**W1 Quiz + Finish**

1. Packet switching, instead of circuit switching, is generally used to transfer data in the internet. T or F? = **TRUE**

2. Propagation delay depends on the size of the packet. T or F? = **FALSE** (Propagation depends on the type of medium + length)

3. Which of the following delays is significantly affected by load in the network? = **Queuing** **Delay**

* If the load is high, packets will just sit in the queue increasing queuing delay.

4. What is the correct order of delays encountered by a packet until it reaches the next-hop router?

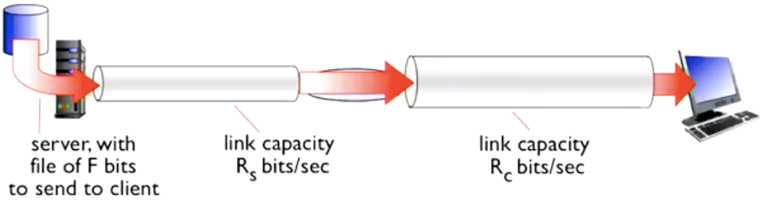
* **Processing, Queuing, Transmission, Propagation**

Packet Loss

* The Queue (buffer) has a finite capacity. A packet which arrives to a full queue will drop.
* A lost packet may be re-transmitted by a previous node, by the source system or not at all.

Throughput

* Throughput is the rate **(bits / time unit)** at which bits are transmitted between the sender / receiver.
  + **Instantaneous**: Rate at a given point in time. (not very useful – may not give the true data)
  + **Average**: Rate over a longer period of time. (more useful – more accurate)



* Different pipe rates:
  + **RS < RC** , what is the average end-end throughput?  
    Average = RS because even if RC is faster, it can’t send any faster than packets arriving at the RS rate.
  + **RS > RC** , what is the average end-end throughput?  
    Average = RC because even of the sane reason above. **(Bottleneck Bandwidth)**
* Throughput: Internet scenario
  + Per-connection end-end throughput: min(RC , RS , R / 10). “10 connections share bottleneck link R bits/sec”
  + Wherever the bottleneck is, you won’t get a faster rate than the bottleneck.

Three networking design steps:

* (1) Break down the problem into tasks (2) Organise these tasks (3) Decide who does what.

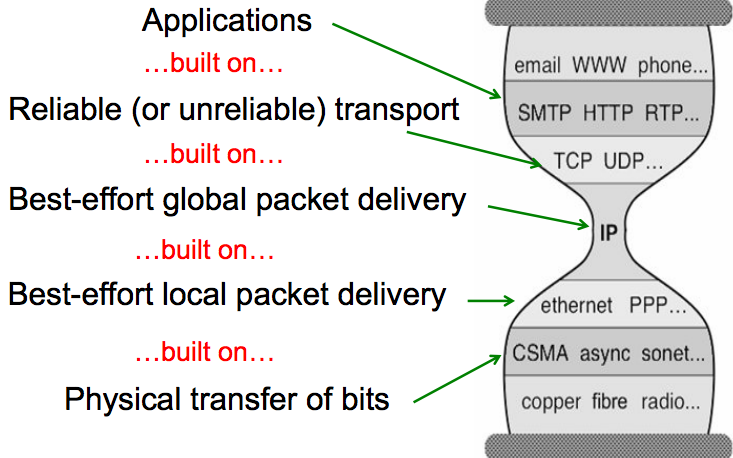
Tasks in networking:

* TASK 1: Send along a single wire
* TASK 2: Stitch these together to go across country

Tasks in networking / Layers

* The table below shows the decomposition of a network (bottom up approach)

|  |  |
| --- | --- |
| **Tasks** | **Layer** |
| 1. Bits on wire | Physical |
| 2. Packets on wire | Physical |
| 3. Deliver frame within local network | Datalink |
| 4. Deliver packets across global network | Network |
| 5. Ensure that packets get to the dest | Transport |
| 6. Do stuff with the data | Application |



* E.g. CEO A sending a letter to CEO B. FedEx will take it to a hub, where it gets sorted then forwarded to the other country.
* CEO doesn’t need to know which hub it goes to, where it will be forwarded to etc.
* Applications in Networks are the “CEO’s” and they need to work with the Transport Layer.

Internet Protocol Stack (stack because they are layered on top of each other)

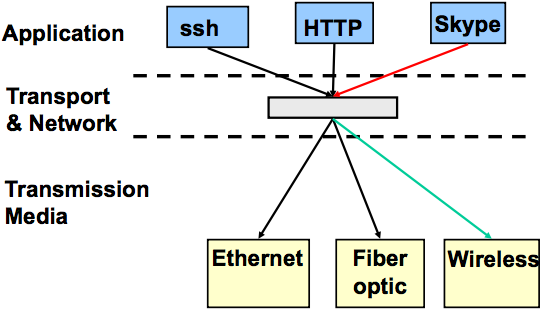
* **Application** **Layer**: Supporting networking applications (FTP, SMTP, HTTP, Skype…)
* **Transport Layer**: Process to process data transfer (TCP, UDP)
* **Network Layer**: (Unifying Protocol) Routing of datagrams from source to destination (IP, Routing Protocols)
* **Link Layer**: Each link that we connect to our own device.
  + Data transfer between neighbouring network elements (Ethernet, 802.111 (WiFi), PPP (Point-to-Point))
* **Physical Layer**: Bits “on the wire”. Can be a physical wire or wireless (WiFi)

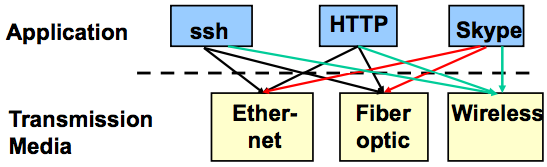
Each layer:

* (1) Depends on the layer below (2) Supports the layer above (3) Independent of others

What are the benefits of layering?

* Abstraction: Applications don’t need to know everything to transmit data to the destination
* API: What hooks are used to provide information to the above layer? What hooks to communicate with layer below?
* **Introduction of an intermediate layer provides a COMMON abstraction for various network technologies**





VS

Can layering be bad?

* Layer N may duplicate lower level functionality (error recovery to re-transmit lost data)
* Information hiding may hurt performance: Packet loss due to corruption vs. congestion
* Headers start to get really big: Typically TCP + IP + Ethernet headers add up to 54 bytes.
* Layer violations when the gains are too great to resist: TCP-over-wireless
* Layer violations when the network doesn’t trust ends: E.g. firewalls

If layering doesn’t exist, each new application (Skype) has to be re-implemented for every network technology = **INEFFICIENT**

Distributing layers across the network

* Layers are simple if only on a single machine: It is just a stack of modules interacting with those above / below
* However, we need to implement layers across machines: Hosts / Routers / Switches

What gets implemented on an end-system (i.e. Host)?

* Bits arrive on wire. All layers must exist at host.

What gets implemented on a router?

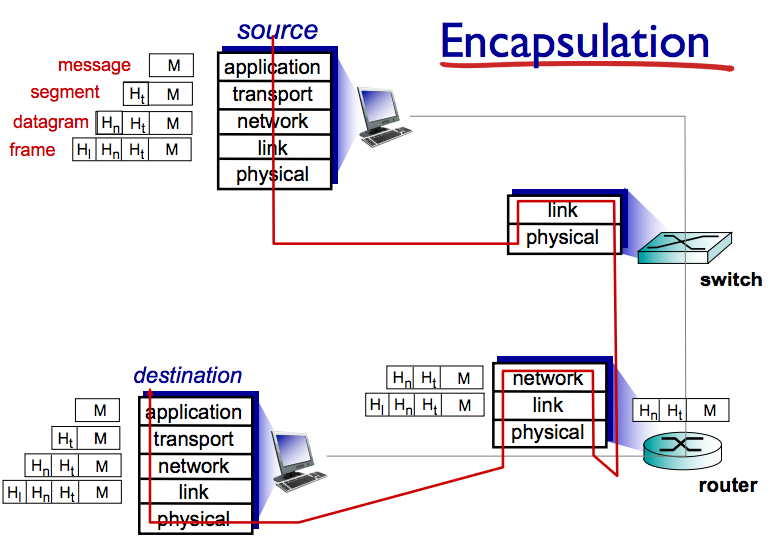
* Bits arrive on wire: Physical layer is unnecessary
* Packets must be delivered to the next-hop: Data-link layer is necessary
* Routers participate in global delivery: Network layer is necessary
* Routers don’t support reliable delivery: Transport Layer (and above) is NOT supported.
* The job of routers/switches is to just forward it to the next-hop. After that, they don’t care about other layers.

Logical Communication: Layers interact with peer’s corresponding layers

* Application Layer, such as HTTP: Two peers (such as a client / server) are interacting with each other. No one else needs to understand anything about HTTP protocol.
* HTTP uses a Transport Protocol (such as TCP). The Transport Layer is done in one end-system and another end-system as an example and they are talking to each other LOGICALLY (not physically, as they are NOT CONNECTED).
* In the Network Layer, the Link Layer and the Physical Layer, the systems are talking to each other.

Physical Communication

* Communication goes downwards to the physical network, then layers get processed, transmitted to another link then communication progresses upwards to the other end’s application layer.



M = bits / file that you want to send

Ht = header / metadata related to T.L

Network layer appends own metadata Hn

= datagram

Link layer appends won metadata Hl

= frame

Finally sent to physical layer.

Switch runs Link + Physical Layer.

Router runs Network + Link + Physical

**The metadata is only treated at their relevant layer.**

When it reaches dest, as it moves up the stack, each header will be stripped off one-at-a-time and passed up the layers.

**Application Layer**

Examples of networked applications: Skype, online games, torrents

Creating a network app:

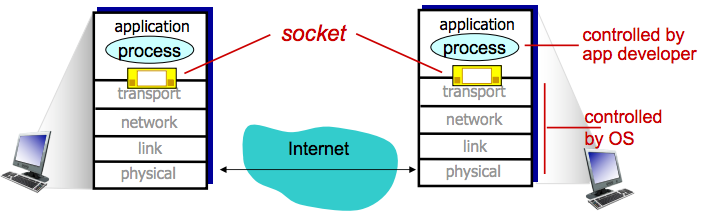
* **Write programs that:** Run on different end systems / communicates over network / E.g. web server software communicates with browser software.
* **Varying degrees of integration:** LOOSE: email, web browsing / MEDIUM: chat, Skype, remote file systems /   
  TIGHT: process migration, distributed file systems
* **No need to write software for network-core devices**: Network-core devices do not run user applications  
  Applications on end systems allows for rapid app development, propagation.

**Interprocess Communication (IPC)**

* Processes talk to each other through Interprocess Communication (IPC)
* On a single machine: shared memory / Across machines: we need other abstractions (message passing)

**Sockets**

* Sockets allow communication between two different processes on the same / different machines.
* The Application or Processes sends / receives messages to / from its sockets.
* Sockets = similar to doors:
  + Sending process shoves message out the door
  + Sending process relies on transport infrastructure on the other side of the door to deliver the message to the socket at the receiving process.



**Addressing Processes**

* To receive messages, processes must have an identifier. Host devices have a unique 32-bit IP address.
* QN: Is the IP address of the host on which the process is running enough for identifying the process?
  + ANS: No, many processes can be running on the same host.
* **Identifier = IP Address + Port Number** associated with the process on the host.
  + E.g. HTTP server: port 80 / mail server: port 25
  + E.g. To send HTTPmessage to cse.unsw.edu.au web server: IP address = 129.94.242.51 / Port = 80

**Client-Server Architecture**

* **Server**
  + Exports well-defined requests / respond interface.
  + Long-lived process that waits for requests
  + Upon receiving a request, carries out the request.
* **Clients**
  + Short-lived process that makes requests. It is the “User-side” of the application
  + Initiates the connection

**Client vs. Server**

|  |  |
| --- | --- |
| **Client** | **Serve**r |
| Always-on host  Permanent IP address  Static host conventions (http: 80 | email: 25 | ssh: 25)  Data centres for scaling  May communicate with other servers to respond | May be irregularly connected  May have a dynamic IP address  Do not communicate directly with each other |

**P2P Architecture**

* There is no “always-on” server. No permanent address. There is no single server
* It is a dynamic system, people can join and become part of the peer system, so that can provide and receive a service from other people.
* Arbitrary end systems (peers) directly communicate with each other
* There is symmetric responsibility (unlike client / server)
* Often used for: File sharing (torrents), Gaming, Video Distribution, Video Chat, in general: “distributed systems”

Pros and Cons of P2P

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| **Self-scalability:** new peers bring new service capability, as well as new service demands.  **Speed**: Parallelism, less disagreements  **Reliability**: Redundancy, fault tolerance  **Geographic Distribution** | Fundamental problems of decentralised control:  **Synchronisation**: If one thing changes, everyone else has to update with the new changes too, otherwise it won’t work with new and old copies of the file everywhere. |

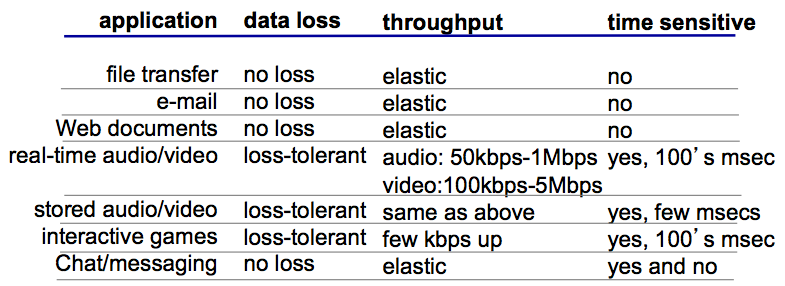
**Application-Layer Protocols**

* **Types of messages**: Request, response
* **Message syntax**: What fields in messages and how fields are delineated.
* **Message semantics**: How do I interpret the information in the fields? What actions do I take?
* **Rules**: For when and how processes send and respond to messages.
* **Open Protocols**: Defined in RFC’s (Request for Comments) / allows for interoperability / HTTP, SMTP.
  + Purpose of these standard protocols: Anyone can develop off these protocols
* **Proprietary Protocols**: Skype, Google etc. who may have developed more efficient technology

**What transport service does an app need? (application layer makes use of the transport layer)**

* **Data Integrity**: No one has tampered with information. Some apps require 100% reliable data transfer.
* **Timing**: Some apps require low delay to be “effective”. E.g. voice apps, online games.
* **Throughput *(units of info processed in a given amount of time)***: Some apps require minimum amount of throughput to be effective. Other apps make use of whatever throughput they get.
* **Security**: Encryption, data integrity

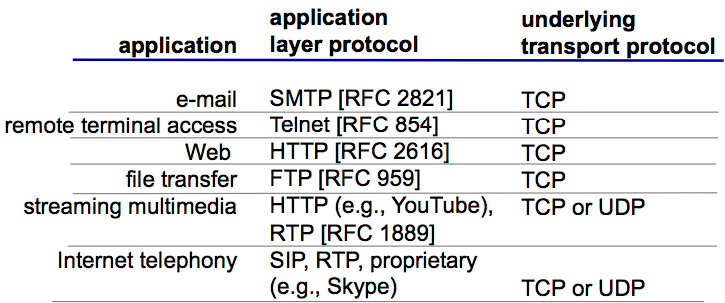
**Transport Service Requirements: Common Apps**



**Internet transport protocol services**

Transmission Control Protocol Service (TCP) vs. User Datagram Protocol Service (UDP) – More on this later.

**Internet apps: Applications, transport protocols**



**NOTE: Applications and Application Layer Protocols are DIFFERENT** **THINGS**

* Applications have a lot more components involved, but they are using standard application layer protocols which make them useable, so even if you use different mail clients (Gmail, Outlook etc.) or servers, you are still able to communicate.
* Applications USE Application Layer Protocols.
* Applications are usually NOT-standardised, but Application Layer Protocols are.

**Web and HTTP**

* A web page consists of objects: HTML files, JPEG images, Java applets, audio files etc.
* It consists of a base HTML file which includes several references objects
* [www.someschool.edu/someDept/pic.gif](http://www.someschool.edu/someDept/pic.gif)  
  hostname | pathname /someDept/pic.gif