Lecture 4

- How is communication architected?
- The network layer

Internet Layers

- Applications
- Reliable (or unreliable) transport
- Best-effort global packet delivery
- Best-effort local packet delivery
- Physical transfer of bits

What gets implemented at the end systems?

- Bits arrive on wire, must make it up to application
- Therefore, all layers exits at host!

What gets implemented in the network?

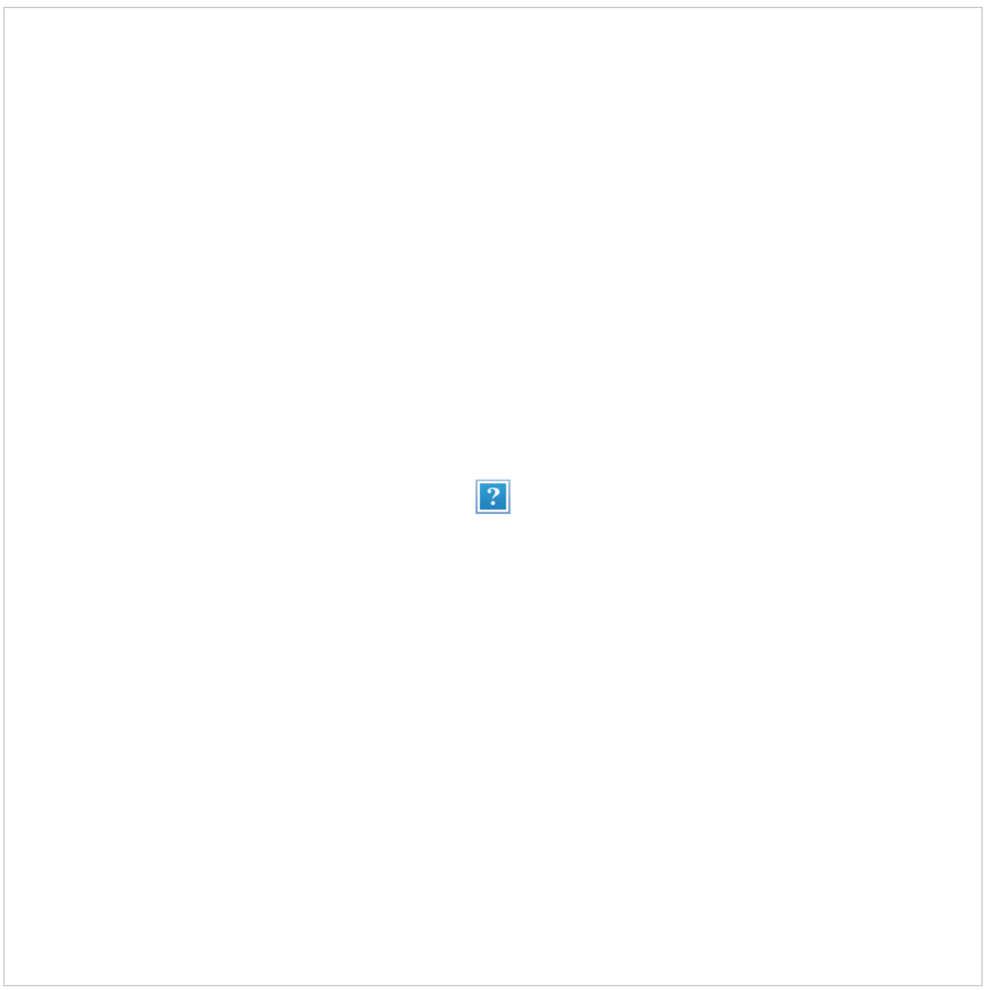
- Bits arrive on wire -> physical layer (L1)
- Packets must be delivered across links and local networks -> datalink layer (L2)
- Packets must be delivered between networks for global delivery -> network layer (L3)
- The network does not support reliable delivery

- Transport layer (and above) **not** supported

- Hence:
    - **Switches**: Implement physical and datalink layers (L1, L2)
    - **Routers**: Implement physical, datalink, network layers (L1, L2, L3)

Simple Diagram

- Lower three layers implemented everywhere
- Top two layers implemented only at hosts



A closer look: end-system

- Application
    - Web server, browser, mail, game
- Transport and network layer
    - Typically part of the operating system
- Datalink and physical layer
    - Hardware/firmware/drivers

Switches vs. Routers

- Switches do what routers do but don’t participate in global delivery, just local delivery
    - Switches only need to support L1, L2
    - Routers support L1-L3
- Won’t focus on the router/switch distinction
    - When I say switch, I almost always mean router
    - Almost all boxes support network layer these days

Why layers?

- Reduce complexity
- Improve flexibility

Why not?

- Sub-optimal performance
- Cross-layer information often useful
    - Several “layer violations” in practice

Architectural Wisdom

- Benefits of layering
    - Reduce complexity, increase flexibility
- “Narrow waist”
    - Simple, minimal requirements for interoperability
- “Smart ends, dumb network”
    - No application knowledge in network -> more general
    - Minimal state in the network -> more robust to failure

End-to-end argument: Intuition

- Some application requirements can only be correctly implemented **end-to-end**
    - Reliability, security, etc.
- Implementing these in the network is hard
    - Every step along the way must be fail proof
- End-systems
    - **Can** satisfy the requirement without network’s help
    - **Will/must** do so, since they can’t rely on the network

---

Recap

- Implementing functionality (e.g., reliability) in the network
    - Doesn’t reduce host implementation complexity

    - Does increase netork complexity
    - Decreases generality (of the network)
- However, implementing functionality in the network can improve performance in some cases
    - e.g., consider a very lossy link

Commonly used examples

- Error handling in file transfer
- End-to-end, versus in-network encryption
- The partition of work between TCP and IP for reliable packet delivery
- What about Quality of Service (QoS)?
    - Communication throughput or delay guarantee

Some consequences

- In layered design, the E2E principle provides guidance on which layers are implemented where
- “Dumb” network and “smart” end systems
    - Often credited as key to the Internet’s success
- “Fate sharing”
    - Store state at the system entities that rely on the state
    - Often translated to keeping state out of routers

Cracks in the E2E arguments

- Ignored incentives of different stakeholders
    - e.g., ISP looking for new revenue-generating services
    - e.g., users looking for a performance boost

- Sometimes we don’t trust the end-systems to do the job
    - e.g., firewalls, intrusion detection systems

Recap: architectural decisions

- How to break system into modules
    - Layering
- Where are modules implemented?
    - End-to-end principle
- Where is state stored?
    - Fate-sharing

Taking Stock

- Basic concepts
    - Components: end-systems, links, switches, routers
    - Performance: delay, throughput
- Basic design choices
    - Packet vs. circuit switching
    - Best-effort vs. guaranteed service
    - Layering
    - “Dumb” network, “smart” ends

The Network Layer

```
Addressing
Forwarding
Routing
```

Addressing (for now)

- Assume each end-system has a unique address
- No particular structure to these addresses
- Will cover IP addresses later in the course

Forwarding

- **Local** router process that determines the output link (a.k.a. “next hop”) for each packet
- How
    - Read address from packet’s network layer header
    - Search forwarding table

Routing

- **Network-wide** process that determines the content of forwarding tables
    - Determines the end-to-end path for each destination
- How
    - Coming up soon

Forwarding vs. Routing

- Forwarding: “data plane”
    - Directing one data packet
    - Each router using local routing state
- Routing: “control plane”
    - Computing the forwarding tables that guide packets
    - Jointly computed by routers using a distributed algorithm
- Very different timescales!

Routing Fundamentals

- **Validity** of routing state

Goal (v1)

- Find a path to a given destination
- How do we know that the state contained in forwarding tables meets our goal?
    - This is what “validity” of routing state tells us
    - [This is non-standard terminology]

Local vs. Global View of State

- *Local* routing state is the forwarding table in a single router
    - By itself, the state in a single router can’t be evaluated
    - It must be evaluated in terms of the global context
- *Global* state refers to the collection of forwarding tables in each of the

routers

    - Global state determines which paths packets take

"Valid" Routing State

- Global state is "valid" if it produces forwarding decisions that always deliver packets to their destinations
- Goal of routing protocols: computer valid state
    - But how can you tell if routing state if valid?
- Need a succinct correctness condition for routing

Necessary and Sufficient Condition

- Global routing state is valid **if and only if**:
    - There are no dead ends (other than destination)
    - There are no loops
- A **dead end** is when there is no outgoing link (next-hop)
    - A packet arrives, but the forwarding decision does not yield any outgoing link
- A **loop** is when a packet cycles around the same set of nodes forever

Necessary ("only if"): Easy

- If you run into a dead-end before hitting destination, you'll never reach the destination
- If you run into a loop, you'll never reach destination

Sufficient ("if")

- Assume no dead-ends, no loops
- Packet must keep wandering, but without repeating
    - If ever enter same switch from same link, will loop
- Only a finite number of possible links for it to visit
    - It cannot keep wandering forever without looping
    - Must eventually hit destination

Checking Validity of Routing State

- Focus only on a single destination
    - Ignore all other routing state
- Mark outgoing link ("next hop") with arrow)
    - There is only one at each node
- Eliminate all links with no arrows
- Look at what's left

Lesson…

- Very easy to check validity of routing state for a particular destination
- Dead-ends are obvious
    - Node without ongoing arrow
- Loops are obvious
    - Disconnected from rest of graph

Goal (v2)

- Find a **least cost path** to a given destination