Networking: Layering and End to End Argument

Aurojit Panda

[Slides heavily borrowed from Ion + Others]

Why networking in a systems class?

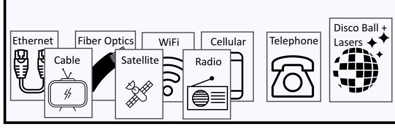
- Many of the most interesting systems today rely on the network.
  - Cellphone games, Siri/Alexa/…, Large data processing, MMORPGs, etc.
  - Important to understand networks.
  - **Note**: This lecture is ***insufficient*** for this purpose.
- The Internet is a very large system.
  - Survived for a few decades now.
  - Scaled to 47% of world population, a variety of applications, and all sorts of connections.
- How?

What is the Problem?

E-mail | Browsers (http/https) | ssh | iTunes (DMAP) | Skype | ftp | BitTorrent | Gopher

Many applications

connected over

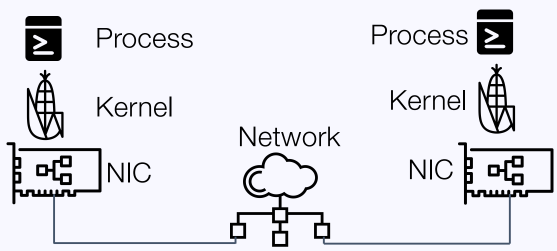


many networks

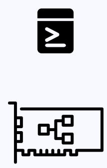
Solution

- Two architectural principles:
  - Layering: **abstraction** in another guise.
  - End-to-end principle: architectural on where functionality is placed.
- Note: these principles are not absolute.
  - Long running debates on what both of these mean and how implemented.
  - How they are used and what they mean is often dictated by performance.

Preliminaries: Participants



Preliminaries: Identity/Names



TCP/UDP port (16 bit) assigned by OS.

IP address (32/64 bit) network assigned.

MAC addr (48 bits) burnt in hardware.

Preliminaries: Goals

- **Delivery**: Packets from Alice reach Bob
  - Alice and Bob can be in the same room, or in different countries.
- **Reliability**: Packets from Alice reach Bob even when things go wrong.
  - For simplicity physical hardware provides **best effort** service.
  - Software or protocols hide temporary errors from users.
- **Congestion Control:** Allow different users to share the network.
- **Flow Control**: Deal with bounded buffers on devices in the network.

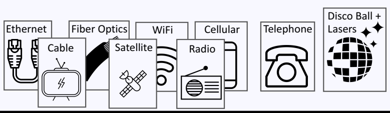
Layering

What is the Problem?

E-mail | Browsers (http/https) | ssh | iTunes (DMAP) | Skype | ftp | BitTorrent | Gopher

Many applications

connected over



many networks

How to make networked applications?

- Application per physical network type doesn't work out.
  - Too much work for developers.
  - Fragments the world into different types of networks.
- Internet interconnects networks. How?
- Answer through abstraction.

Layering

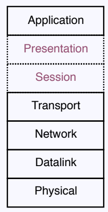
- Partition system into layers (abstractions).
- Each layer expects some functionality to be implemented by layer below.
  - **Note**: My layers grow upwards in this (and subsequent slide).
  - No assumption on how functionality is implemented.
- Each layer provides some functionality to layer above.
  - Can make assumptions on how functionality will be used and restrict interface.
- Layers interact through some agreed upon interface.
  - Defined by operating system (e.g., drivers), standardization body, etc.

Layering: Why?

- Assumptions and interfaces are well specified.
- Can swap out one layer for another assuming interfaces remain constant.
- In networking this means:
  - Applications assume delivery, reliability, congestion control, flow control, etc.
  - Application logic assumes these and is independent of how these are implemented.
  - User/kernel gets to choose congestion control or flow control algorithms.
  - Network operator/owner gets to choose how to route and deliver packets.
- Changing one layer does not require changes to any of the other layers.

Layering in Networking: OSI Model

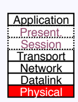
- Open Systems Interconnection (OSI) model (1983/4)
  - ISO + CCITT developed this model in 1983-4
  - Defined **seven** layers.
- TCP/IP Transport Control Protocol/Internet Protocol
  - Only considers **five** layers
  - Presentation and session functionality implemented by application.



Layering in Networking

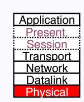
- Going to analyze three questions for each layer
- **Service**: What functionality is implemented by the layer?
- **Interface:** How does the next layer access this functionality?
- **Protocol**: What messages does layer send over the network?
  - Rules and packet formats expected by this layer at each node.
  - How is the **service** implemented between **machines** in the network.

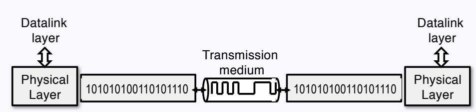
Layer 1: Physical Layer



- **Service**: Send bits over some physical medium.
  - Could be electrical signals, optical signals, radio signals, …
- **Interface**: Send and receive streams of bits.
  - The precise interface depends on the operating system (or library) and NIC.
- **Protocol**: Decides how data is encoded in the medium.
  - Code used for encoding bits, detecting or correcting physical errors, etc.

Layer 1: Physical Layer



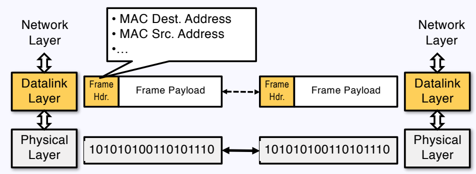


Layer 2: Datalink Layer

- **Service**: Allow physically connected devices to exchange data frames
  - Exchange messages when connected to the same wired or wireless link.
- **Interface**: Send and receive frames addressed to a device.
- **Protocols**: Several protocols available, depends on physical media.
  - Examples: Ethernet, 802.11 (wireless), Point-to-Point (PPP), etc.

Layer 2: Datalink Layer

- Each frame has a header
  - Specifies source and a destination MAC address
- MAC address is 48 bits, assigned by NIC manufacturer.



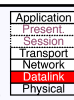
Layer 2: Datalink Layer

- Notice there are a variety of datalink layer protocols.
  - Some of these depend on the physical link connecting machines.
- Why depend on physical links?
  - Because some physical links connect exactly two devices. Called **point-to-point** links.
    - Example: Almost all wired networks today.
  - But others are shared by many different devices. Called **broadcast channels**.
    - Example: WiFi (radio frequency shared by all users), old ethernet (10BASE-2), etc.
- When sharing physical links need to decide who gets to send data.

Layer 2: Sharing the Link

- Problem: Multiple devices share a single link.
  - All devices on the same link hear any message that is transmitted.
  - If two devices transmit simultaneously message gets garbled.
- How to avoid collisions, i.e., take turn speaking on shared media?
- Three solutions
  - Partition the channel.
  - Take turns
  - Random access: Detect when network is unused and try to transmit.

Layer 2: Partitioning Channel

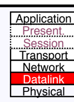


- Split a single channel into N channels each with 1/N bandwidth.
- Each device gets one of the channels.
- How to split channels
  - Frequency division (FDMA), time division (TDMA), code division (CDMA)
- Problem: Wasted bandwidth when a device has no data to send.
  - No one can use unused bandwidth.

Layer 2: Taking Turns

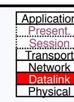
- Pass a token around between active devices.
- Devices can send data if and only if it has token.
- Device passes token along if no data to send.
- Example: Token ring.
- **Pros**: More efficient, less wasted bandwidth.
- **Cons**: Susceptible to failure, what happens when token gets lost.

Layer 2: Random Access



- Three steps to get multiple nodes to access link
  - **Carrier Sense** (CS): First check if someone else is using the link.
    - If yes then wait until they are done, otherwise try and use the link.
  - **Collison Detection** (CD): When transmitting detect if someone else uses the link.
    - If collision is detected then stop transmitting.
  - **Random Wait**: When collision is detected wait for some random time before retry.
- Example: CSMA/CD used by ethernet.
- Pros: Efficient at low load, not susceptible to failure.
- Cons: High overhead during collision.

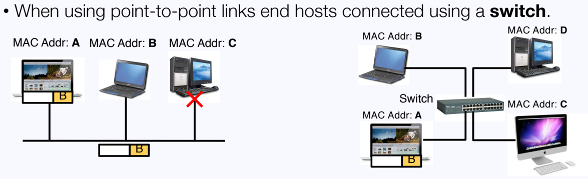
Layer 2: Sharing the Link



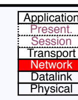
- Looked at three solutions
  - Partitioning channel: FDMA/TDMA/CDMA. Used in some cellular networks.
  - Taking Turns: Token ring. Popular for local networks in the 1980s and 1990s.
  - Random Access: CSMA/CD. Used by ethernet.
- Other solutions exist and are used in other contexts.
  - Largely constructed from similar primitives.
  - Example: CSMA/CD in wireless networks.

Layer 2: Local Area Network

- A set of end hosts which can communicate with each other over layer 2.
- Implemented using both **broadcast channels** and **point-to-point links**.
- When using point-to-point links end hosts connected using a **switch**.

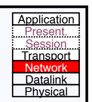


Layer 3: (Inter)Network Layer



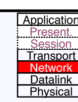
- **Service**: Deliver packet to a specified **network address.**
  - Network address might refer to host on another data link layer.
  - **Connect** multiple layer 2 networks together.
- **Interface**: Send packets to a remote network address.
  - Receive packets addressed to computer.
- **Protocol**: Internet Protocol (IP). Define network addresses, forwarding.

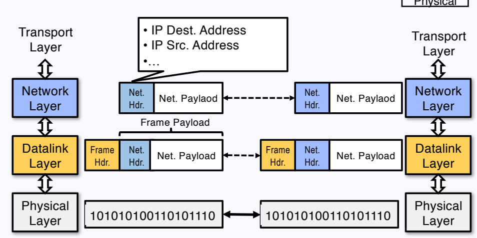
Layer 3: (Inter)Network Layer



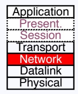
- End hosts are identified by IP addresses
  - Each address is 32 (IPv4) or 128 (IPv6) bits.
- Addresses are ”globally unique”.
  - Uniqueness necessary to ensure that end hosts all over the world can talk.
- Assigned by network operator
  - Either statically configured or (more likely) configured when computer connects.

Layer 3: (Inter)Network Layer



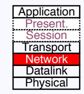


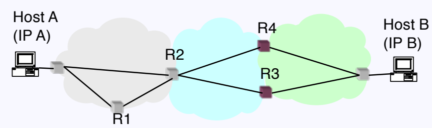
Layer 3: Where Used?



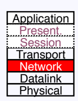
- Used to connect several layer 2 (LANs) together.
  - When connecting a large organization (e.g., Berkeley's network)
  - When connection a geographically spread out network (e.g., AT&T's network).
  - When connecting multiple networks together (e.g., the Internet).
- How are LANs interconnected? Through **routers**.
- Routers bridge LANs together.

Layer 3: Where Used?

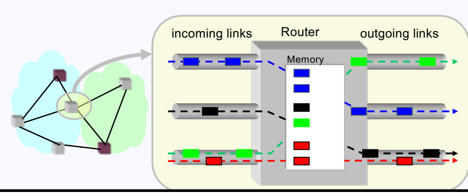




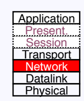
Layer 3: How Impelemented? Routers



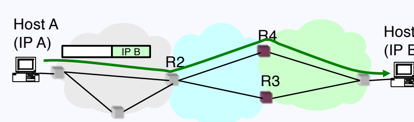
- Forward each packet on a link.
- Decides how to forward a packet depending on the forwarding table.



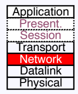
Layer 3: Packet Forwarding



- How it works:
  - Router waits to receive the entire packet (**store and forward**).
  - Once received uses destination address to index **forwarding table**.
  - Forwards packet along output port specified by forwarding table.

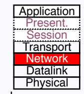


Layer 3: Why use IP addresses?



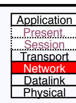
- Layer 2 used MAC address, why not use that at Layer 3?
  - Scalability: IP addresses are hierarchical, MAC addresses are not.
- MAC addresses are assigned by a device vendor
  - Uniquely identifies your hardware. Invariant as you move.
- IP addresses change depending on where you are connected.
  - For example, you get a different IP address at home vs when at Berkeley.

Layer 3: Why use IP addresses?



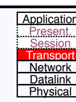
- Networks assign IP addresses. As a result addresses identify routes.
  - For example: any IP address of the form 128.32.xxx.xxx belongs to Berkeley.
  - Any IP address of the form 128.30.xxx.xxx belongs to MIT's CS Department.
- Can route traffic to 65535 Berkeley hosts by just using 16 bits of space.
- Same for MIT.
- By contrast: MAC address varies by who manufactured computer.
  - Need an entry per end host to route packets. Does not scale globally.

Layer 3: Internet Protocol (IP)



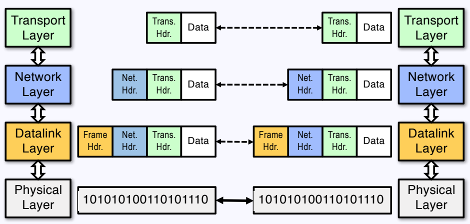
- Layer 3 on the Internet.
- Provides "best-effort" packet delivery.
  - Will try its best to deliver packets to the destination.
  - Packets might be lost, or corrupted.
  - Makes no ordering guarantees. Packets can be arbitrarily reordered.

Layer 4: Transport Layer



- **Service:** End-to-end communication between processes.
  - Demultiplex communication between processes running on the same end host.
  - Might implement: reliability, congestion control, timing, etc.
- **Interface**:
  - Send message to a process.
  - Establish connection with a process.
  - Receive messages sent to a process.
- **Protocol**: TCP and UDP: define port numbers for demultiplexing processes.

Layer 4: Transport Layer



Layer 4: Port Numbers

- 16-bit numbers carried in the transport layer header.
- Identifies what process sent/should receive a packet.
- A port is only meaningful at a particular end host.
- Assigned by the kernel (network stack).
- When sending messages need to know port to use to reach a **process**.
  - Client learns this out-of-band. Many services have known port (e.g., 80 is http).
  - Server can learn port from the initial request: all packets carry source port.

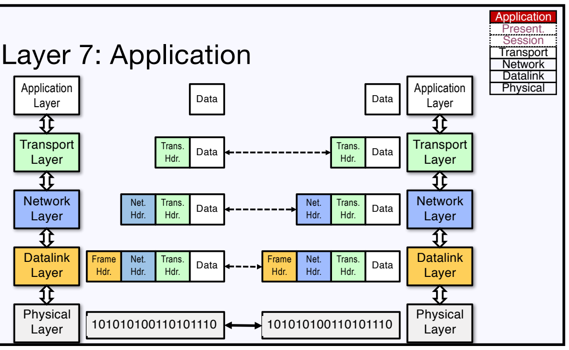
Layer 4: Protocols

- UDP: Datagram service
  - Send best effort messages over IP. Single IP packet with ports specified.
- TCP: Reliable, in-order stream
  - Connection oriented: one process connects to another.
  - Protocol is responsible for discarding corrupted packets.
  - Protocol retransmits any lost packets.
  - Performance congestion control and flow control.

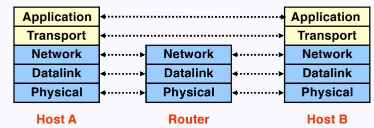
Layer 7: Application

- **Service**: depends on application: e-mail, HTTP, etc.
- **Interface**: depends on application: some UI, some API, etc.
- **Protocol**: depends on application: HTTP, SMTP, …

Layer 7: Application

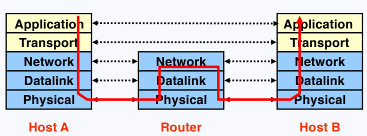


Network Layer Summary



- Physical, Datalink and Network layer are implemented on all devices.
- Application and transport are implemented only on hosts.
- Logically each layer interacts with the corresponding layer on other devices

Physical Communication



- When sent packets go down the layers until they reach the physical layer.
- When received packets go up the layers until processed.

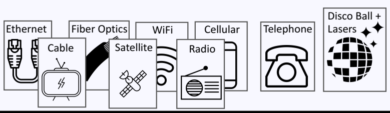
Break

Back to the Beginning

E-mail | Browsers (http/https) | ssh | iTunes (DMAP)
Skype | ftp | BitTorrent | Gopher

Many applications

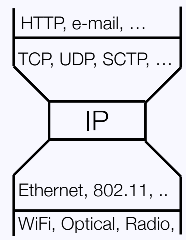
connected over



many networks

How Does the Internet Solve This?

- Chose a single “narrow waist”
  - Everyone agreed on IP as interconnection protocol.
- Everything else could vary.
- Responsible for Internet’s success today.
  - Usable in planes, space station, rural areas, etc.
  - Usable by a wide variety of applications.
- Was it the right layer to standardize?



Layering: A Critique

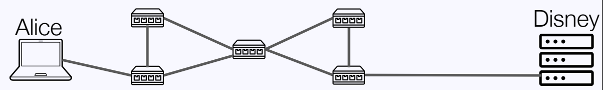
- Abstraction and indirection are the two hammers we use to solve problems
- Except for the problems introduced by these approaches.
  - Layering can hurt performance if implemented naively.
  - Information hiding makes it hard to implement some protocols.
  - Header overheads.
  - Figuring out if this set of layers is even correct. For example where does TLS fit?
- Reality: Layering is a powerful design tool, but not a silver bullet.

End to End Principle

End-to-End: Origins

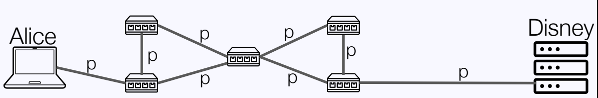
- End-to-End Arguments in System Design: Saltzer, Reed, Clark 1981
- Design principle for placement of functions in distributed computer systems.
- Question from the early days of the Internet
    - What should the network implement to support an application?
    - How does this support impact other applications?
    - What services should the application provide?
- E2E is widely considered to be the foundation of how the Internet is designed.
    - Everyone agrees it is a good idea, people often disagree on what the idea states.

End-to-End: Example



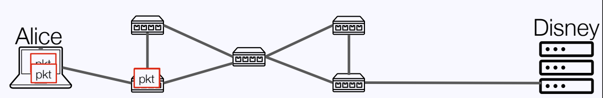
- Original challenge: Achieving reliability from an unreliable network.
    - Links can only be so reliable: things fail.
    - How can Alice reliably download Star Wars Ep VIII from Disney?

End-to-End: Example



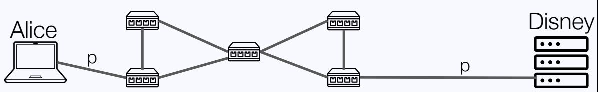
- Problems when downloading file
    - Assume that when going through a link packet can be lost with probability **p**.
    - How to make sure that file is eventually delivered?
- Two options: Hop by hop reliability, end to end reliability.

End-to-End: Example



- Hop-by-Hop reliability: Each router implements reliability mechanism.
    - Keep sending packet until other router receives the packet.
- End-to-End reliability: The computers implement reliability.
    - Keep sending packet until other **end-host** receives packet.

End-to-End: Example

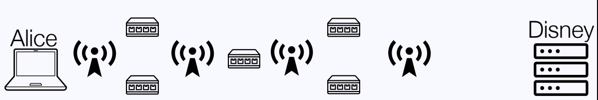


- Which is better?
  - Observe that end-hosts need to implement reliability: last hop can loose packets.
  - Hop-by-hop reliability adds overheads even when no reliability is required.
  - When probability p is low, doing it at end hosts is good.
- Chance that a packet is lost is only slightly higher than when you do hop by hop.

End-to-End Principle

- Implement functionality as high up in the stack as possible.
  - Example: Do not implement encryption in the kernel, do it at application level.
  - Example: No reliability or ordering in network, do it in transport layer.
- Otherwise: might affect other applications who do not require functionality.
- Important exception: Performance.
  - Implement functionality lower down if necessary for performance.
- Note: E2E is an important design principle, but not gospel.
  - Can almost always take the performance exception.
  - A reasonable interpretation: Be cognizant when adding functionality to lower layers.

End-to-End: Exception



- Switching to wireless (from wired) medium increases loss probability.
- As a result most wireless protocols today support hop-by-hop reliability.
- Does this violate end-to-end?
- Does it matter?

Summary

- The Internet is one of the largest real systems.
  - Survived several decades, exponential growth, etc. with very few changes.
  - It just works, mostly.
- Principles that helped: Layering, End to End

Summary: Layering

- Abstractions that allow networks to be modularized.
  - Can change lower layers of the network without affecting application.
  - Can change applications without impacting other layers of the network.
- Internet today has five layers:
  - Physical Layer: Send bits on a wire, over the radio, etc.
  - Datalink Layer: Connect physically linked devices.
  - Network: Connect end hosts over a WAN or across the Internet.
  - Transport: Connect two processes across a network.
  - Application: Implement interesting features that lead people to use the network.

Summary: End to End Principle

- Implement functionality as high up in the stack as possible.
  - Avoid placing functionality in the network.
  - Enables networks to support a wide range of application requirements.
- Not an absolute principle: e.g., performance is often an exception.
- Fine to implement functionality in lower layer assuming:
  - Higher layers don't need to replicate functionality.
  - Leads to improved performance.
  - Does not place an impedement to other uses of the system.