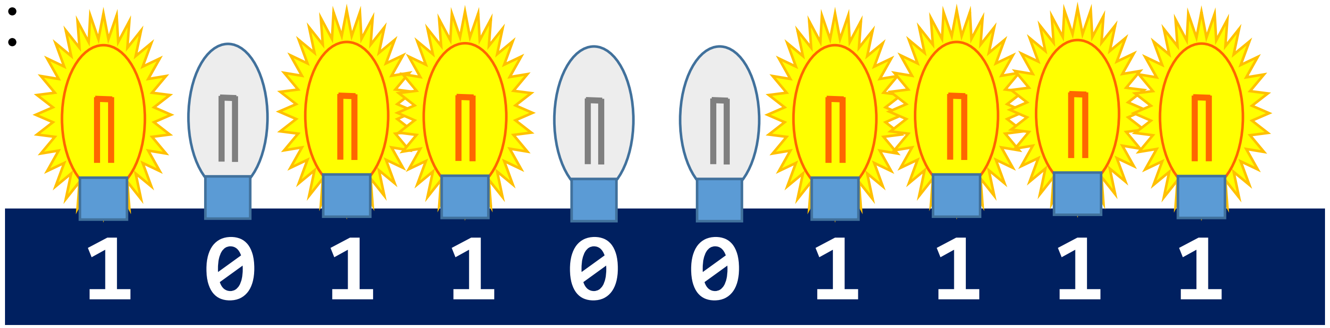
- AND operation:
  - Any bit ***and*** 0 is turned/left **OFF**
  - In contrast, any bit ***and*** 1 is simply left **unchanged**

Bit		Mask	Result	
1	AND	1	1	Unchanged
0	AND	1	0	Unchanged
1	AND	0	0	Turned OFF
0	AND	0	0	Turned OFF (if zero, would have been)

  - If you use a bit mask with AND , it will turn some bits off while keeping the others as they were.

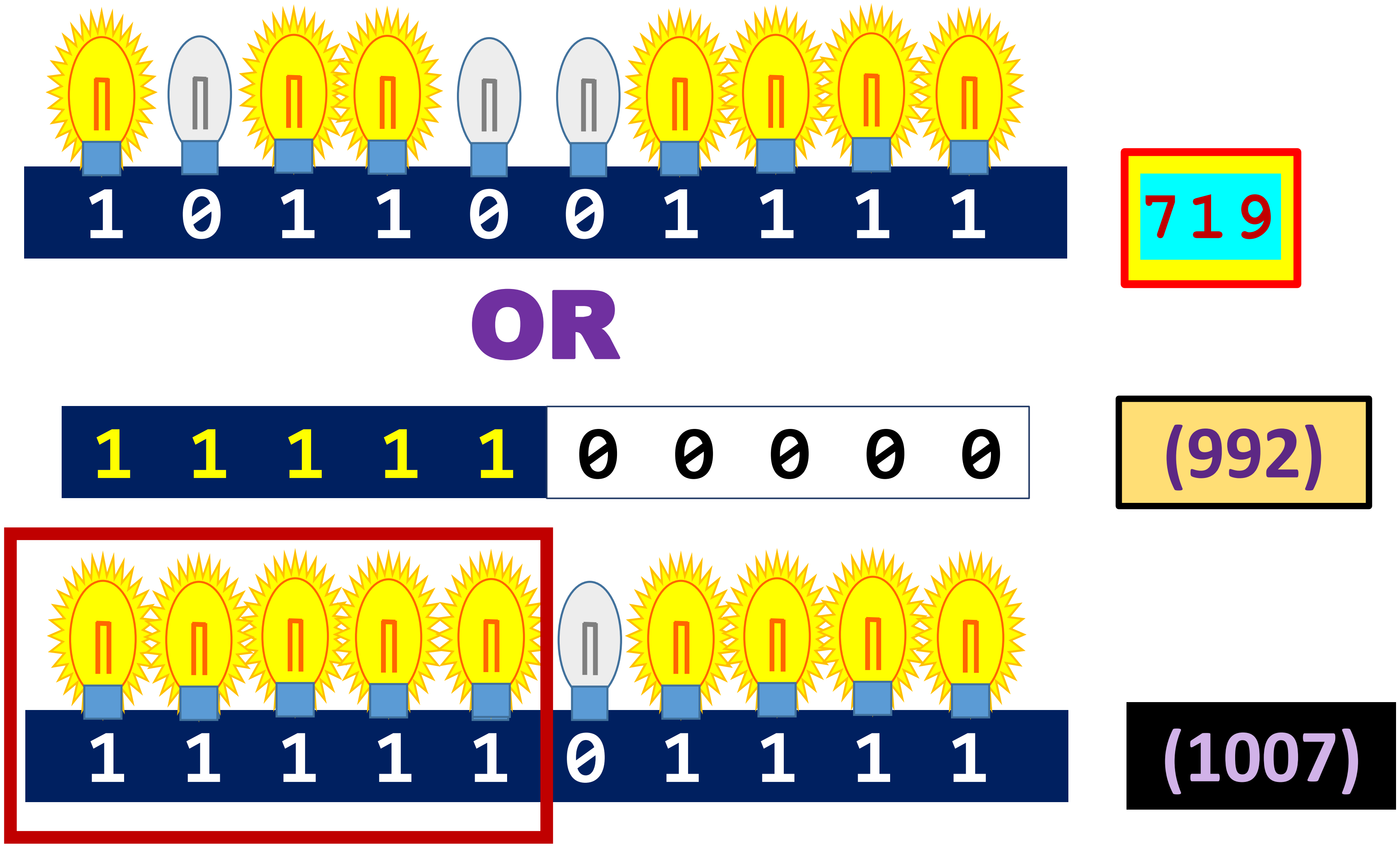
Bits and Number Bases

- Let's look at an example:
  - Our original bit string:



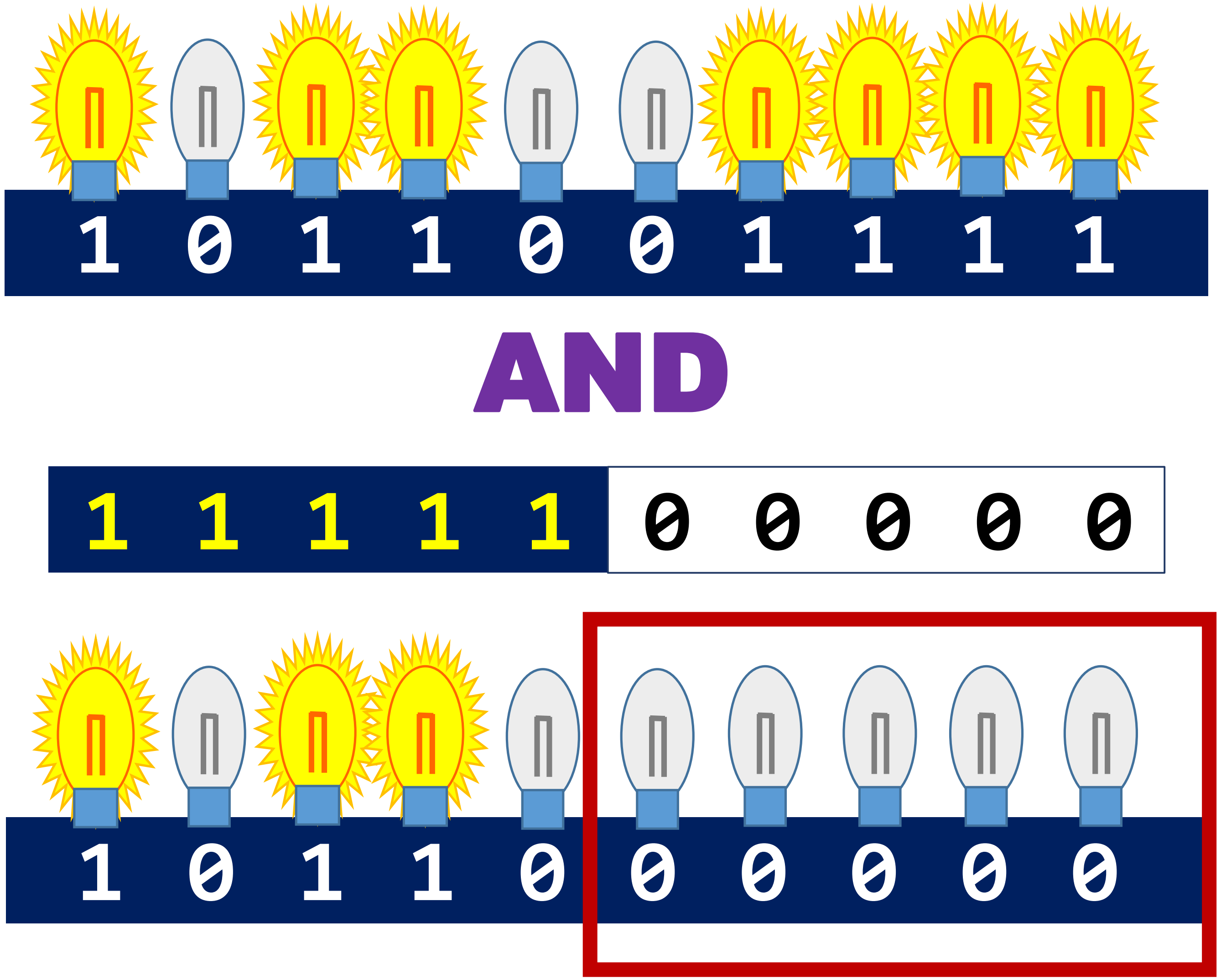
  - Bit string's decimal value: 719
  - A bit mask: 1 1 1 1 1 0 0 0 0 0 (992)

Mask applied with OR



First 5 bits are turned **ON**

Mask applied with AND



719

(992)

(704)

Last 5 bits are turned OFF

IPv4 Addressing

- IP addressing allows hosts and other devices to have routable addresses, both in LANs and within wider networks -- most notably the Internet.
  - MAC addresses provide for forwarding within a LAN, but IP addresses let us extend beyond that.
  - The predominant version of IP today is **IPv4**, though we will cover IPv6 later on.
- IPv4 addresses are divided into five *classes*, indicated by letters **A-E**

IPv4 Addressing

```
Class A:   0.0.0.0 - 127.255.255.255
Class B: 128.0.0.0 - 191.255.255.255
Class C: 192.0.0.0 - 223.255.255.255
Class D: 224.0.0.0 - 239.255.255.255
Class E: 240.0.0.0 - 254.255.255.255
```

- We will primarily be using the first three classes, though Class D is relevant to *Chapter 9, "Routing Protocols"*.
- These classes are useful for *demonstrative* purposes, though the classification system is now outdated…

IPv4 Addressing

- An IPv4 address is expressed in **32 bits**:
  - In theory, this allows for 2^{32}, or **4294967296**, possible IPs
  - Each **octet** (8-bit chunk) will have a value in the range **0-255**
  - Normally, you will see IP addresses expressed in *decimal* form, where the octets' values are separated by *periods*.
  - ***Example:*** **www.google.com**
    - Decimal: **146.115.22.166**
    - Binary: **10010010011100110001011010100110**

IPv4 Addressing

- Generally, the bits of an IP address are divided into two parts that, in combination, give *the full network location of a particular host*.
  - The **network bits** comprise the first part of the longer bit string, and they convey the location of the network where the host resides.
  - Following are the **host bits**, which indicate the location of the host within the network.
  - Traditionally, *each octet* in an IPv4 address contains *either network bits or host bits*, according to address class…

IPv4 Addressing

Address Class	Network Octets (Bits)	Host Octets (Bits)	# Hosts per Network
A	1 (8)	3 (24)	1.68e7 (2^{24})
B	2 (16)	2 (16)	6.55e4 (2^{16})
C	3 (24)	1 (8)	2.56e2 (2^{8})

See also ***Figure 6-13***

- Depending on the number of host bits (vs network bits), different classes of networks will have a different possible number of hosts per network -- specifically, ***two raised to the power of number of host bits***.

IPv4 Addressing

- Within each class, some IPs are designated as **private**:

```
Class A:     10.0.0.0 - 10.255.255.255
Class B:   172.16.0.0 - 172.31.255.255
Class C: 192.168.0.0 - 192.168.255.255
```

  - These are for internal networks, or **intranets**, such as…
    - The IT Lab's inner network
    - A home network
  - ***Private IP addresses are <u>not</u> routable over the Internet!***

IPv4 Addressing

- On a wider level, the **Internet Assigned Number Authority (IANA)** is responsible for the allocation of IP addresses.
  - However, it delegates this task to **regional Internet registries (RIRs)**, who allocate addresses according to geographical location.
  - In North America, the **American Registry for Internet Numbers (ARIN)** assigns IP addresses.
  - Large entities like ISPs and universities are allocated blocks of IP addresses to further assign as they choose.

Subnetting

- A network can be partitioned into smaller entities called **subnets**.
  - These create a *hierarchical* network structure.
  - Subnets are separated *at layer 3*, in the sense that you use *IP and routing* to move between them
- Example: You have a network at address **192.145.17.0**

```
+-----------+-----------+-----------+-----------+
|    192    |    145    |    17     |     0     |
+-----------+-----------+-----------+-----------+
| 11000000  | 10010001  | 00010001  | 00000000  |
+-----------+-----------+-----------+-----------+
```

Subnetting

- This network, however, might be divided into 4 subnets:

```
192.145.17.0    (IP range: 192.145.17.0   - 192.145.17.63)
192.145.17.64   (IP range: 192.145.17.64  - 192.145.17.127)
192.145.17.128  (IP range: 192.145.17.128 - 192.145.17.191)
192.145.17.192  (IP range: 192.145.17.192 - 192.145.17.255)
```

  - Each such subnet is *logically independent* of the others.
  - Traffic from one subnet to another would pass through a *router*.

Subnetting

- Subnets within a network are established by declaring a space of **`subnet bits`**:
  - These subnet bits are ***borrowed*** from the host bits
  - Together with the network bits, they establish the identity of *the network and subnet*
  - Those bits also become the basis of a **`subnet mask`**
- The material that follows will be especially pertinent to ***Lab 8*** and ***Homework 8***.

Subnetting

- Consider the subnets -- in particular, their *fourth-octet* binary values:

```
  0:  00000000
 64:  01000000
128:  10000000
192:  11000000
```

- Notice that the *only* bits that vary are the first two.
- This is because two bits were ***borrowed*** -- from the host bits -- to serve as *subnet bits*.

Subnetting

- This is where the *math* starts to come in...
  - Let's identify two variables
    - **x** (# of bits borrowed from host portion)
    - **y** (# of total host bits, by address class)
  - Based on this, we can calculate two possible values:
    - Number of subnets: 2^x
    - Number of possible IPs per subnet: 2^{y-x}
- For 2 subnet *bits*, we get *4 subnets*, with *64 IPs each*...

Subnetting

- For each subnet – such as **192.145.17.64** – two of the possible IPs are reserved for special uses:
  - The **subnet address**: (**192.145.17.64**)
    - All host bits are *zeroes* (**64: 01000000**)
    - The IP identity of the subnet itself
  - The **broadcast address**: (**192.145.17.127**)
    - All host bits are *ones* (**64: 01111111**)
    - Data sent to this address is broadcast to all hosts within the subnet
- Thus, # of possible hosts per subnet equals $2^{y-x} - 2$

Subnetting

- To distinguish the *net and subnet portion* of an IP address from the *host portion*, you will ***apply a subnet mask***
- A subnet mask is a **32-bit (four-octet)** value that resembles an IP address when expressed in decimal form.
    - The first **N** bits are all set to a value of one, where **N** is equal to the number of *network and subnet bits*.
    - You apply a subnet mask to a network address by **AND**-ing the two (see previous slides about *bit masking*).

Subnetting

- In the example above, where we had the network **`192.145.17.0`**…
  - It is a *Class C* address, so there are ***24 network bits***
  - In addition, we borrow ***two*** of the host bits so that we can have four subnets
  - Thus, in our subnet mask, ***the first 26 bits*** are set to ***one***
    - **`Binary:`** `11111111.11111111.11111111.11000000`
    - **`Decimal:`** `255.255.255.192`
  - If **`192.145.17.0`** the network was not subnetted, at all, then we would have a mask of **`255.255.255.0`** (*first 24 bits*)

Subnetting

- We will look at another example:
  - What we know:
    - **IP address:** 172.27.213.94
    - **Subnet mask:** 255.255.240.0
  - So, *what is the **subnet** address?*
  - To start with, let's put our IP address and subnet mask into *binary form*:
    - **Addr:** 10101100.00011011.11010101.1100001
    - **Mask:** 11111111.11111111.11110000.0000000

Subnetting

- If we AND the bits…

10101100.00011011.11010101.1100001

11111111.11111111.11110000.0000000

…then we get this result:

10101100.00011011.11010000.0000000

- So, the *subnet* is **172.27.208.0**

<u>**Subnetting**</u>

- The lab will ask you to do such things as:
  - *Calculating the subnet* of an IP address, by applying a subnet mask
  - *Determining a subnet mask*, based upon IP address class and the number of subnets to be established
  - Given a particular subnet mask…
    - How many <u>*subnets*</u>?
    - What are the <u>*subnet address*</u> and <u>*broadcast address*</u> for **each** subnet?
    - <u>*How many possible hosts*</u> per subnet?
- This, of course, leads us into the topic of CIDR…

Classless Interdomain Routing

- So far, we have been looking at **classful addressing**, in which a network is simply defined by the first one, two, or three octets -- depending on the address class.
  - That way, the network would have a range of possible IPs, according to the number of host bits.
  - For example, a Class A network has 24 host bits, allowing for $\mathbf{2^{24}}$ possible addresses within it
- The problem? Lots of unused IPs!

Classless Interdomain Routing

- What if a Class A network did not need all $\mathbf{2^{24}}$ possible addresses?
- This is part of why classful addressing is now obsolete.
- In its place, we now have the practice of **supernetting**, which lets us combine *smaller* networks (or subnets) into *larger* networks.
- For this, we use **classless interdomain routing (CIDR)** notation to express the subnet mask in a much shorter form: A backslash, followed by the number of bits.

Classless Interdomain Routing

- CIDR notation example:
  - A subnet mask of `255.255.255.0` would be expressed as `/24`
  - If you have a *subnet* at address **`172.27.208.0`** with a netmask of **`255.255.240.0`**
  - ...the CIDR notation would be **`172.27.208.0/20`**
- Just as you might partition a network into subnets by borrowing host bits – networks can be combined by *borrowing **network** bits*.

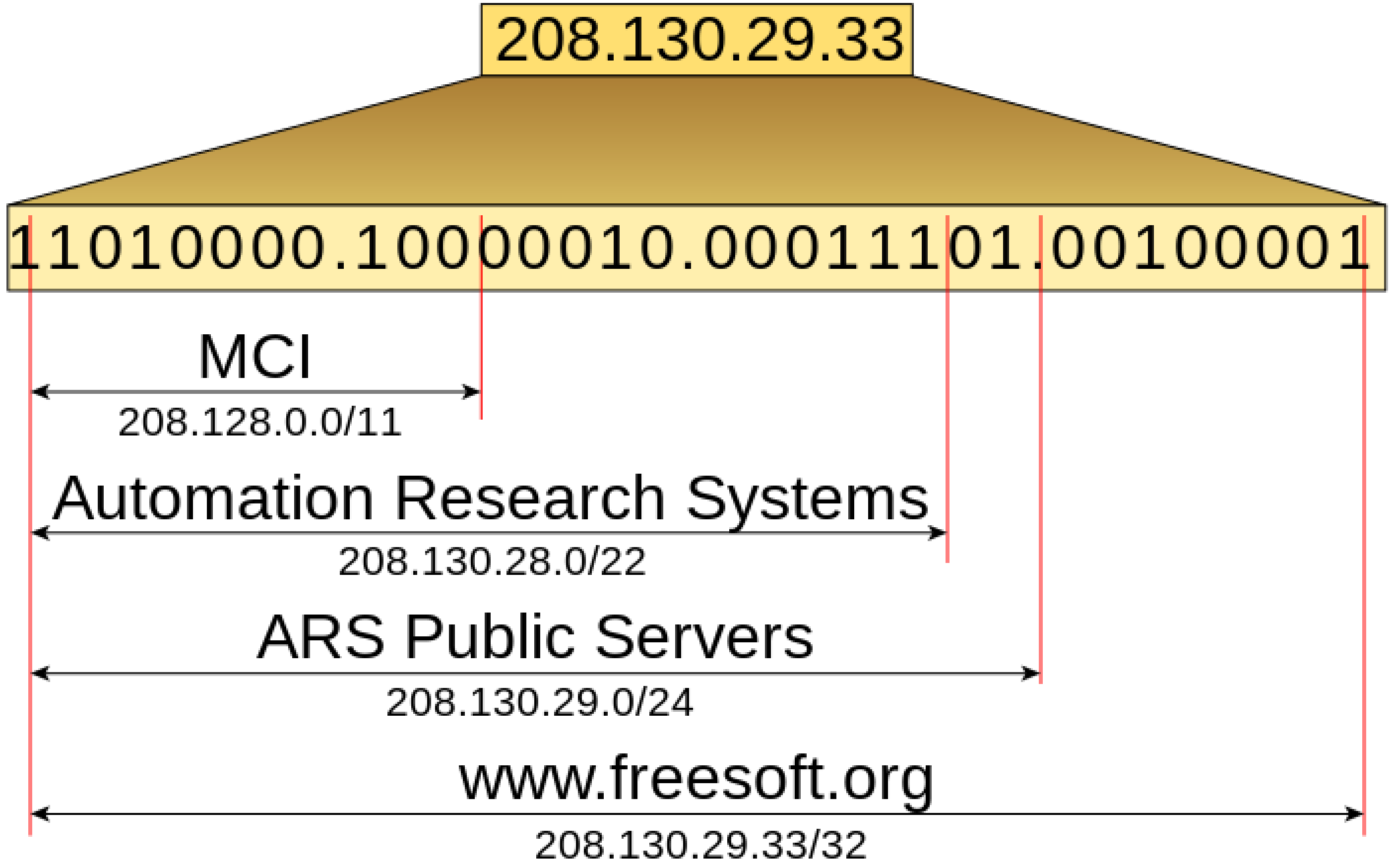
Classless Interdomain Routing

- Example: Networks 172.21.0.0, 172.22.0.0, and 172.23.0.0
  - Networks 172.21.0.0, 172.22.0.0, and 172.23.0.0, being Class B, have a subnet mask of 255.255.0.0 or /16
  - However, those IPs share the *first 14 bits* 10101100.000101xx.xxxxxxxx.xxxxxxxx
  - Therefore, those could be treated as part of a larger network -- a **supernet** -- of 172.20.0.0/14

Classless Interdomain Routing

- Multiple classful networks, grouped together as a supernet, are also called a **`CIDR block`**.
- While classful IP addressing would limit networks to certain sizes by allocating address space *1 octet (8 bits) at a time*, CIDR allows for a much more *flexible* allocation of IP ranges.
- This way, you do not have to allocate IP addresses to a network beyond its needs.

Classless Interdomain Routing



Source: https://upload.wikimedia.org/wikipedia/commons/2/26/CIDR_Address.svg

Classless Interdomain Routing

- When grouping subnets into a CIDR block, they *must resolve to the same IP* when the subnet mask is applied to them. For example…

  172.20.0.0/14

  172.21.0.0/14

  172.22.0.0/14

  172.23.0.0/14

- …would not be a problem because the **/14** mask resolves them to the same value of **172.20.0.0**

Classless Interdomain Routing

- However, these *would not work* as a CIDR block....

  172.22.0.0/14

  172.23.0.0/14

  172.24.0.0/14

  172.25.0.0/14

- ...would be problematic because the mask resolves *some* to **172.20.0.0** and *others* to **172.24.0.0**

IPv6

- IPv4 addressing, using *32 bits*, allows for roughly *4.3 billion* unique IP addresses.
- As the Internet grows and more devices are connected to it, this number is becoming insufficient.
- This is where **IPv6** (also known as **IPng**) comes in.
- IPv6 addressing uses ***128 bits*** -- which allows for 2^{128} possible addresses

IPv6

- Whereas IPv4 addresses are usually written in *dotted decimal* form (e.g., **`192.168.0.1`**), IPv6 addresses are expressed in *hexadecimal digits* -- separated by *colons*.
  - Example: **`9b32:e6da:d14f:6698:a9e5:7fae:1ba2:ed81`**
  - The example would be considered a **`full IPv6 address`** because none of the hex digits are zero.
- When some of the digits are zeroes, there may be ways to *"compress"* the zeroes to shorten the address.

IPv6

- **Zero compression**: Replace *consecutive zeroes* with *two colons*
  - **From:** `9b32:0000:0000:0000:a9e5:7fae:1ba2:ed81`
  - **To:** `9b32::a9e5:7fae:1ba2:ed81`
- **Leading zero compression**: For individual quartets, *omit leading zeroes*
  - **From:** `9b32:000a:d14f:0698:00e5:7fae:0002:ed81`
  - **To:** `9b32:a:d14f:698:e5:7fae:2:ed81`

IPv6

- *Both* compression types:
  - `From:  9b32:0000:0000:0000:00e5:7fae:0002:ed81`
  - `To:    9b32::e5:7fae:2:ed81`
- To *recover* the original IPv6 address from its compressed form…
  - Start with the *rightmost digit* (of the latter)
  - Place each into their appropriate slots, from right to left
  - Fill in *zeroes* as needed.

IPv6

- To *convert* an IPv4 address (`172.27.213.94`) to IPv6:
  - Convert the 32-bit address to 2 quartets of hexadecimal digits

    `ac1b` `d55e`

  - Separate the quartets by a colon

    `ac1b:d55e`

  - Place two colons at the start, to indicate the leading zeroes:

    `::ac1b:d55e`

IPv6

- IPv6 addresses belong to three categories:
  - **Unicast**: Associated with a single network interface controller on a networked device.
  - **Multicast**: Indicates a group of devices, and data sent to such an address will be sent to the entire group.
  - **Anycast**: Comes from a list of addresses.
- Although IPv6 allows for a much better range of addresses, IPv4 is near-universal and will be in play *for a long time to come*.

IPv6

- There are a number of technologies out there to facilitate the transition to IPv6.
- One such technology is the **`6to4 prefix`**, which allows IPv6 devices to use IPv4 networks.
    - This involves the use of special *6to4 devices* that do the routing required.
    - A *32-bit IPv4 address* will be included within the larger 128-bit IPv6 address

IPv6

- Until IPv6 becomes more common, there are other solutions out there for the issue of limited IPv4 addresses.
  - For example, a private IP address -- not being routable over the Internet -- can be used by many different hosts -- so long as it is unique ***within*** a private network.
    - In the IT Lab: **`10.0.0.0/24`** addresses
    - On home networks: **`192.168.0.0`**
  - When hosts on private networks need Internet connectivity, they may use **`Network Address Translation (NAT)`** -- where the router replaces the *inner, private* source IP with its own *outer, public* one.