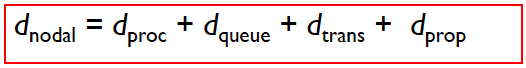
**Common Conversions**

**Time:** 1 second = 1,000 milliseconds

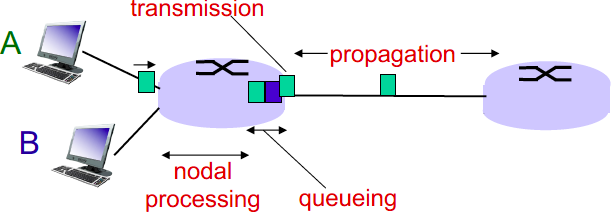
**Data Transfer:** 1 gigabit (Gb) = 1,000 megabits (Mb) 🡪 1 megabit (Mb) = 1,000 kilobits (Kb) 🡪 1 kilobit (Kb) = 1,000 bits (b)

**Storage:** 1 gigabyte(GB) = 1,024 megabytes (MB) 🡪 1 megabyte (MB) = 1,024 kilobytes (KB) 🡪 1 kilobyte = 1,024 bytes

**Week 1 – Introduction to Networks**



**Four Sources of Packet Delay**



**Combined delay = END-TO-END DELAY**

N \* [ proc + queue + trans + prop ]  
N = number of links

**1. Processing Delay (dPROC):** Checking bit errors, determine where to send the packet.

* **dPROC = < 1 ms. Usually done very quickly**

**2. Queuing Delay (dQUEUE):** Time waiting at the output link before transmission.

* **dQUEUE = Time spent in buffer**

**3. Transmission Delay (dTRANS):** Time required to push ALL BITS on the wire.

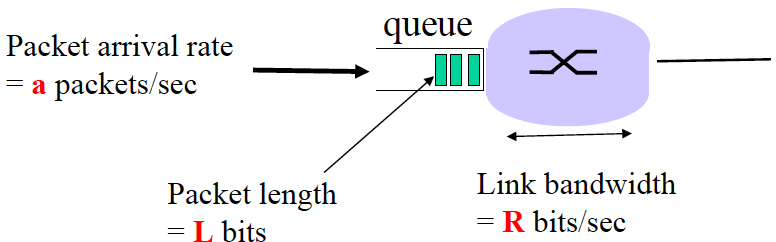
* **dTRANS = L / R where L = packet length / R = link bandwidth or rate**

**4. Propagation Delay (dPROP):** Time taken for ONE BIT to travel from the src 🡪 dest.

* **dPROP = d / s where d = physical length of link / s = propagation speed in medium**
* (~2 x 108 meters/s = speed of light / fiber)

**Queueing Delay**

**Traffic Intensity** = La / R (**La:** bits arriving at queue per second) (if **La > R** then **packets will drop**)

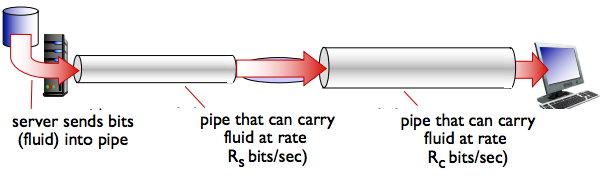
* **La/R ~= 0** is AVERAGE traffic intensity (Average is **L (N – 1) / (2R)** assuming queue is empty at t=0)
* **La/R = 0.8** should be FIXING YOUR NETWORK
* **La/R approaches -> 1** is BAD traffic intensity
* **La/R > 1** INFINITE delay, collapse of system

**Arrival Rate (a)** = R / L (**R/L:** packets arriving per second

**Throughput**

**Throughput**: The rate (bits / time unit) at which bits are transferred between sender / receiver.

* **Instantaneous:** Rate at given point in time (not useful, not accurate) | **Average:** Rate over long period of time (useful)



If RS < RC , what is the average throughput?

* RS is the average throughput

**The link on the end-end path that is the slowest will be the bottleneck. You can’t get faster than bottleneck.**

**Week 2 – Protocol Layering, Application Layer, Web**

**HTTP**

Uses a **Client-Server Model**. It is a **Stateless Protocol** (server maintains no info about past client requests).

HTTP is **all text**, which makes the protocol simple but not the most efficient as “12345678” as a str is 8-bytes vs 4-byte int rep.

**HTTP Performance Goals**: As a user you want fast DL rates, as a provider you want happy users + cost-effective infrastructure

* **Caching and Replication**: Replicate content, don’t need to go to origin server every time.
* **Improve HTTP**: Achieve faster downloads + compensate for TCP’s weak spots.
* **Exploit Economies of Scale:** CDN’s, datacentres.

**Non-Persistent HTTP**: At most one object sent over TCP connection, connection is then closed.

* Downloading multiple objects require multiple downloads.

**Persistent HTTP**: Server leaves TCP connection open after sending response.

* Persistent without Pipelining: Client issues new request only when prev response has been received (1 RTT per object
* Persistent with Pipelining HTTP/1.1: Client requests as soon as it encounters a referenced object (1 RTT for ALL objects)

**Locality of Reference**: Scenario where same values/storage locations are frequently accessed. Therefore, caching works.

* However, if there are many UNIQUE requests, benefits of caching start to go downs

**Caching**

**Access Link Utilisation**

* **Access Link Utilisation = bits per second requested / access link rate**
* When access link utilisation is high, queueing is high.
* With caching, if cache hit rate = 0.4 🡪 access link utilisation = 0.6

**Caching Solution**

* **Total delay =** **(Cache Hit Rate \* delay at cache) + (Access Link Utilisation \* delay from sever)**

**Caching Example**

15 requests per second

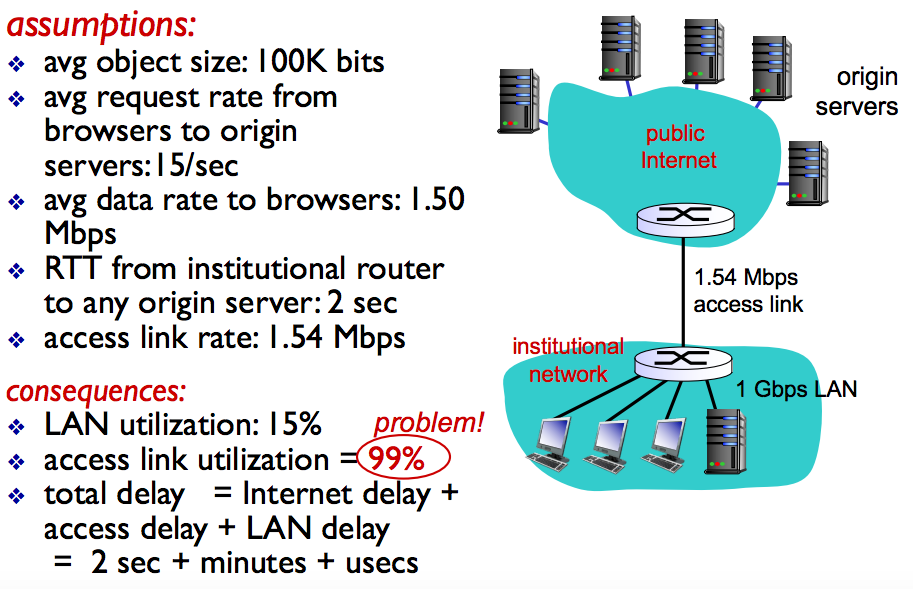
100k (100 \* 103) bits per request

**A = 15 x 100 \* 103 bits**

1.54 Mbps Link Speed  
**B = 1.54 x 106 bits**(1 megabit = 106 bits)

A / B = 0.974 **= 97% access link utilisation = ~2+ seconds**

With Caching Solution of 0.4 hit rate: Total delay is  
**= (60% access link utilisation \* 2 secs) + (40% \* ~millisecs)  
=** **~1.2 seconds**



**UDP and TCP**

**Socket** is a door between application process and end-to-end transport (an endpoint)

* UDP Socket = Datagram Socket or SOCK\_DGRAM
* TCP Socket = Socket /ServerSocket or SOCK\_STREAM

|  |  |
| --- | --- |
| **TCP** | **UDP** |
| **Connection-orientated**: setup required between C-S  **Reliable transport** between sending and receiving  **Flow control**: sender won’t overwhelm receiver  **Congestion control**: throttle sender when network overload  **Does not provide**: timing or minimum bandwidth guarantees | **Message-orientated**  **Unreliable data transfer** between sender and receiver  **Does not provide**: connection setup, reliability, flow control, congestion control, timing or bandwidth guarantees |
| Email  Web Browsing | VoIP  Music Streaming |

**Week 3 – Application Layer**

**SMTP**

**Simple Mail Transfer Protocol** is a protocol which uses TPC to reliably transfer files BETWEEN MAIL SERVERS (port 25)

* Three phases of transfer: **(1) handshake / open TCP connection (2) transfer of message (3) close TCP connection**
* Messages must be in 7bit ASCII
* SMT is a PUSH protocol

**Mail Access Protocols**

**Mail Access Protocols** handle the retrieval of emails / files from the mail server.

* **POP**: Post Office Protocol. Features = authorisation, download
* **IMAP**: Internet Mail Access Protocol. Features = manipulation of stored messages + auth, download
* **HTTP(S)**: Gmail, Yahoo Mail etc. HTTP(S) is a PULL protocol

**PULL**: Someone loads data on a web server and users use HTTP to pull data from their server. TCP connection initiated by machine that wants to receive the file.

**PUSH**: The sending email server pushes the data to the receiving mail server. TCP initiated by machine that wants to send file.

**Domain Name System (DNS)**

**DNS** is a hierarchy of many name servers. Key idea of DNS is **Hierarchy**.

**1. Hierarchical Namespace**: Naming scheme that allows for delegation of namespaces to parties.

**2. Hierarchical Administration**: A **zone** corresponds to an admin authority that is responsible for that portion of hierarchy.

* E.g. UBC controls .berkeley.edu

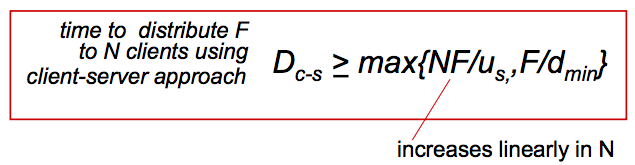
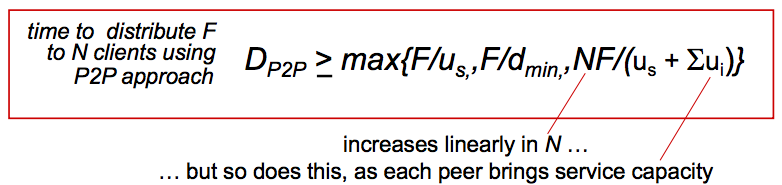
**3. Server Hierarchy**:

* Top of hierarchy = ROOT SERVERS (hardwired into other servers)
* Next level = TOP-LEVEL DOMAINS (TLD) such as .com .edu etc.
* Bottom level = Authoritative DNS Servers (store the name-to-address mapping + maintained by admin authority)  
  Authoritative DNS Servers stores all resource records for the names in the domain that it has authority for.

Each server stores a small subset of the total DNS database.

**Video Streaming and CDN’s**

**Week 4 – P2P Networks and Transport Layer**

**Client-Server Model P2P Model**

Performance of C-S vs. P2P: How much time does it take to distribute a file to N peers?

**N = # peers | F = file size | us = server upload capacity | ui = peer upload capacity | di= peer download capacity**