Foundations

COS 460 & COS 540

1

Foundations

- Applications
- Requirements
- Architecture
- Performance
- Writing Code

2

Applications

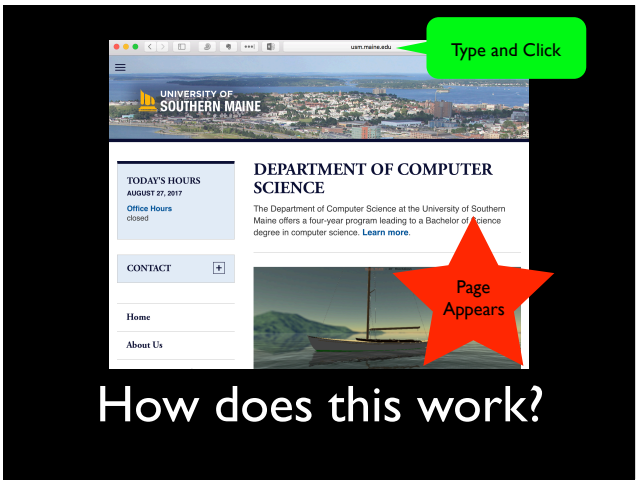
...and these are just the web "social media" ones

3

Example: HTTP

- HyperText Transfer Protocol
- HTTPS is "secure" HTTP
- Request for Comment (RFC) 2616, 2068, and 1954
- Utilizes TCP/IP (Internet Architecture)

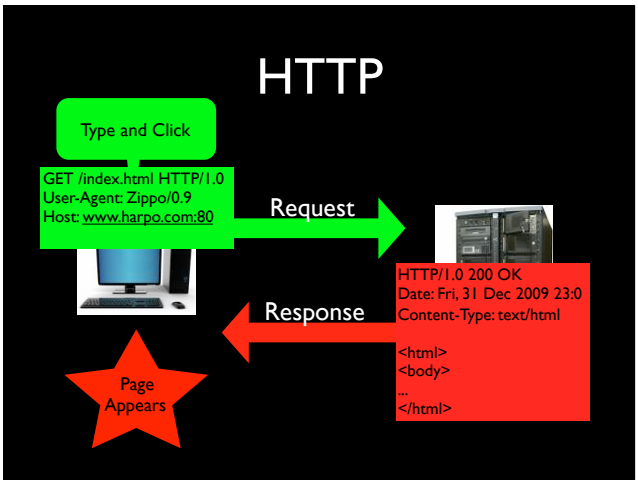
4



How does this work?

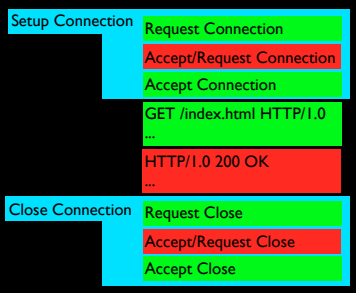
5

HTTP



6

HTTP, TCP



7

HTTP

- HTTP is essentially **file transfer**
- Every bit is important
- Similar to Email, Printing, File Servers...

8

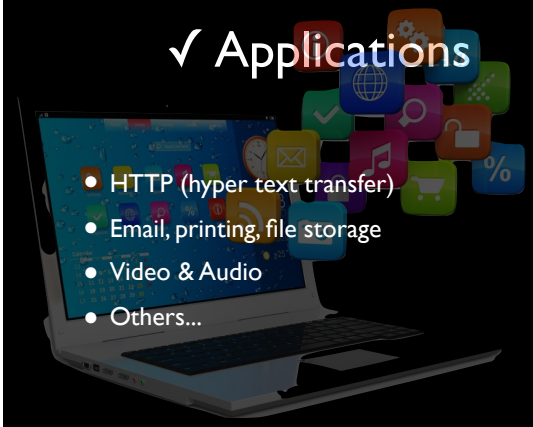
Audio & Video

Every bit is not **important** but their **timing** is

9

10

✓ Applications



- HTTP (hyper text transfer)
- Email, printing, file storage
- Video & Audio
- Others...

11

Foundations

- ✓ Applications
- Requirements
- Architecture
- Performance
- Writing Code

12

Requirements

- Connectivity
- Cost-effective
- Supports (common) Services

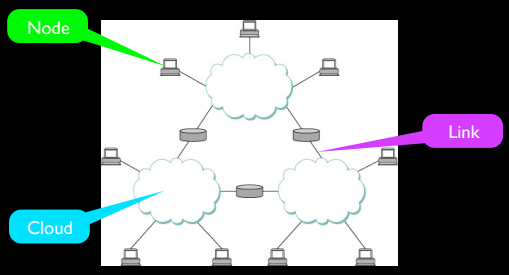
13

Who's Requirements?

- Application Programmer
- Network Provider (ISP)
- Network Designer

14

Connectivity



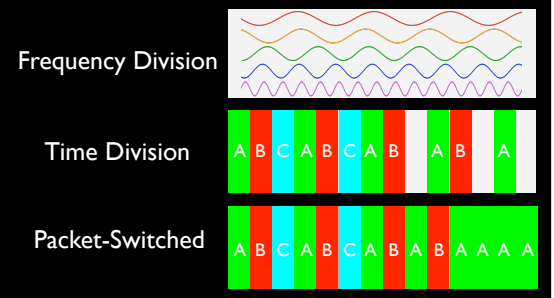
15

Cost-Effective

How do we get as **much information** as possible, from **multiple** sources to multiple sources across the "wire"?

...one bit at a time is slow.

Multiplexed



16

Imagine A has 2x B's data needs and B has 2x C's
FDM – Unused allocation for D and E as well as B and C not using all
TDM – Imagine A has more data, unused time slots
PSwitched – Quality of Service & Congestion problems

Common Services

Define **useful channels** that understands the **application needs** and the **networks ability**.

17

Common Services

- File transfer vs. Streaming
- Reliable vs. Unreliable
- Bandwidth vs. Latency
- Large vs. Small

18

19

✓ Requirements

- **Connectivity**
  Links, Nodes, Clouds
- **Cost-Effective**
  Packet Switched
- **Support (common) Services**
  File transfer, streaming, bandwidth...

20

Foundations

- ✓ Applications
- ✓ Requirements
- Architecture
- Performance
- Writing Code

21

Code

It Takes Two to Tango

- Server (passive)
- Client (active)

22

Server

- **Binds** to a "port"
  - Passively **listens** for connection request
    - **Accepts** connections
      - reads and writes data
        - **Closes** connection when done

23

Client

- Actively **connects** to server's port
  - reads and writes on connection
    - **Closes** connection when done

24

Java Server

```
// passive bind and listen
ServerSocket srv = new ServerSocket(port);
Socket s = srv.accept();
    s.getInputStream() & read()
    s.getOutputStream() & write()
 s.close();
```

25

Java Client

```
// active connect
Socket s = new Socket("server", port);
    s.getOutputStream() & write()
    s.getInputStream() & read()
s.close()
```

26

✓ Code

- Client (active) - Server (passive)
- Peer to Peer networks?
  - Who's the server?
- Distributed networks
  - Multiple servers and clients

27

Foundations

- ✓ Applications
- ✓ Requirements
- ✓ Architecture
- ✓ Performance
- ✓ Writing Code

Architecture

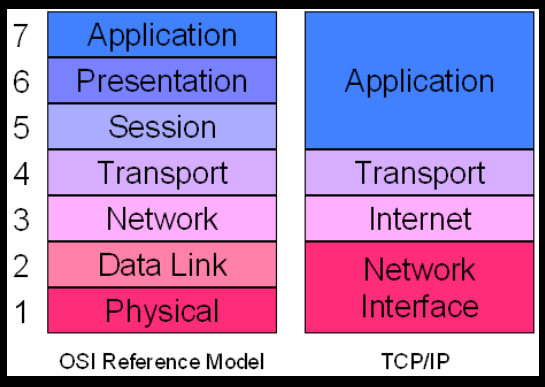
How do we build a Network so it is...

- understandable
- programmable
- flexible
- useful

28

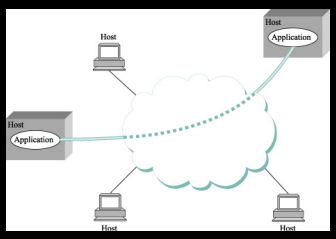
Layers make everything better

29



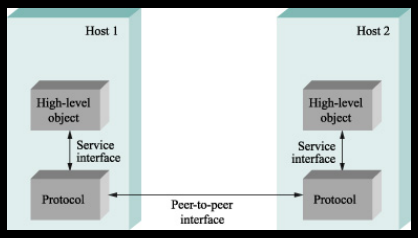
30

31



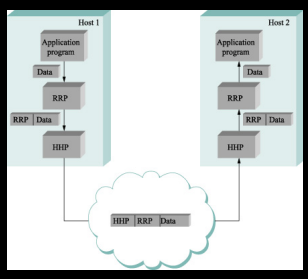
Application View

32



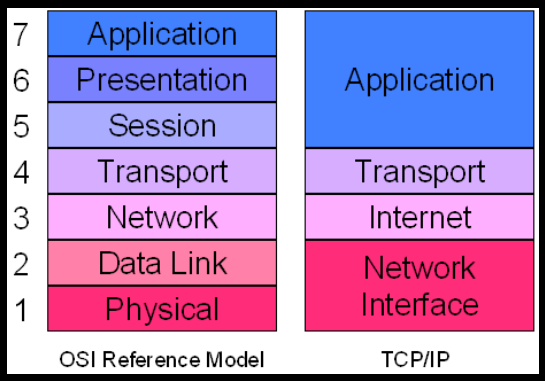
Service/Peer Interfaces

33



Encapsulation

34



35

✓ Architecture

Network Layers

- Subdivide the problem
- Address one at a time
- Do a good job at each layer

36

Foundations

- ✓ Applications
- ✓ Requirements
- ✓ Architecture
- Performance
- Writing Code

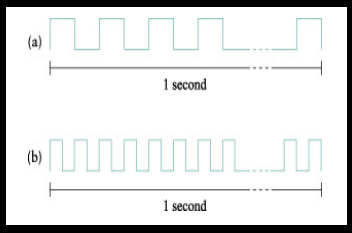
37

Performance

- Bandwidth
- Latency
- Delay X Bandwidth
- High-Speed Networks

38

Bandwidth



Bits transmitted per unit of time

39

Bandwidth

- 10 Mbps
  - 10 million bits per second
  - 0.1μS for each bit
- 20 Mbps; 0.5μS for each bit
- 100 Mbps; 0.01μS for each bit

40

Latency

How long does it take to get there?

41

Latency

- = Propagation + Transmit + Q-Delay
  - Propagation Delay (distance / c)
  - Transmit Time (size / bandwidth)
  - Queueing Delays

1-byte to/from server, 1Mbps
Transmit = 1us/bit = 8us
1-byte to/from server, 100Mbps
Transmit = 0.01us/bit = 0.08us
25MB file @ 10Mbps
25x10^6 x 8bits / 10x10^6 Mbps = 20sec

42

Latency

- Speed of Light (c)
  - copper = 2.3 x 10^8 m/s
  - vacuum = 3.0 x 10^8 m/s
  - fiber = 2.0 x 10^8 m/s

43

Latency

- One Way Latency
- Round Trip Time (RTT)

44

Bandwidth vs Latency

For...

- HTTP?
- File Transfer?
- Audio?
- Video?

45

Bandwidth vs Latency

Consider a 1-byte message and a 1-byte response over a 10Mbps link with a 10ms RTT.

…Does **bandwidth** or **latency** dominate the transmission?

…When will bandwidth dominate?

0.8us + 10ms = 10.0008ms — Latency dominates
Even at 1Gbps — 0.008us + 10ms = 10.000008 Latency dominates

@10Mbps; 10,000bits = 1ms; 100,000bits = 10ms; 12,500 bytes or ~12kB

46

Delay x Bandwidth

Amount of data "in-flight"

10 Mbps x 50 mS

(10×10^6) bits/sec x (50×10^{-3}) sec

$500 \times 10^3 = 500Kb$

62.5KB

47

High-Speed Networks

- Very large **bandwidth**
- Latency does **not** change
- Delay X Bandwidth goes **up**
- More data in-flight during RTT
- Latency begins to dominate transfer times.

48

✓ Performance

- Bandwidth
- Latency
- Delay x Bandwidth
- High-Speed networks

49

Foundations

- ✓ Applications
- ✓ Requirements
- ✓ Architecture
- ✓ Performance
- Writing Code

50

Foundations

End

51

```
# Exercise 1.4
1.5MB file,
80ms RTT,
1KB packet size,
2 RTT handshake at start

    size = (1.5 * 2^20) * 8 => 12,582,912
    rtt = 0.080
    packet_size = 2^10 * 8 => 8,192
    handshake = 2 * rtt => 0.16
```

52

```
# Exercise 1.4
1.5MB file, 80ms RTT, 1KB packet size, 2 RTT handshake
at start

    size = (1.5 * 2^20) * 8 => 12,582,912
    rtt = 0.080
    packet_size = 2^10 * 8 => 8,192
    handshake = 2 * rtt => 0.16

a) 10Mbps bandwidth, send continious

    bandwidth = 10 * 10^6 => 10,000,000

    propogation_delay = rtt / 2 => 0.04
    transmit_time = size / bandwidth => 1.2583

    handshake + propogation_delay + transmit_time
                                          => 1.4583
```

53

```
# Exercise 1.4
1.5MB file, 80ms RTT, 1KB packet size, 2 RTT handshake
at start

    size = (1.5 * 2^20) * 8 => 12,582,912
    rtt = 0.080
    packet_size = 2^10 * 8 => 8,192
    handshake = 2 * rtt => 0.16

b) 10Mbps bandwidth, wait 1 RTT after each packet

    bandwidth = 10 * 10^6 => 10,000,000

    propogation_delay = rtt / 2 => 0.04
    transmit_time = size / bandwidth => 1.2583

    packet_count = size / packet_size => 1,536
    t_time = transmit_time + ((packet_count - 1) * rtt)
                                       => 124.0583

    handshake + propogation_delay + t_time => 124.2583
```

54

```
# Exercise 1.4
1.5MB file, 80ms RTT, 1KB packet size, 2 RTT handshake
at start

    size = (1.5 * 2^20) * 8 => 12,582,912
    rtt = 0.080
    packet_size = 2^10 * 8 => 8,192
    handshake = 2 * rtt => 0.16

b) Alternate version

    bandwidth = 10 * 10^6 => 10,000,000
    propogation_delay = rtt / 2 => 0.04

    packet_time = (packet_size / bandwidth) + rtt
                                         => 0.0808
    t_time = (packet_count - 1) * packet_time
                                        => 124.0575

    handshake + propogation_delay + t_time => 124.2575
```

```
# Exercise 1.4
1.5MB file, 80ms RTT, 1KB packet size, 2 RTT handshake
at start

    size = (1.5 * 2^20) * 8 => 12,582,912
    rtt = 0.080
    packet_size = 2^10 * 8 => 8,192
    handshake = 2 * rtt => 0.16

c) infinite transmit, 20 packets per RTT

    packet_count = size / packet_size => 1,536
    packet_chunks = ceil(packet_count / 20) => 77

    handshake + (packet_chunks * rtt) => 6.32

Don't need to wait for response on last packet to
calculate arrival times.

    handshake + (packet_chunks * rtt)
              - propogation_delay => 6.28
```

55

```
# Exercise 1.4
1.5MB file, 80ms RTT, 1KB packet size, 2 RTT handshake
at start

    size = (1.5 * 2^20) * 8 => 12,582,912
    rtt = 0.080
    packet_size = 2^10 * 8 => 8,192
    handshake = 2 * rtt => 0.16

d) 0 transmit time, 1 packet first RTT, 2 second RTT, 4
third...

    packet_count = size / packet_size => 1,536
    packet_chunks = ceil(log(packet_count, 2)) => 11

Don't need to wait for response on last packet to
calculate arrival times.

    handshake + (packet_chunks * rtt)
              - propogation_delay => 1
```

56

```
# Exercise 1.3
1,000KB file,
50ms RTT,
1KB packet size,
2 RTT handshake at start

   size = (1,000 * 2^10) * 8 => 8,192,000
   rtt = 0.050 => 0.05
   packet_size = 2^10 * 8 => 8,192
   handshake = 2 * rtt => 0.1
```

57

58

```
# Exercise 1.3
1,000KB file, RTT 50ms, 2 RTT handshake at start

    size = (1,000 * 2^10) * 8 => 8,192,000
    rtt = 0.050 => 0.05
    packet_size = 2^10 * 8 => 8,192
    handshake = 2 * rtt => 0.1

a) 1.5Mbps, continuious

    bandwidth = 1.5 * 10^6 => 1,500,000
    propogation_delay = rtt / 2 => 0.025
    transmit_time = size / bandwidth => 5.4613

    handshake + propogation_delay + transmit_time
                                          => 5.5863
```

59

```
# Exercise 1.3
1,000KB file, RTT 50ms, 2 RTT handshake at start

    size = (1,000 * 2^10) * 8 => 8,192,000
    rtt = 0.050 => 0.05
    packet_size = 2^10 * 8 => 8,192
    handshake = 2 * rtt => 0.1

b) 1.5Mbps, one RTT after each packet

    bandwidth = 1.5 * 10^6 => 1,500,000
    propogation_delay = rtt / 2 => 0.025
    transmit_time = size / bandwidth => 5.4613

    packet_count = size / packet_size => 1,000
    t_time = transmit_time + ((packet_count - 1) * rtt)
                                         => 55.4113

    handshake + propogation_delay + t_time => 55.5363
```

60

```
# Exercise 1.3
1,000KB file, RTT 50ms, 2 RTT handshake at start

    size = (1,000 * 2^10) * 8 => 8,192,000
    rtt = 0.050 => 0.05
    packet_size = 2^10 * 8 => 8,192
    handshake = 2 * rtt => 0.1

c) infinite bandwidth, 20 packets per RTT

    packet_count = size / packet_size => 1,000
    packet_chunks = ceil(packet_count / 20) => 50

    handshake + (packet_chunks * rtt) => 2.6
    handshake + (packet_chunks * rtt)
              - propogation_delay => 2.575
```

```
# Exercise 1.3
1,000KB file, RTT 50ms, 2 RTT handshake at start

    size = (1,000 * 2^10) * 8 => 8,192,000
    rtt = 0.050 => 0.05
    packet_size = 2^10 * 8 => 8,192
    handshake = 2 * rtt => 0.1

d) exponential packet counts, 1, 2, 4, 8, ...

    packet_count = size / packet_size => 1,000
    packet_chunks = ceil(log(packet_count, 2)) => 10

Don't need to wait for response on last packet to
calculate arrival times.

    handshake + (packet_chunks * rtt)
              - propogation_delay => 0.575
```