COS 461/561: Computer Networks
Class Meeting, Lectures 1 & 2

Kyle Jamieson (461), Ravi Netravali (561)

Spring 2023

Class Meeting: Tue 10:00-10:50 AM

www.cs.princeton.edu/courses/archive/spr23/cos461

Today

1. Course Introduction and Policies (461 & 561)

2. Internet Basic Principles

3. Link Layer

461: What You Learn in This Course

- **Knowledge:** how the Internet works, and why
  - Protocol stack: link, network, transport, application
  - Resource allocation: congestion control, routing
  - Applications: Web, P2P, …
  - Networks: enterprise, cloud, backbone, wireless, …
- **Insight:** key concepts in networking
  - Naming, layering, protocols, resource allocation, …
- **Skill:** network programming
  - Many nodes are general-purpose computers
  - Can innovate and develop new uses of networks

561: What You Learn in This Course

- **Knowledge:** how the Internet works, and why
- **Insight:** key concepts and state of the art in networking
  - Naming, layering, protocols, resource allocation, …
  - Discuss classic & state of the art networking research papers, in depth. Tied to lecture topics in 461
- ~~**Skill: network programming**~~
- **Skill:** network research
  - Semester systems-building/research project, in groups
  - Reproduce a result (more common), or build a novel project

Learning the Material: 461 People

- Class Meeting: Prof Kyle Jamieson
  - Tue 10:00 – 10:50 AM
  - OH: Thursday 9:00-10:00 AM

- Precepts: TAs Murali Ramanujam, Ryan Torok
  - Ryan OH: TBA
  - Murali OH: Friday 9:00-10:00 AM (location TBA)

Learning the Material: 561 People

- Precepts: Prof Ravi Netravali
  - Friday: 10:00 – 11:30 AM
  - Room: Friend 008

- Precept TA: Mike Wong

Learning the Material: 461 & 561 Class Meetings

- Class Meetings: Tuesdays 10:00 – 10:50
- **461** attend class meeting, view lectures, participate in Q&A
  - Recommendation: print slides and take notes
  - Not everything covered in class is on slides
  - You are responsible for everything covered in class
- **561** is responsible for all 461 lecture material, but need not attend 461 class meeting or Q&A

Learning the Material: Precepts

- 461 precepts focus on working problems, expanding concepts, programming prep for assignments
  - Led by TAs

- 561 precepts discuss papers in depth
  - Discuss 1 research paper in depth each week; 5 *insightful* comments due on Perusall the evening before each precept (i.e., Thursday)
  - Topic will relate to that week's 461 lectures, but assumes 461 content as background
  - Precept attendance is critical
  - Let instructors know if you must miss, accommodations made

Learning the Material: Books

- Main textbook
  - *Computer Networks: A Systems Approach*, by Peterson and Davie
  - Also online: https://book.systemsapproach.org/

- Additional books (may be of interest)
  - Networking textbooks
    - *Computer Networking: A Top-Down Approach Featuring the Internet*, by Kurose and Ross
    - *Computer Networks*, by Tanenbaum
  - Network programming references
    - *TCP/IP Illustrated, Volume 1: The Protocols*, by Stevens
    - *Unix Network Programming, Vol 1: Sockets Networking API*, by Stevens, Fenner, & Rudolf

Grading in **UG COS 461**

- Five assignments (50% total)
  - 90% 24 hours late, 80% 2 days late, 50% >5 days late
  - **Three** free late days (we'll figure which one is best)
  - Only failing grades I've given are for students who don't / try to do assignments
- Midterm exam (20%)
- Final exam (25%)
- Class participation (precept, 5%)

Grading in **Graduate-Level COS 561**

- Semester-long Research Project (40% total)
  - Includes proposal, presentation, and final write-up
  - In groups of 3-4 students; must involve programming
  - Can (1) reproduce research results, or (2) conduct novel research; regardless, *must* relate to COS 561 topics
- One take-home midterm exam (30% total)
  - Open-ended questions, e.g., how solutions work/don’t work, extensions to solutions for different goals/settings
- Participation (precept, 30%)
  - Includes in-precept discussion, paper presentation, & Perusall comments

Policy: Write Your Own Code

Programming is an individual, creative process. At first, discussions with friends is fine. When writing code, unless stated otherwise, the program must be your own work. ChatGPT and similar disallowed.

Do not copy another person's programs, comments, or any part of submitted assignment. This includes character-by-character transliteration but also derivative works. Cannot use another's code, etc. even while "citing" them.

Writing code for use by another or using another's code is academic fraud in context of coursework and will be referred to the University.

Do not publish your code e.g., on github, during/after course!

Setting Expectations:
Don’t expect 24x7 answers

- Try to figure out yourself

- Forums are not for debugging
  - Utilize right venue: Go to TA office hours
  - Send detailed Q’s / bug reports, not “no idea what’s wrong”

- Instructors are not on call 24 x 7
  - Don’t expect response before next business day
  - Questions Friday night @ 11pm should not expect fast responses.

Today

1. Course Introduction and Policies (461 & 561)

2. **Internet Basic Principles**

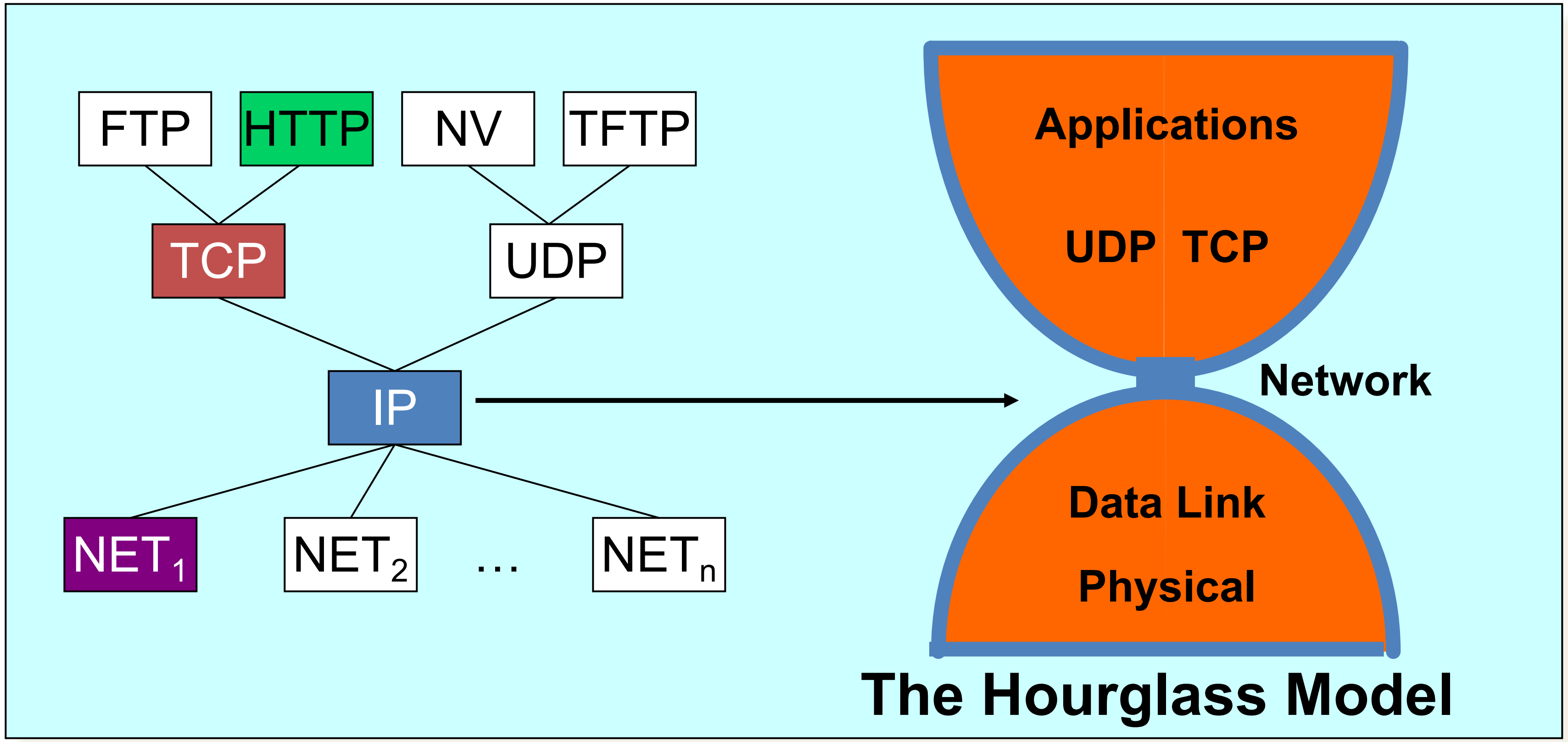
3. Link Layer

What *is* the Internet?

The Internet is the worldwide, publicly accessible network of interconnected computer networks that transmit data by packet switching using the standard Internet Protocol (IP).

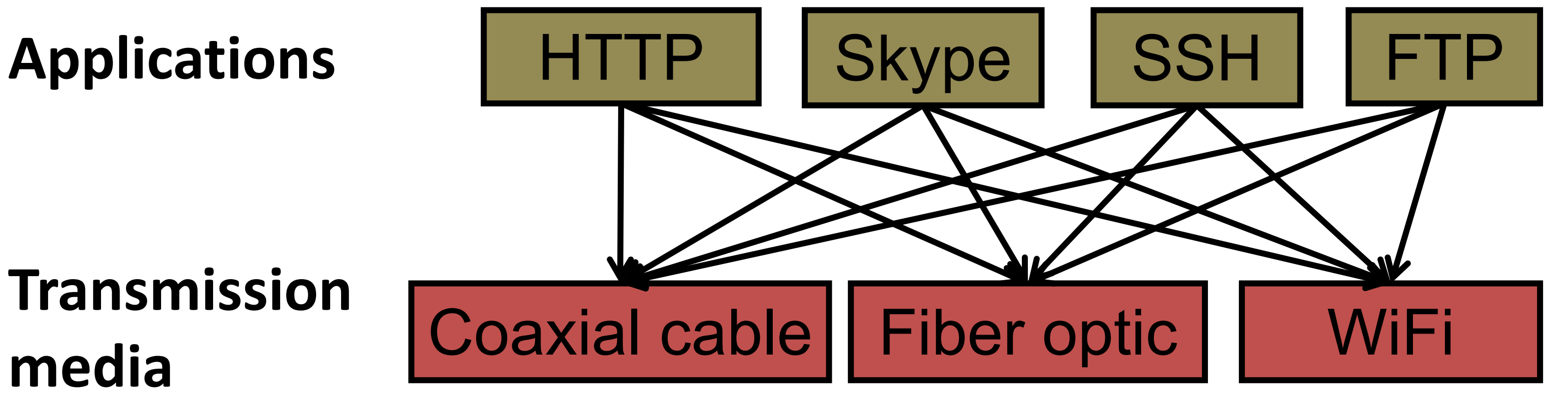
It is a "network of networks" that consists of millions of smaller domestic, academic, business, and government networks, which together carry various information and services.

The Internet Protocol Suite



The thin Network layer facilitates **interoperability**

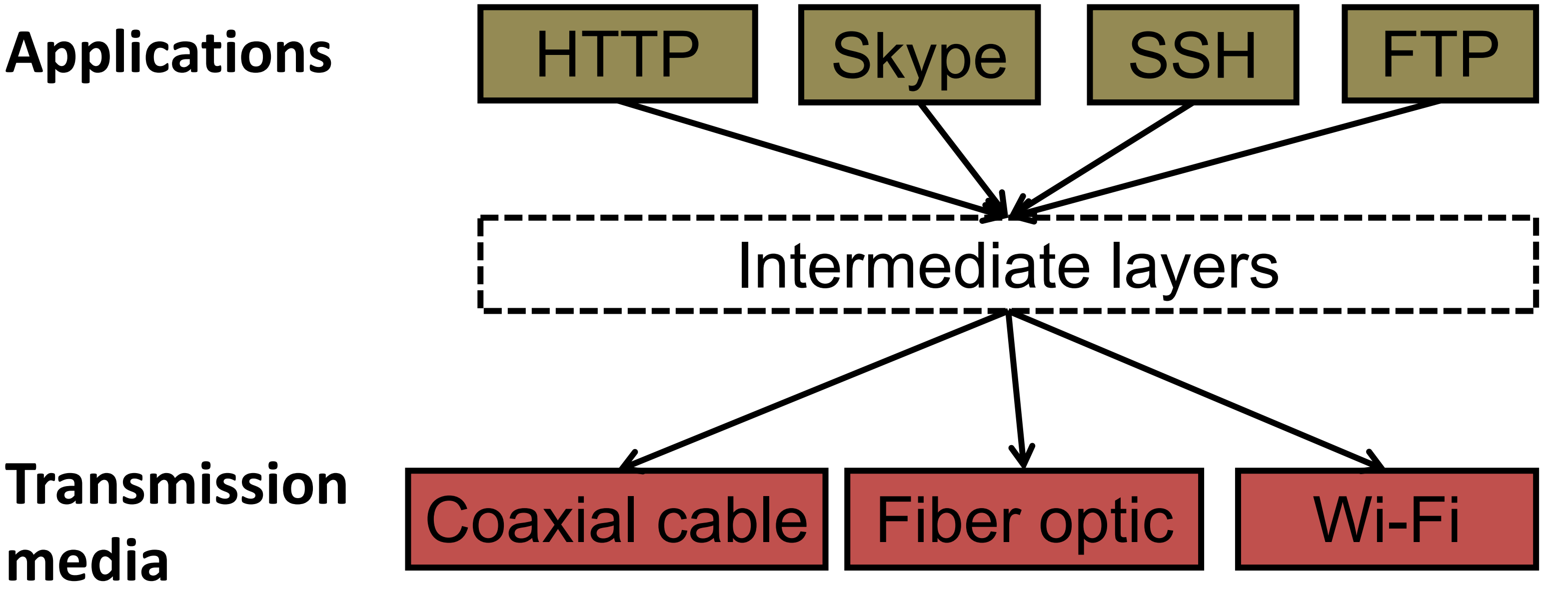
Coping with application/link heterogeneity



- 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?

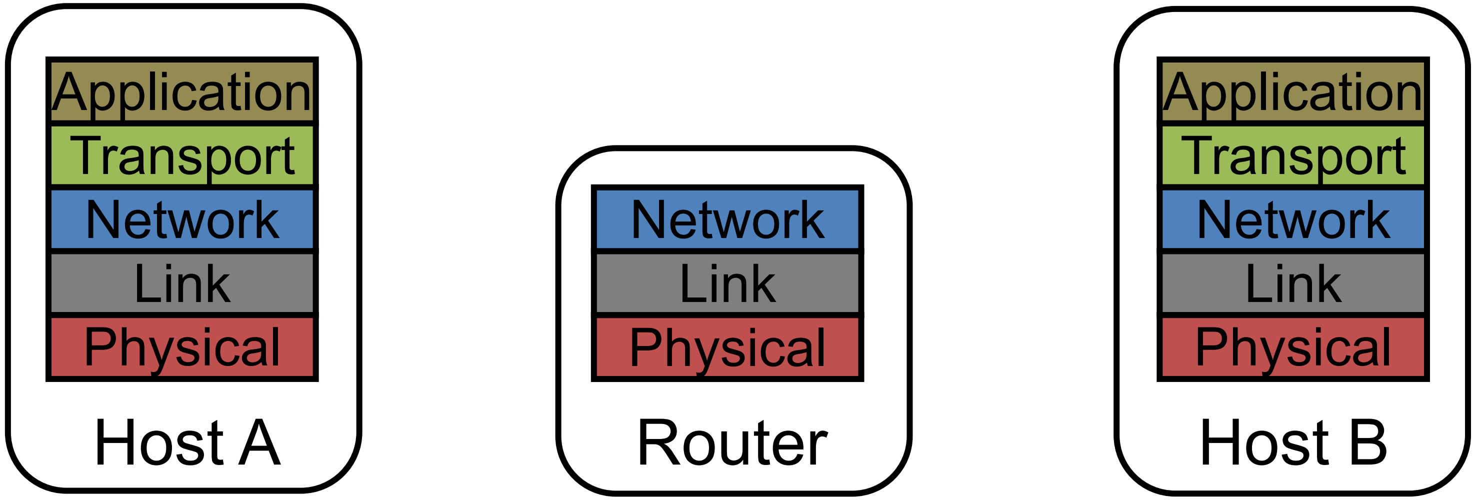
Internet solution: Intermediate layers



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

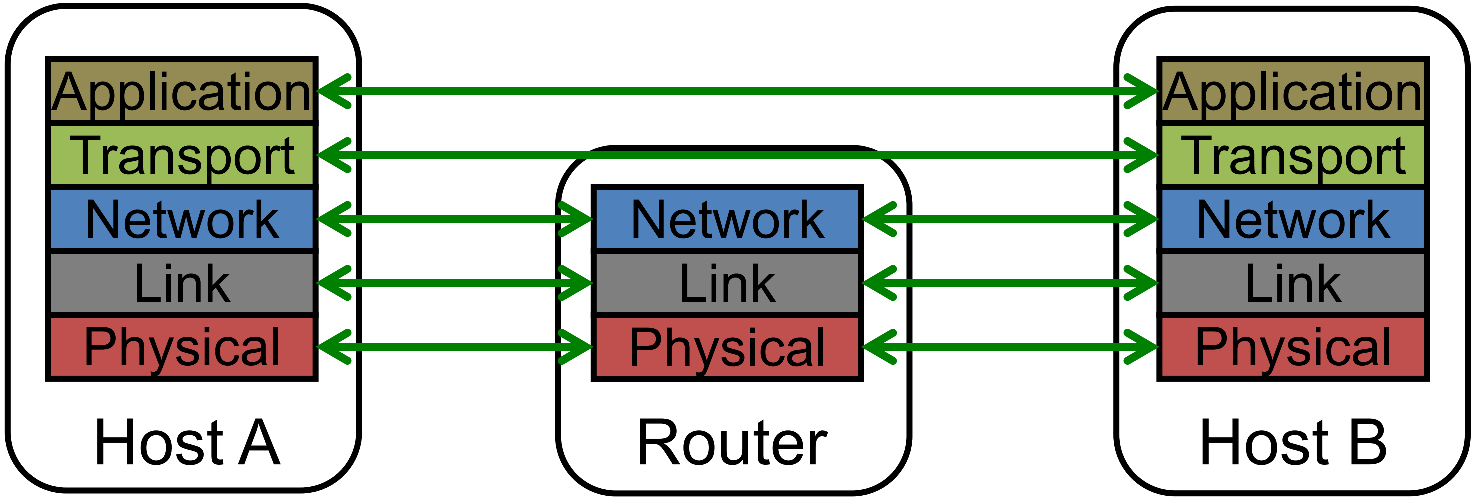
Who does what?

- Five "Internet architecture" layers
  - Lower three layers are implemented **everywhere**
  - Top two layers are implemented **only at end hosts**



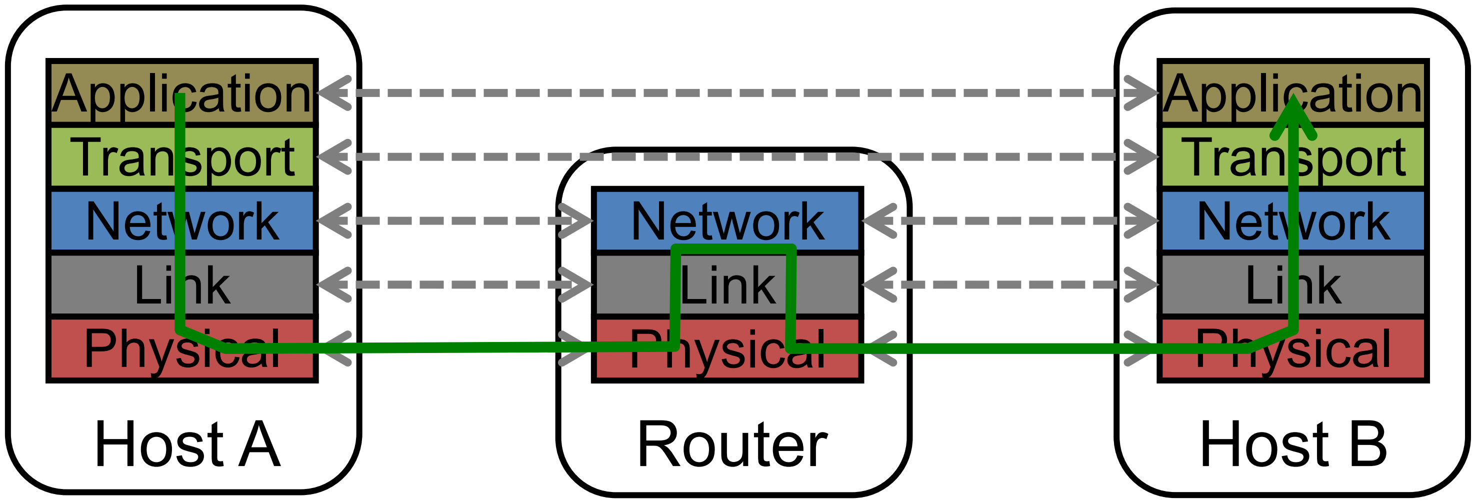
Logical communication

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



Physical communication

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



THE END-TO-END ARGUMENT

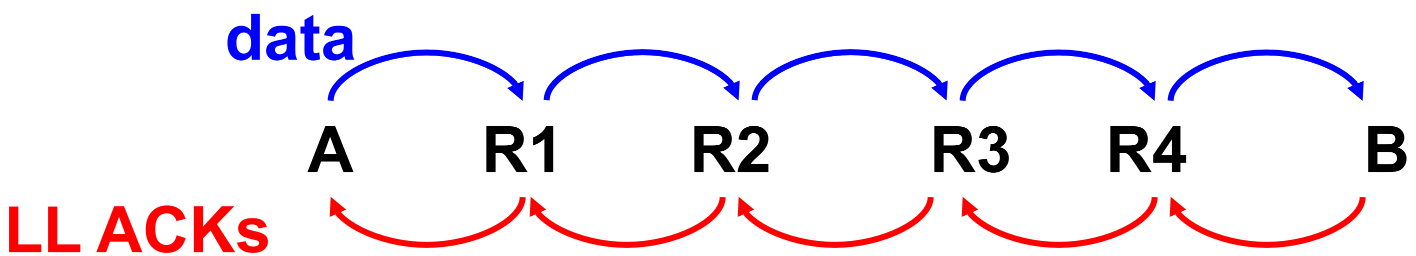
Motivation: End-to-End Argument

- 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?***

Discussion Prompt: *Careful file transfer* from A to B



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

- A hypothetical design:
  - Read file from **A**'s disk
  - **A** sends stream of packets containing file data to **B**
    - Layer 2 (hop by hop) 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 can 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** **router** 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?
  – Suppose there's a router with bad RAM; how to find it?

- Is a whole-file checksum enough?
  – Poor performance: must resend whole file each time one packet (bit) corrupted!

Perils of low-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,** would jitter packets when a few drops OK anyway

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

Advantages of low-layer implementation

- 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!**

The 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 <u>often</u> **wasteful of effort, inefficient**

Today

1. Course Introduction and Policies (461 & 561)

2. Internet Basic Principles

3. **Link Layer**

Link-Layer Services

- Encoding
  - Representing the 0s and 1s
- Framing
  - Encapsulating packet into frame, adding header, trailer
  - Using MAC addresses rather than IP addresses
- Error detection
  - Errors caused by signal attenuation, noise
  - Receiver detects presence, may ask for repeat (ARQ)
- Resolving contention
  - Deciding who gets to transmit when multiple senders want to use a shared media
- Flow control (pacing between sender & receiver)

Sharing the Medium- Comparing the Three Approaches

- Channel partitioning is
  (a) Efficient/fair at high load, inefficient at low load
  (b) Inefficient at high load, efficient/fair at low load

- “Taking turns”
  (a) Inefficient at high load
  (b) Efficient at all loads
  (c) Robust to failures

- Random access
  (a) Inefficient at low load
  (b) Efficient at all load
  (c) Robust to failures

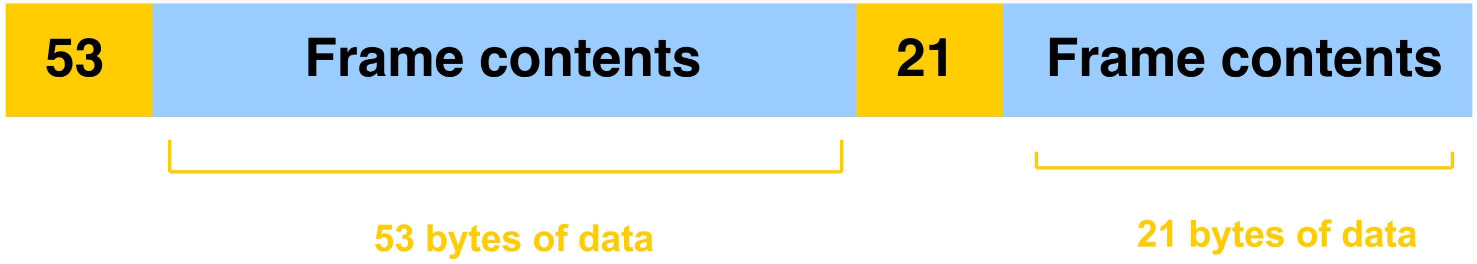
Framing

- Specify how blocks of data are transmitted between two nodes connected on the same physical media
  - Service provided by the data link layer
  - Implemented by the network adaptor

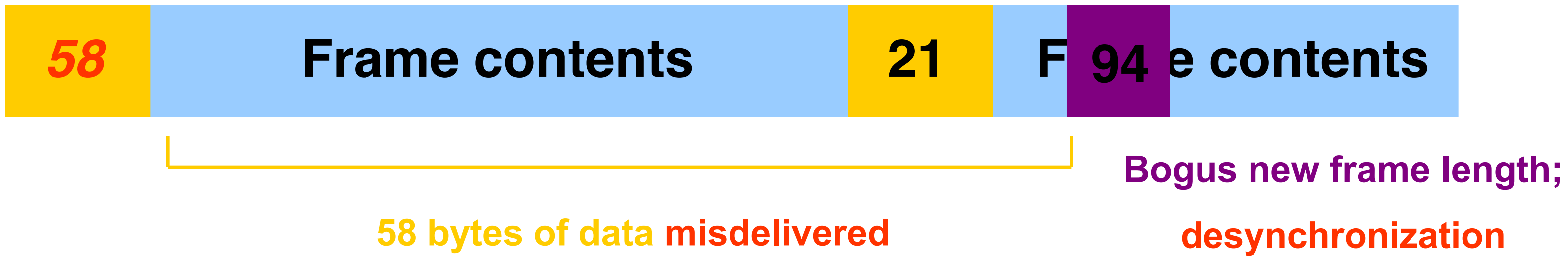
- Challenges
  - Decide when a frame starts & ends
  - How hard can that be?

Simple Approach to Framing: Counting

- Sender: begin frame with byte(s) giving length
- Receiver: extract this length and count



- How can this go wrong?
- On occasion, the count gets **corrupted**



Framing: Sentinels

- Delineate frame with special pattern (*sentinel*)
  - e.g., **01111110** ⇒*start*, **01111111** ⇒*end*

- Problem: what if sentinel occurs within frame?
- Solution: escape the special characters
  - E.g., sender always inserts a 0 after five 1s
  - … receiver always removes a 0 appearing after five 1s
- Similar to escaping special characters in C programs

When Receiver Sees Five 1s

01111110	**Frame content**	**01111111**

- If next bit 0, remove it, and begin counting again
  - Because this must be stuffed bit
  - Can’t be at beginning/end of frame

- If next bit 1, then:
  - If following bit is 0, this is start of frame
    - receiver has seen 01111110
  - If following bit is 1, this is end of frame
    - receiver has seen 01111111

Coming Up in 461

Next Class Meeting

Lectures 3 (Network Layer) and 4 (Network Devices)