End-to-End Transport Over Wireless I: Preliminaries, Split Connection

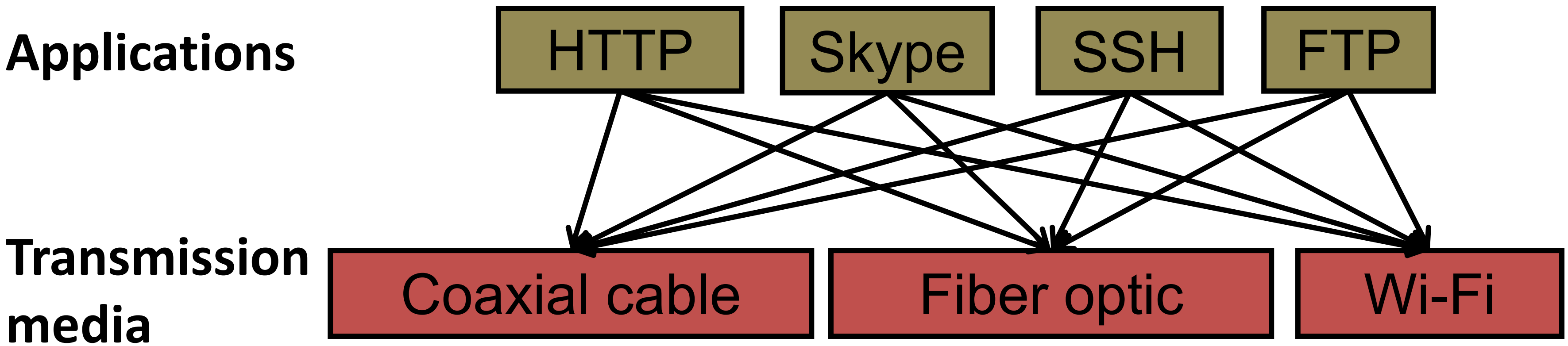
COS 463: Wireless Networks
Lecture 2
Kyle Jamieson

[Various parts adapted from S. Das, B. Karp, N. Vaidya]

Today

1. **Layering** and the End-to-End Argument
2. Transmission Control Protocol (TCP) primer
3. Split Connection TCP over wireless

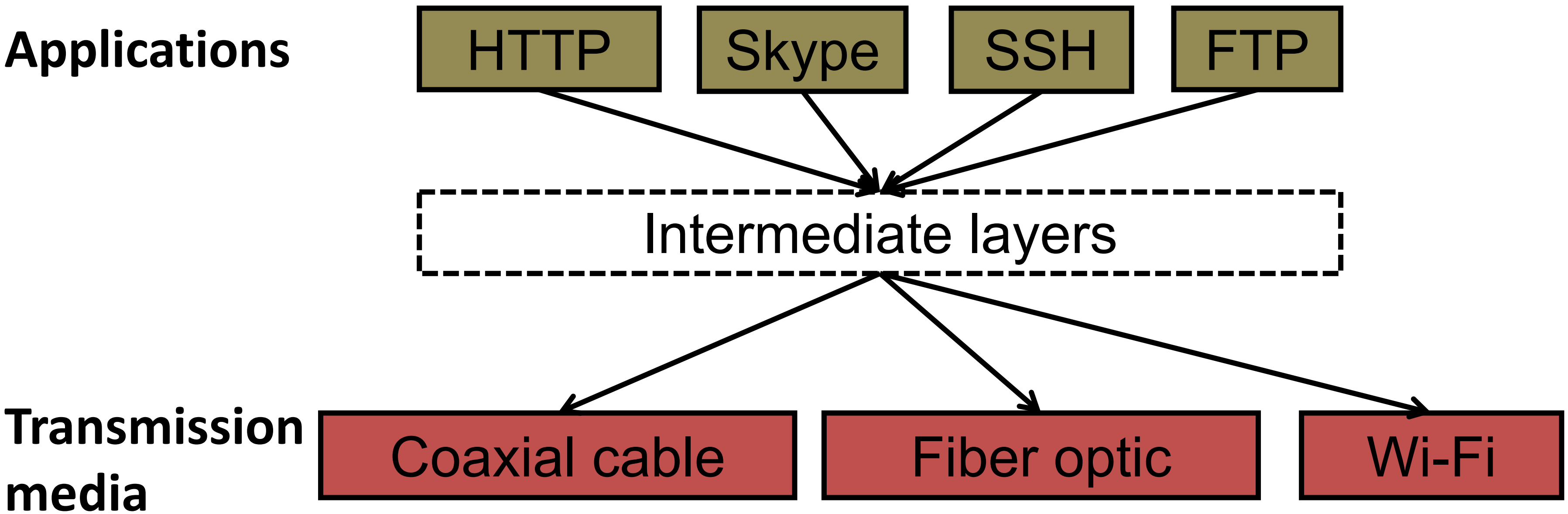
Layering: Motivation



- Re-implement every application for every new underlying transmission medium?
- Change every application on any change to an underlying transmission medium (and vice-versa)?

- **No!** But how does the Internet design avoid this?

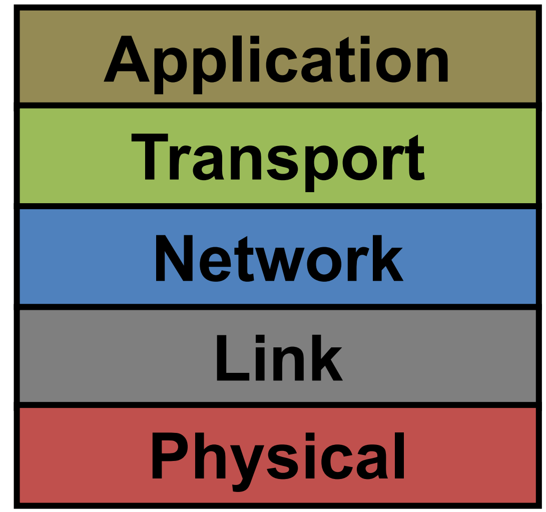
Internet solution: Intermediate layers



- Intermediate layers provide a set of abstractions for applications and media
- New applications or media need only implement for intermediate layer’s interface

Properties of layers

- **Service:** **What** a layer does
- **Service interface:** **How to access** the service
  - Interface for the layer **above**
- **Protocol interface:** **How peers communicate** to implement service
  - Set of rules and formats that govern the communication **between two Internet hosts**



Physical layer (L1)

- **Service**: **Move bits** between two systems connected by a **single physical link**

- **Interface**: specifies **how to send, receive bits**
  - *e.g.*, require quantities and timing

- **Protocols**: coding scheme used to represent bits, voltage levels, duration of a bit

Data link layer (L2)

- **Service**: **End hosts exchange atomic messages**
  - Perhaps over multiple physical links
    - But using same ***framing*** (headers/trailers)
  - **Arbitrates access** to common physical media
  - Implements **reliable transmission, flow control**
- **Interface**: send messages (frames) to other end hosts; receive messages addressed to end host
- **Protocols**: Addressing, routing, medium access control

Network layer (L3)

- **Service**: **Deliver *datagrams* to other networks**
  - Cross-technology (*e.g.*, Ethernet, 802.11, optical, …)
  - Possibly includes packet scheduling/priority
  - Possibly includes buffer management
  - **Best effort:** may **drop, delay, duplicate** datagrams

- **Interface**:
  - Send packets to specified internetwork destination
  - Receive packets destined for end host

- **Protocols**:
  - Define inter-network addresses (globally unique)
  - Construct routing tables and forward datagrams

Transport layer (L4)

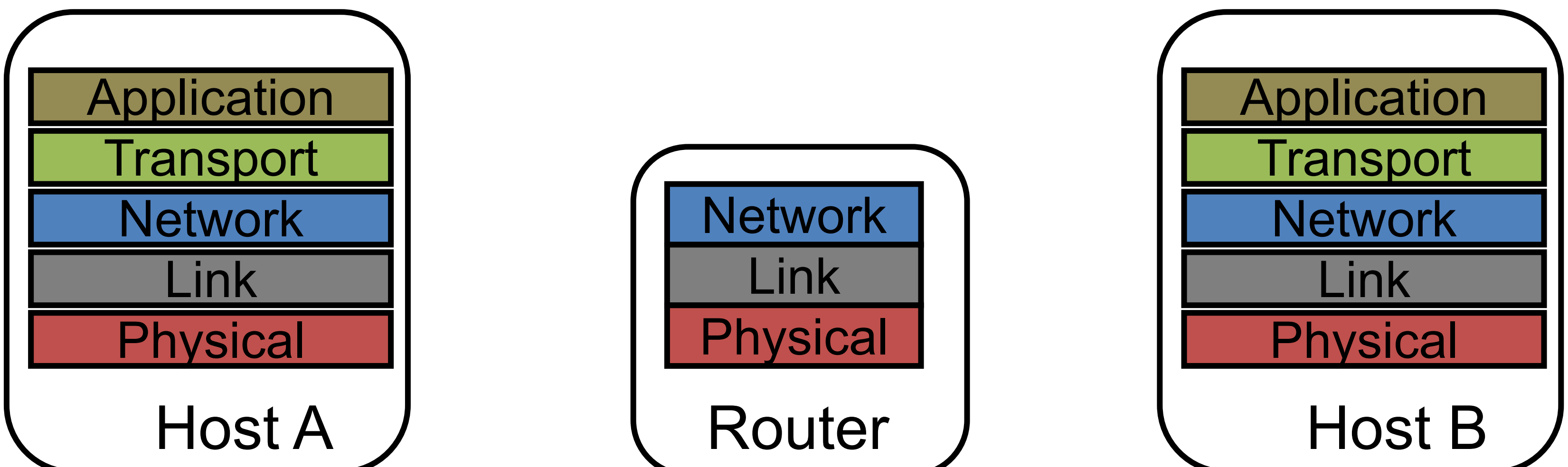
- **Service**: Provide **end-to-end** communication between **processes on different hosts**
  – Demultiplex communication between hosts
  – Possibly reliability in the presence of errors
  – Rate adaptation (**flow control, congestion control**)

- **Interface**: send message to specific process at given destination; local process receives messages sent to it

- **Protocol**: perhaps implement reliability, flow control, packetization of large messages, framing

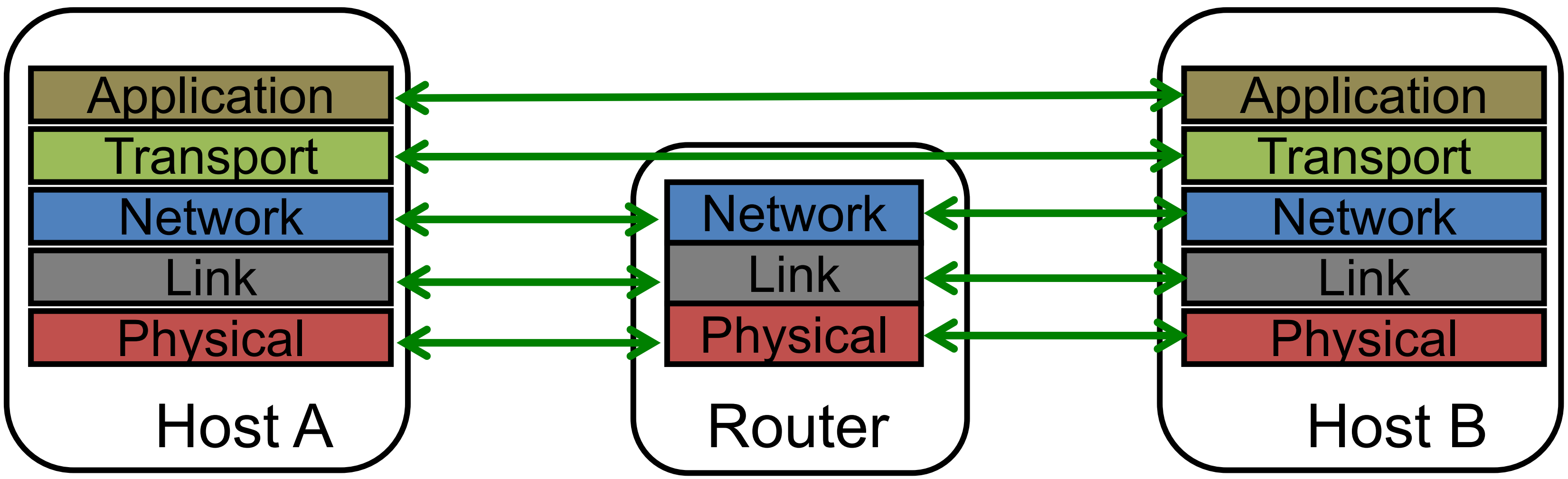
Who does what?

- Five layers
  - **Lower three layers** are implemented **everywhere**
  - **Top two layers** are implemented **only at end hosts**
    - **Their protocols are *end-to-end***



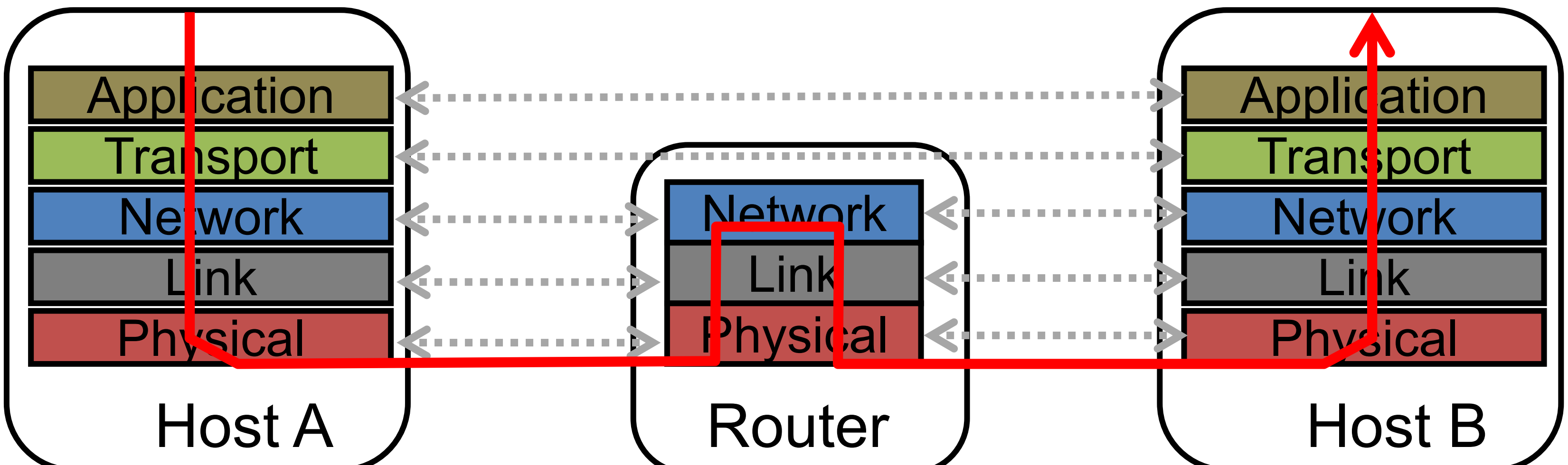
Logical communication

- Each layer on a host interacts with its **peer** host's corresponding layer via the **protocol interface**



Physical path across the Internet

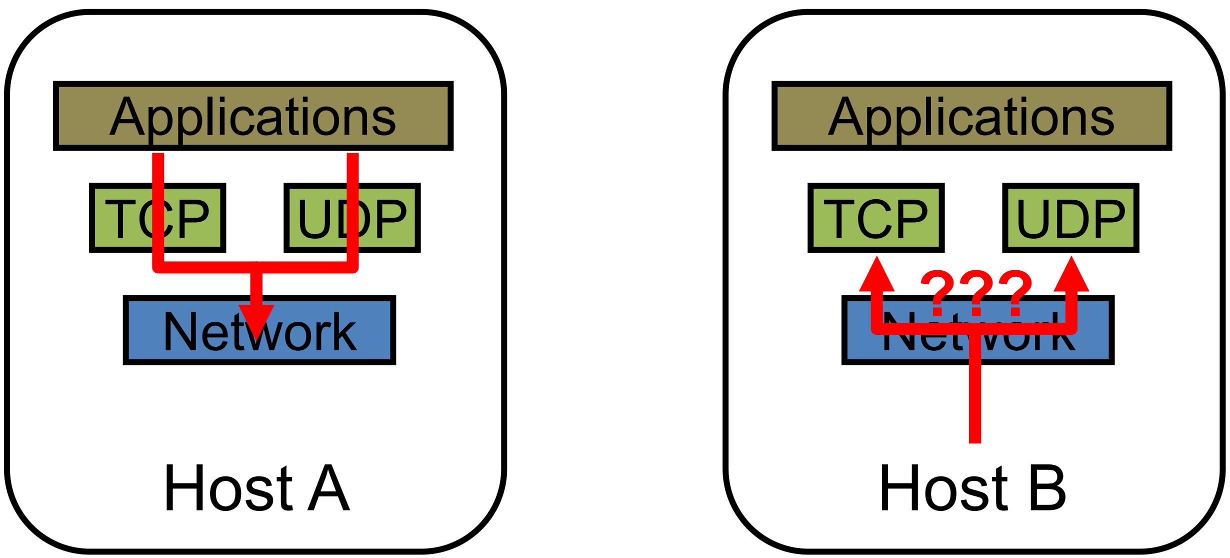
- Communication **goes down to physical network**
- Then from **network peer to peer**
- Then **up to the relevant layer**



Protocol multiplexing

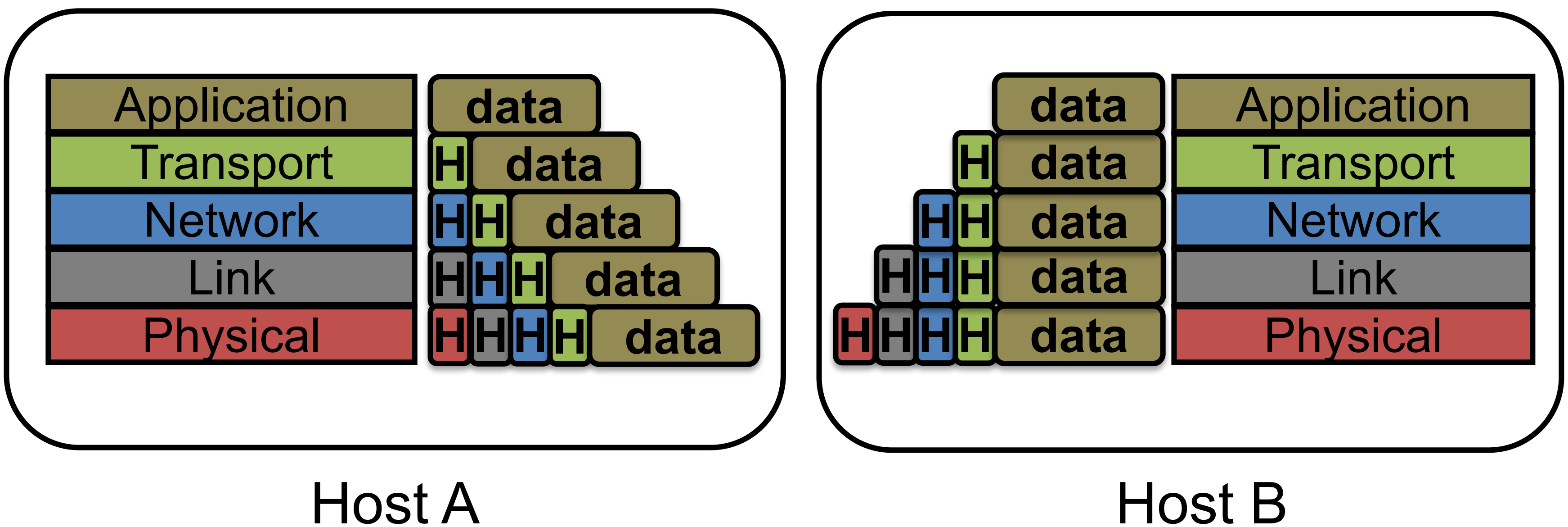
- **Multiplexing:** Multiple **overlying** protocols share use of a single **underlying** protocol

- **Problem:** How does the underlying protocol decide **which overlying protocol** messages go to?

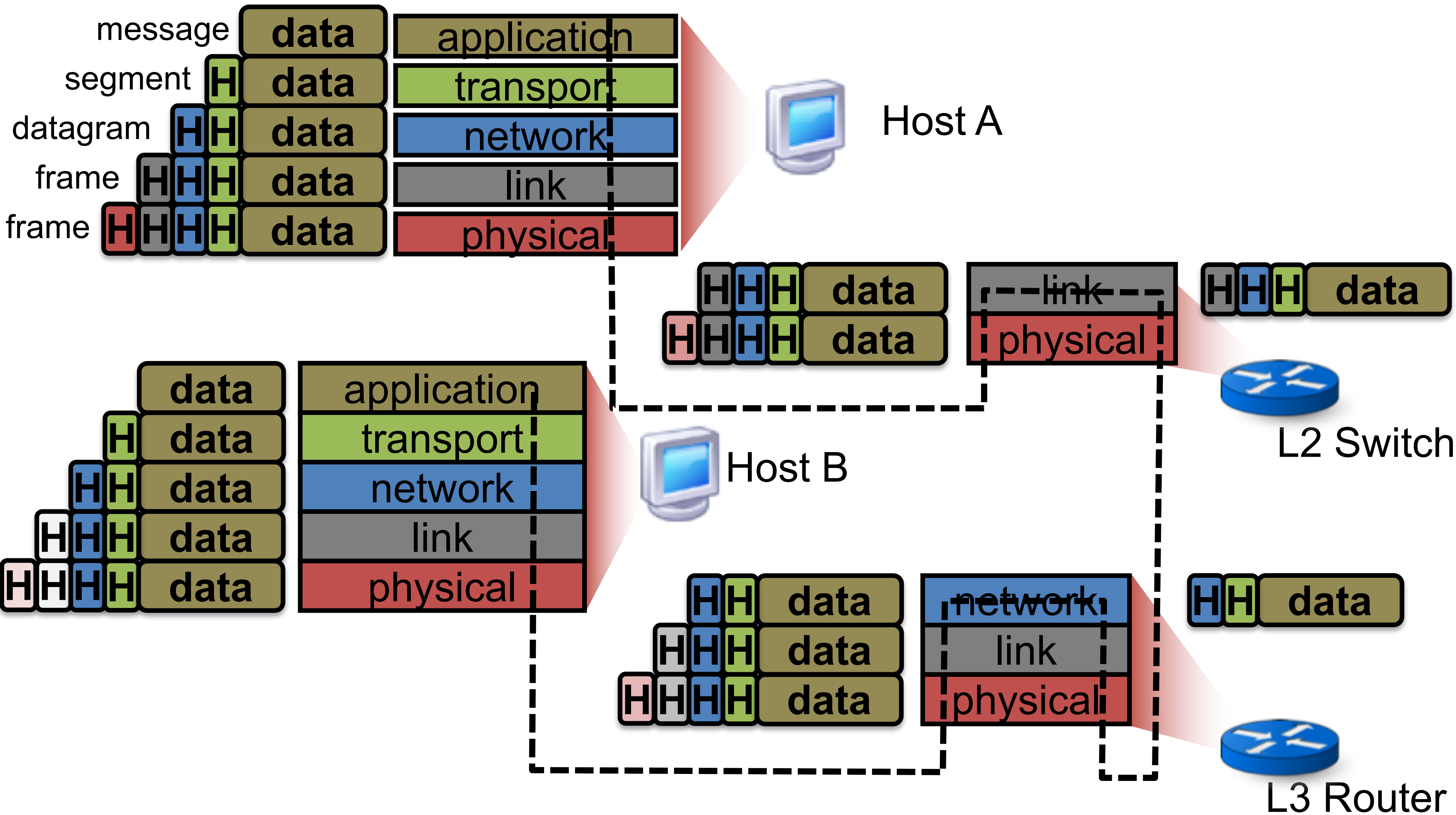


Protocol headers

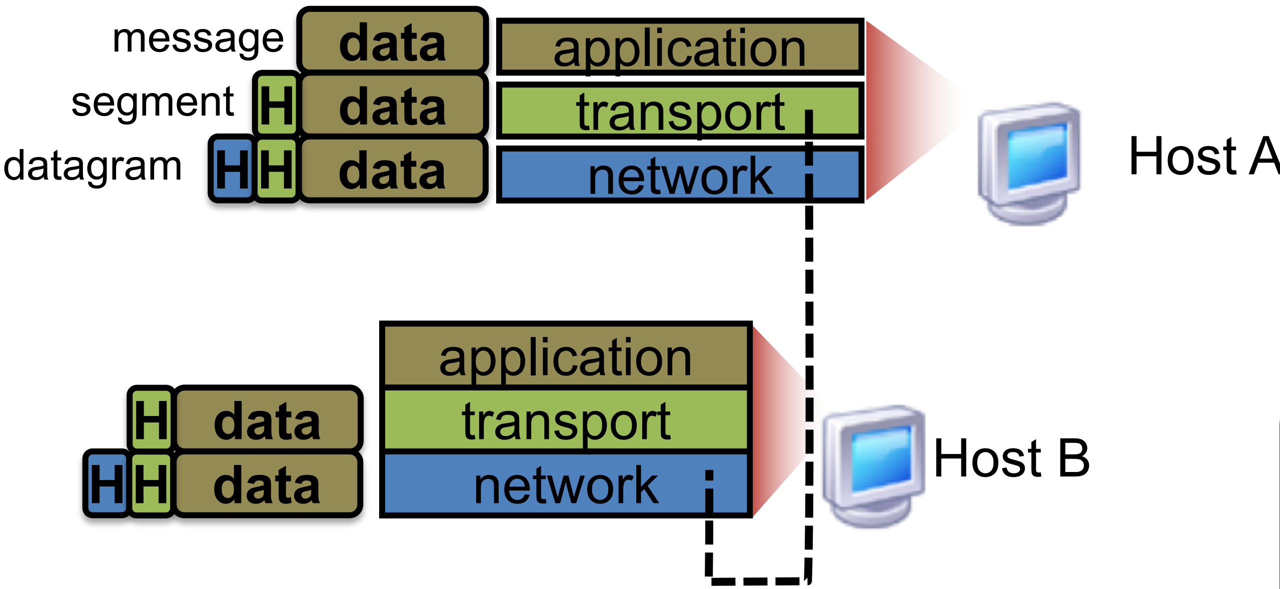
- Each layer attaches its own header (H) to facilitate communication between peer protocols
- On reception, layer **inspects and removes** its own header
  – Higher layers **don't see** lower layers' headers

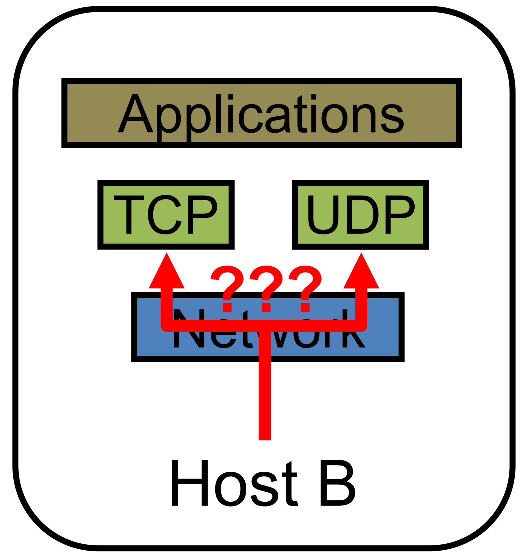


Encapsulation in the Internet



Protocol demultiplexing





- Lower-layer header contains demultiplexing information
- **Network header** contains **Protocol** field specifying overlying protocol

Drawbacks of layering

- Layer *n* may **duplicate** lower level functionality
  - *e.g.*, error recovery to retransmit lost data

- Layers may need **same information in headers**
  - *e.g.*, timestamps, maximum transmission unit size

- Layering can **hurt performance**
  - *e.g.*, previous lecture

Layer violations

- Two types:

1. **Overlying** layer examines **underlying** layer's state
   - *e.g.*, transport monitors wireless link-layer to see whether packet loss from congestion or corruption
2. **Underlying** layer inspecting **overlying** layer's state
   - *e.g.*, firewalls, NATs (network address translators), "transparent proxies"

Today

1. Layering and **the End-to-End Argument**

2. Transmission Control Protocol (TCP) primer

3. Split Connection TCP over wireless

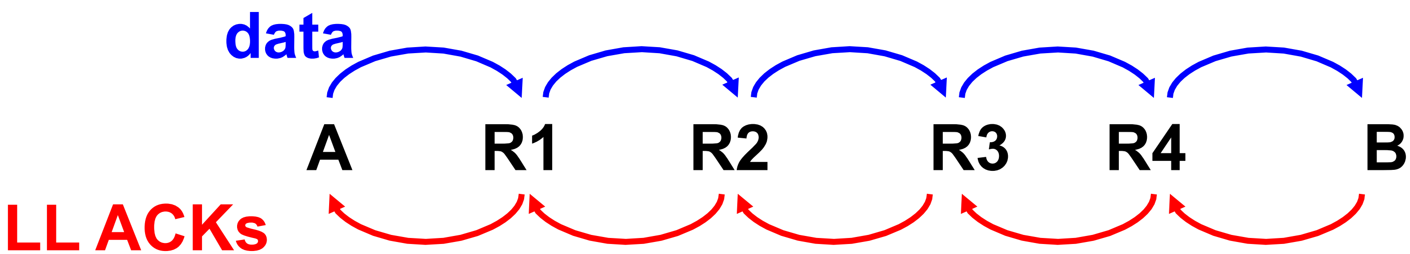
Motivation: End-to-End Argument

- **Five layers** in the Internet architecture model

- **Five places** to solve many of same problems:
  - In-order delivery
  - Duplicate-free delivery
  - Reliable delivery after corruption, loss
  - Encryption
  - Authentication

- ***In which layer(s) should a particular function be implemented?***

Example: Careful file transfer from A to B



- **Goal: Accurately copy file on A's disk to B's disk**

- Straw man design:
  - Read file from **A**'s disk
  - **A** sends stream of packets containing file data to **B**
    - L2 retransmission of lost or corrupted packets at each hop
  - **B** writes file data to disk

- ***Does this system meet the design goal?***
  - Bit errors on links not a problem

Where might errors happen?

- On **A**'s or **B**'s disk
- In **A**'s or **B**'s RAM or CPU
- In **A**'s or **B**'s software
- In the RAM, CPU, or **software** of **any** **<u>router</u>** that forwards packet

- Why might errors be **likely?**
  - Drive for CPU speed and storage density: pushes hardware to EE limits, engineered to tight tolerances
    - *e.g.*, today's disks return data that are the output of an maximum-likelihood estimation!
  - Bugs abound!

Solution: End-to-End verification

1. **A** keeps a **checksum** with the on-disk data
   – *Why not compute checksum at start of transfer?*
2. **B** computes checksum over received data, sends to **A**
3. **A** compares the two checksums and resends if not equal

- Can we eliminate hop-by-hop error detection?

- Is a whole-file checksum, **alone,** enough?

End-to-End Principle

- Only the **application at communication endpoints can completely and correctly** implement a function

- Processing in **middle alone cannot** provide function
  - Processing in **middle may,** however, be an **important performance optimization**

- Engineering middle hops to provide guaranteed functionality is *often* **wasteful of effort, inefficient**

Perils of lower-layer implementation

- **Entangles** application behavior with network internals

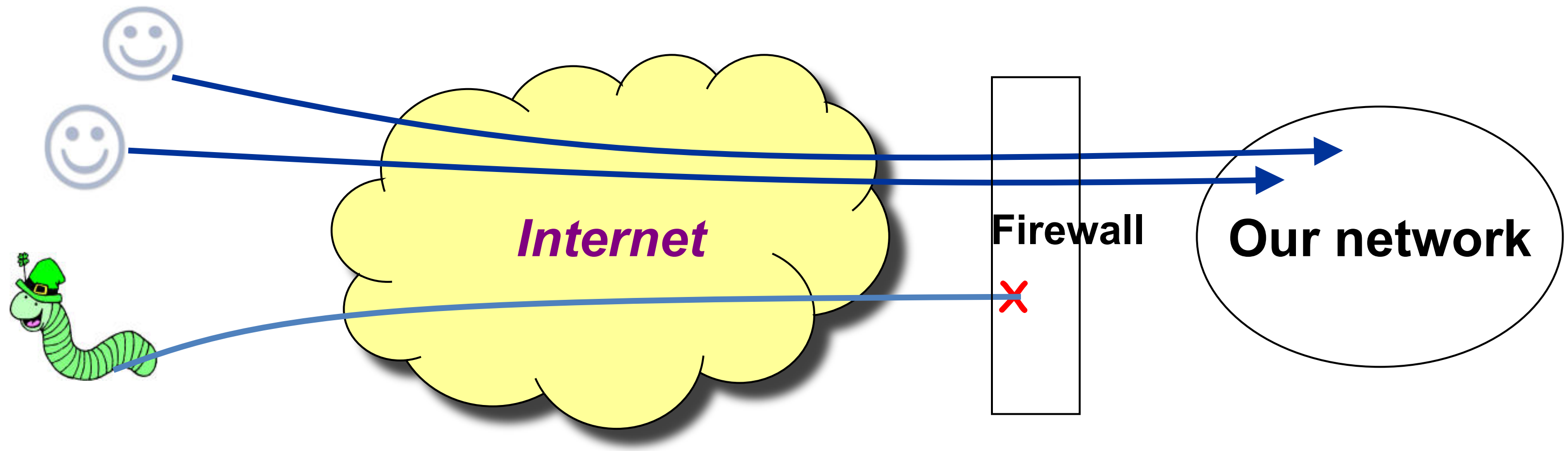
- **Suppose** each IP router **reliably transmitted** to next hop
  - **Result:** Lossless delivery, but **variable delay**
    - ftp: **Okay,** move huge file reliably (just end-to-end TCP works fine, too, though)
    - Skype: **Terrible,** jitter packets when a few corruptions or drops not a problem anyway

- **Complicates deployment** of innovative applications
  - Example: Phone network *v.* the Internet

Advantages of lower-layer implementation

- Can improve **end-to-end system performance**
- Each application author **needn't recode a shared function**
- Overlapping error checks (*e.g.*, checksums) at all layers invaluable in **debugging and fault diagnosis**
- If end systems not cooperative (increasingly the case), **only way to enforce resource allocation!**

End-to-end violation: Firewalls



- Firewalls clearly **violate the e2e principle**
  - **Endpoints** are capable of deciding what traffic to ignore
  - Firewall **entangled** with design of network and higher protocol layers and apps, and vice-versa
    - ***e.g.:*** New ECN bit to improve TCP (wireless) congestion control; many firewalls **filter all such packets!**

- **Yet, we probably do need firewalls**

Summary: End-to-End principle

- Many functions **must** be implemented **at application endpoints** to provide desired behavior
  - Even if implemented in “middle” of network

- End-to-end approach **decouples design** of components in network interior from design of applications at edges
  - Some functions still **benefit** from implementation in **network interior** at cost of entangling interior, edges

- End-to-end principle is **not sacred;** it’s just a way to think critically about design choices in communication systems

Today

1. Layering and the End-to-End Argument

2. **Transmission Control Protocol (TCP) primer**

3. Split Connection TCP over wireless

TCP: Connection-Oriented, Reliable Byte Stream Transport

- Layer-four protocol for reliable transport
  - Sending app offers a sequence of bytes: d0, d1, d2, …
  - Receiving app sees all bytes arrive in same sequence: d0, d1, d2…
    - **Result: Reliable byte stream** transport **between endpoints** on the internet
- Each such byte stream is called a ***connection***, or ***flow***

TCP’s Many End-to-End Goals

- Recover from **data loss**
- Avoid receipt of **duplicated** data
- Preserve data **ordering**
- Provide **integrity** against corruption
- Avoid sending faster than **receiver** can **accept** data
- **Avoid congesting** network

Fundamental Problem: Ensuring At-Least-Once Delivery

- Network **drops** packets, so to **ensure delivery:**
  - Sender attaches ***sequence number (seqno)*** to each data packet sent; keeps copy of sent packet
  - Receiver returns ***acknowledgement (ACK)*** to sender for each data packet received, containing seqno

- **Sender** sets a ***retransmit timer*** on each transmission
  - If timer expires < ACK returns: **retransmit** that packet
  - If ACK returns, **cancel timer, forget** that packet

- How long should the retransmit timer be?

Fundamental Problem: Estimating RTT

- Expected time for ACK to return is *round-trip time* (*RTT*)
  - End-to-end delay for data to reach receiver, then its ACK to reach sender

- **Strawman:** use fixed timer (*e.g.,* 250 milliseconds)
  - What if the route/wireless conditions change?
  - W

Fixed timer violates E2E argument; details of link behavior should be left to link layer!

Hard-coded timers lead to brittle behavior as technology evolves

- **Too small** a value: needless **retransmissions**
- **Too large** a value: needless **delay detecting loss**

Estimating RTT: Exponentially Weighted Moving Average (EWMA)

- Measurements of RTT readily available
  - Note time ***t*** when packet sent, corresponding ACK returns at time ***t'***
  - **RTT *measurement sample:* m** = t'− t

- Single sa… …ic)

EWMA weights newest samples most

How to choose α? (TCP uses 1/8)

Is mean sufficient to capture RTT behavior over time? (more later)

- Adapt over time, using EWMA:
  - **Measurement samples:** m0, m1, m2, …
  - fractional weight for new measurement, **α**
  - $RTT_i = ((1 - \alpha) \times RTT_{i-1} + \alpha \times m_i)$

How does TCP know congestion has occurred?

- Packet loss; binary signal

- How does TCP know that a packet loss has occurred?
  - Lack of Acknowledgements → Timeouts

- How can packets get lost in wired networks?
  - Buffer overflows

Retransmission and Duplicate Delivery

- When sender's retransmit timer expires, two **indistinguishable cases:**
  - **Data packet dropped** *en route* to receiver, or
  - **ACK dropped** *en route* to sender

- In both cases, **sender retransmits**

- In latter case, **duplicate data packet reaches receiver!**

Eliminating Duplicates: Exactly-Once Delivery

- Sender marks each packet with a **monotonically increasing sequence number seqno**
- Sender includes greatest ACKed seqno in its packets
- Receiver remembers only greatest received sequence number, drops received packets with smaller ones

Doesn't guarantee delivery!
Properties: If delivered, then only once.
If undelivered, sender will not think delivered.
If ACK not seen, data may have been delivered, but sender will not know.

End-to-End Integrity

- Achieved by using **transport checksum**
- Protects against things link-layer reliability cannot:
  - Router memory corruption, software bugs, &c.

- Covers **data in packet, transport protocol header**

- Also should cover **layer-3 source and destination!**
  - Misdelivered packet should not be inserted into data stream at receiver, nor should be acknowledged
  - Receiver drops packets w/failed transport checksum

Today

1. Networking primer/review
2. Transmission Control Protocol (TCP) primer
3. **TCP over wireless**

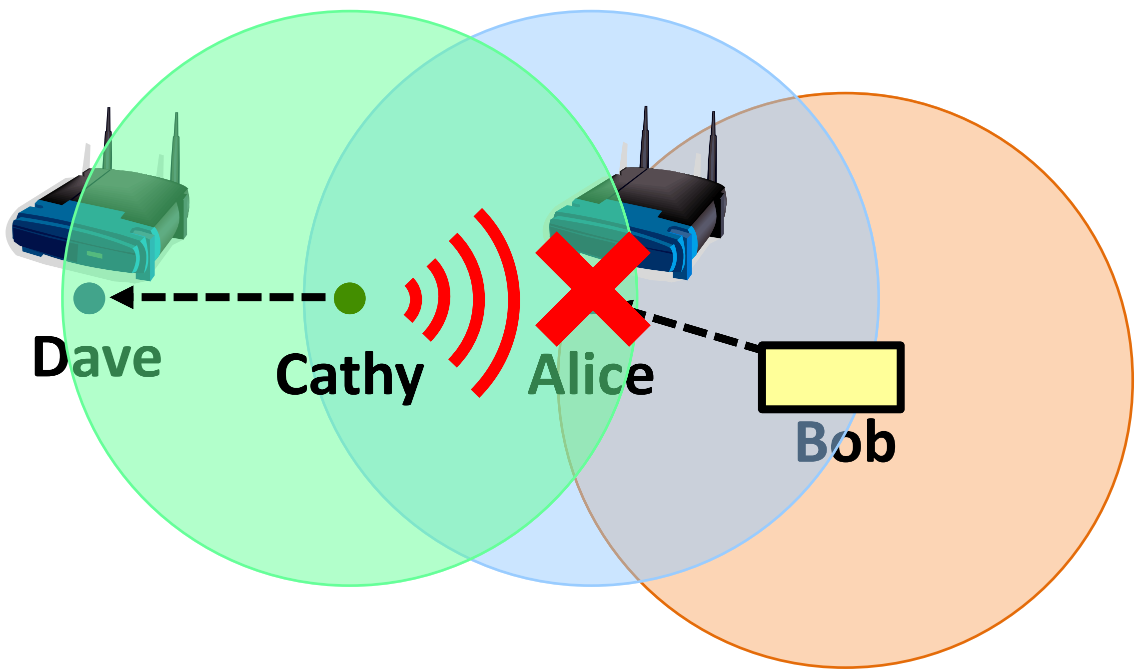
Running TCP on Wireless Links

- Generally, TCP interprets any **packet loss** as a **sign of queue congestion**
  - TCP sender **reduces congestion window**

- Wireless links operate at **higher** bit error rates and frame loss rates

- On **wireless links**, packet loss can also occur due to random channel errors, or cellular or WLAN handoffs
  - Temporary loss **not due to congestion**
  - Reducing window **may be too conservative**
  - Leads to **poor throughput**

Wireless can be congested, too



Shared wireless medium leads to a *collision* of Bob and Cathy's packets *at* Alice

Wireless: Best sender strategy becomes unclear

	Congestion loss	Link loss
Wireless link:	Frequent (collision)	Frequent (multipath, interference)
Wired links:	Frequent (queue drop)	Rare

Slow down! **Maintain rate**

Fundamental question:

How to differentiate between

1. Loss due to **congestion**
2. Loss due to **wireless link itself**

Hard to do:

TCP is fundamentally an “end-to-end” protocol: only sees a loss

Two Broad Approaches

1. **Mask wireless losses from TCP sender**
   - Then TCP sender will not slow down

   - **Split Connection Approach**
   - TCP Snoop

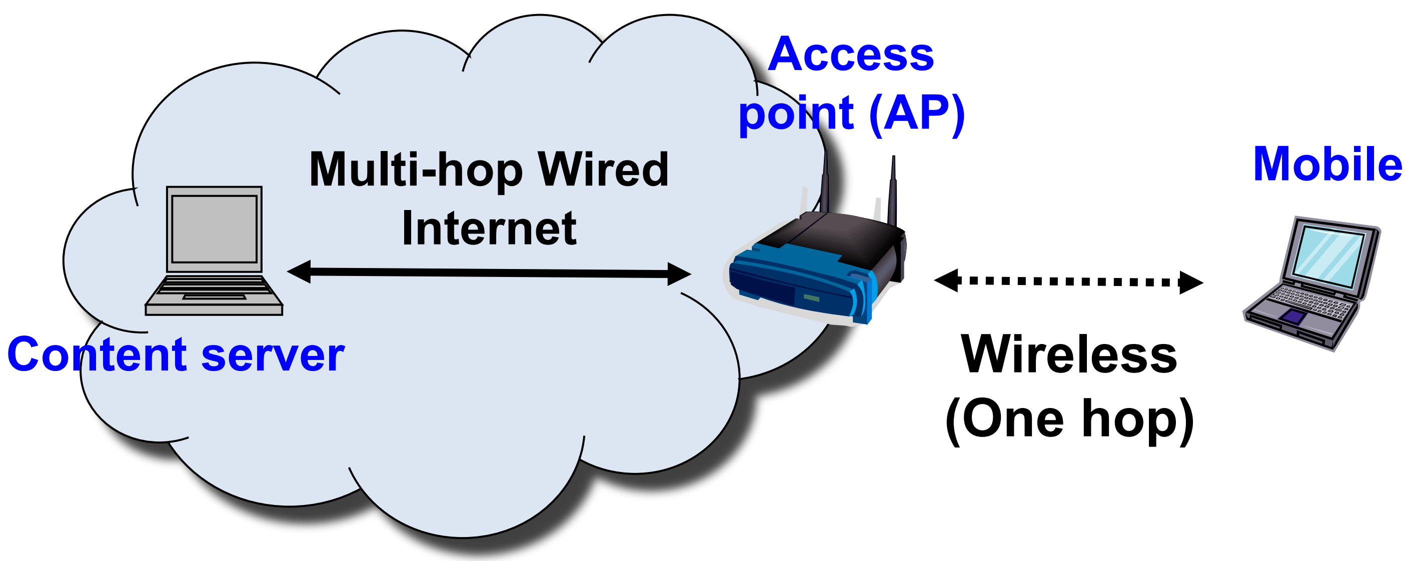
2. Explicitly notify TCP sender about cause of packet loss

Split Connection Approach

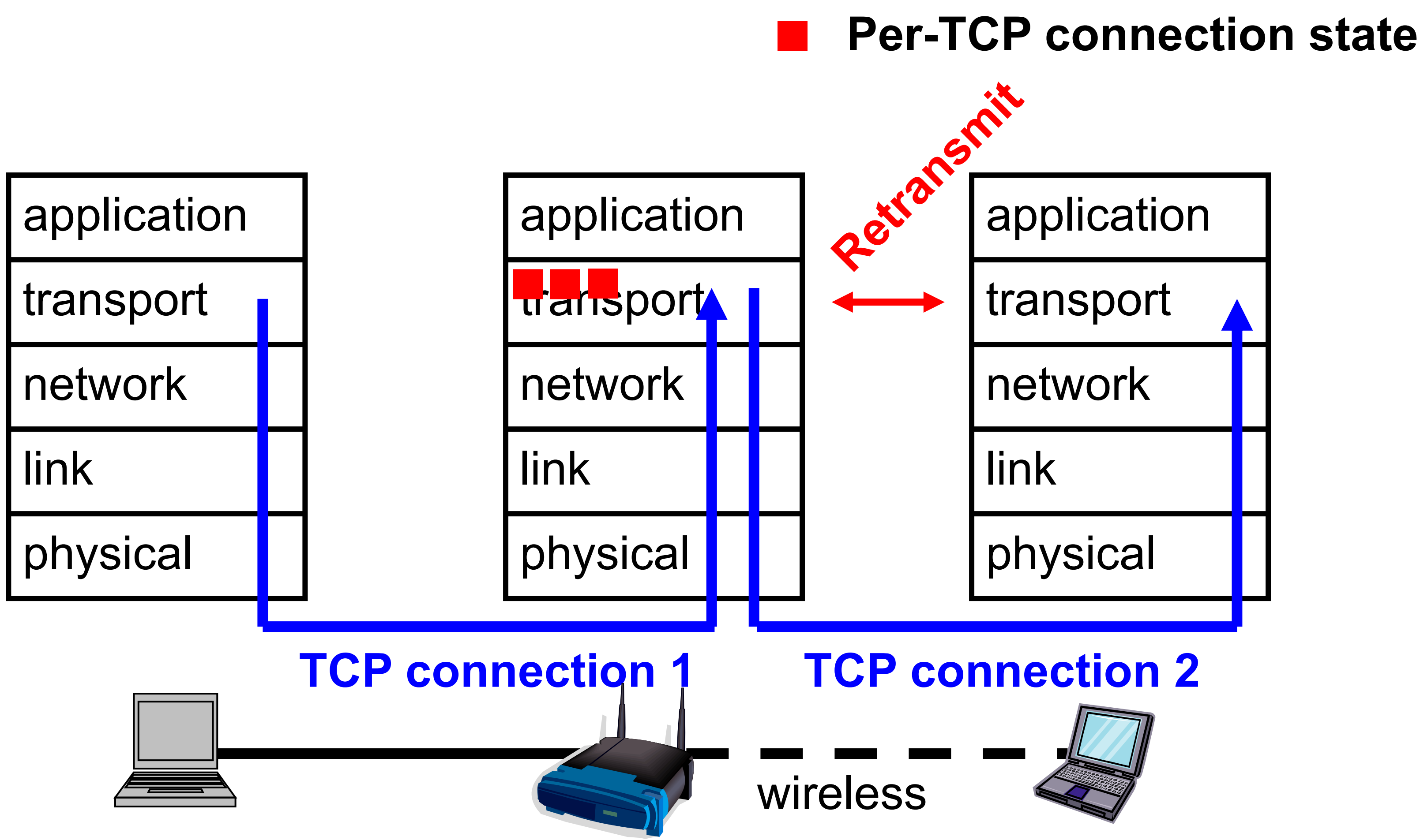
- Also called Indirect TCP (I-TCP)

- Segment the TCP connection into two parts:
  1. TCP connection between content server and AP
  2. Another connection between AP and mobile host
  – **No real end-to-end connection**

- **No changes** to the TCP endpoint at the content server



Split Connection: TCP Implementation

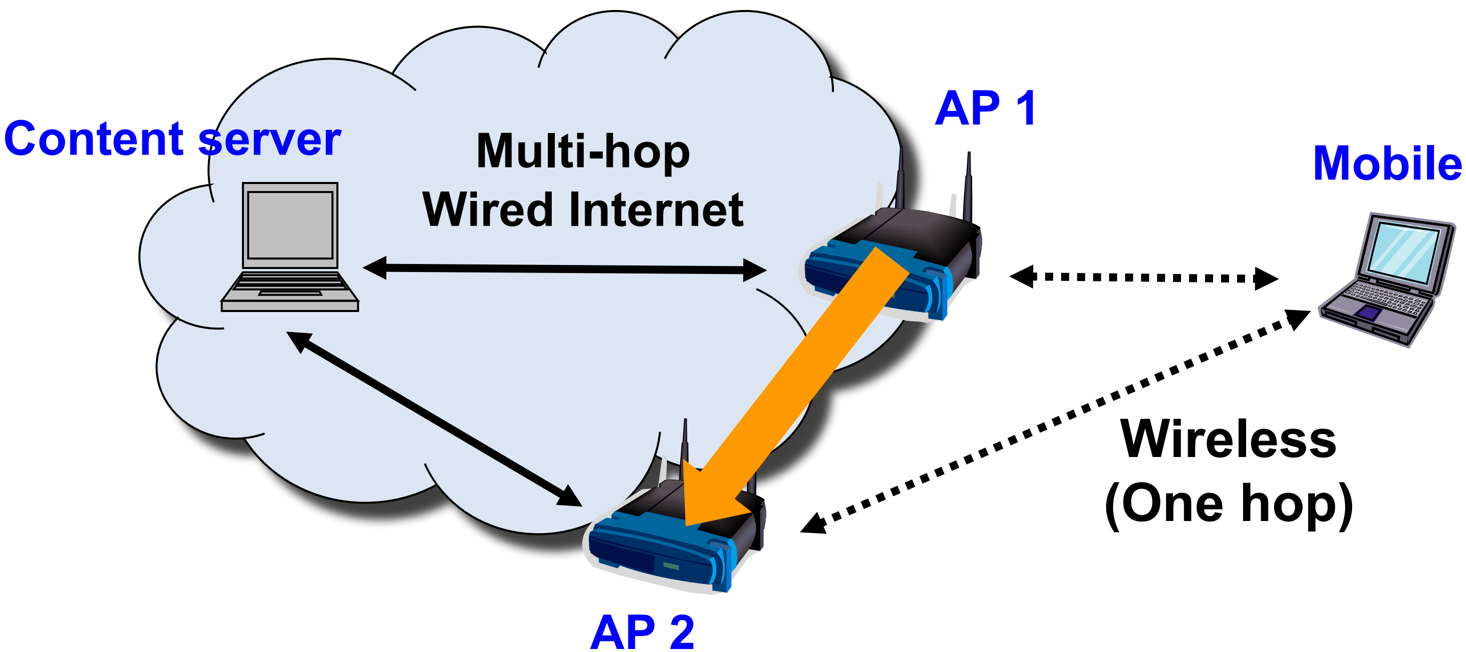


Split Connection: Considerations

- Connection between AP and mobile need not be TCP
  - Could be *e.g.*, Selective Repeat over UDP
- Assume that the **wireless part is just one hop** (traditional cellular or wireless LAN)
- Wireless losses **assumed not caused by congestion**
  - Not true always (*e.g.* collisions): Sender **should slow down, but doesn't**

Split Connection Socket and State Migration

- Consequence of breaking end-to-end connection:
  - On handoff from AP 1 to AP 2, **connection state** must **move from AP 1 to AP 2**



Split Connection: Advantages

- **No changes needed** in wired network or content servers

- Transmission errors on the wireless link do not propagate into the fixed network
  - Local recovery from errors

- Possibility of using custom (optimized) transport protocol for the hop between AP and mobile

Split Connection: Critique

- **Loss of end-to-end semantics:**
  - ACK at TCP sender no longer means that receiver must have received that packet
    - TCP **no longer reliable** if **crash/bug at AP**
- **Large buffer space** may be needed at AP
- AP must maintain **per-TCP connection state**
- **State must be forwarded** to new AP on handoff
  - May cause higher handoff latency

Friday Precept

Python Intro, Signal Processing Primer,
Lab 1/Part 0 Intro

Location: 87 Prospect Street, Room 065

Tuesday

Transport over Wireless I: Snoop
and Explicit Loss Notification