Housekeeping

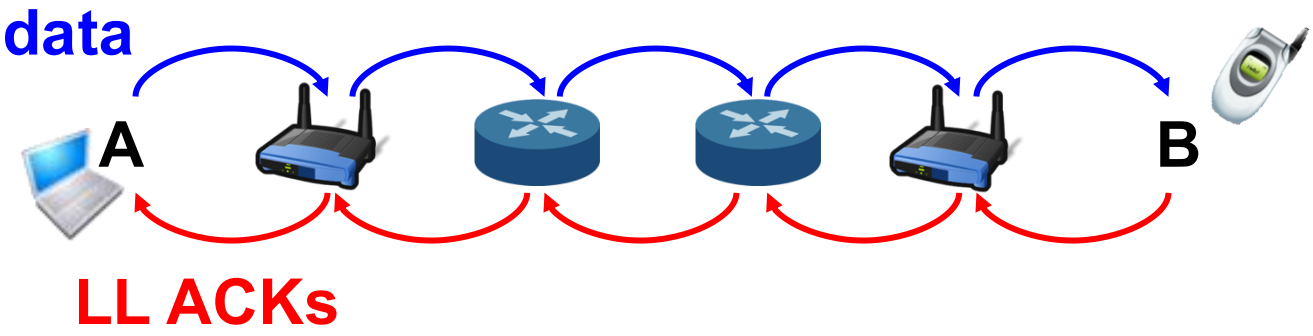
- Shifting from lecture-style to student-led readings throughout the semester
- Guest lectures coming soon:
  - Scaling real-time communications
  - Internet architecture and formal reasoning
  - (tentatively,) Video on Demand
- Reminder to please submit ”reflection” after class

"End-to-End Arguments in System Design"

(*ACM Trans. on Computer Systems*, November 1984)

J. Saltzer, D. Reed, and D. Clark

Example: Careful file transfer from A to B



- **Goal: Accurately copy file** on **A's** storage to **B's** storage
- Straw man design:
  - Read file from **A**'s storage
  - **A** sends packetized file to **B**
    - Link layer **resends** lost or corrupted packets at each hop
  - **B** writes file data to storage
- ***Does this meet the design goal?*** Assume Bit errors on links no issue

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 router** that forwards packet
- Why might errors be **likely?**
  - Drive for CPU speed & 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**

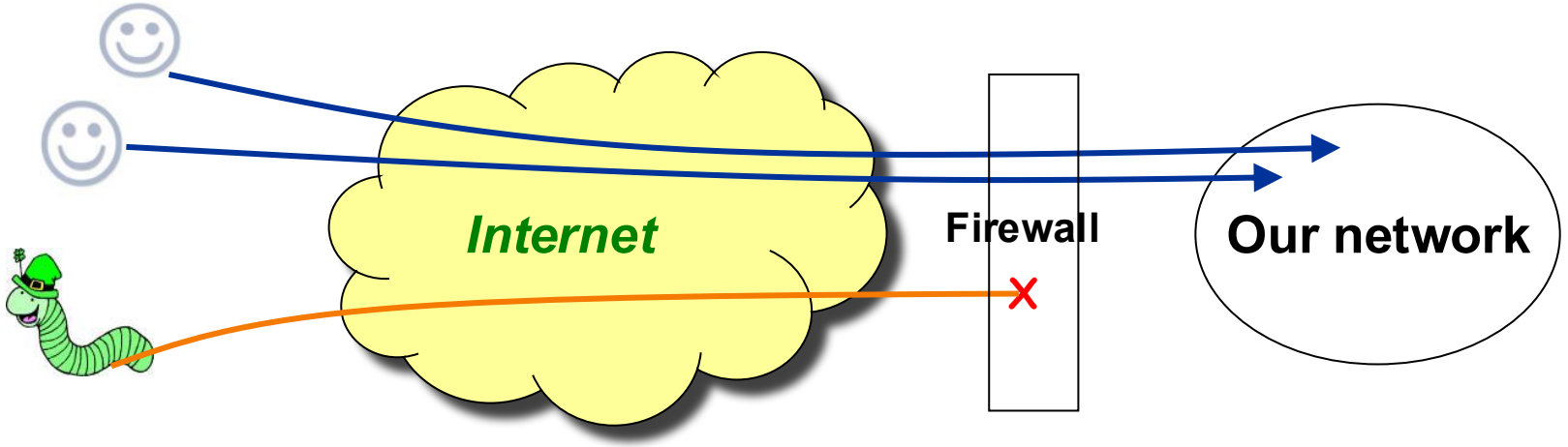
Perils of lower-layer implementation

- **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 is **entangled** with network, transport, apps, & vice-versa
    - ***e.g.:*** New header bit to improve congestion control? Many firewalls **filter all such packets!**
- **Yet, we probably do need firewalls**

Discussion

- When should the network support a function anyway?
  - E.g., link-layer retransmission in wireless networks?
- Who's interests are served by the e2e argument?
- How does a network operator influence the network without violating the e2e argument?
- Does the design of IP and TCP make it *hard* to violate the e2e argument?
  - E.g., middlebox functionality like NATs, firewalls, proxies
- Should the e2e argument apply to routing?

"A Protocol for Packet Network Intercommunication"

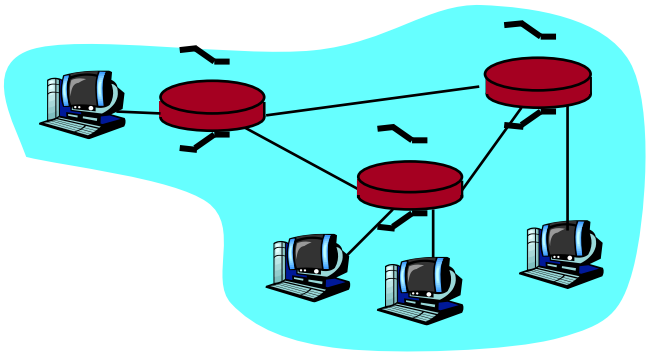
(*IEEE Trans. on Communications*, May 1974)

Vint Cerf and Bob Kahn

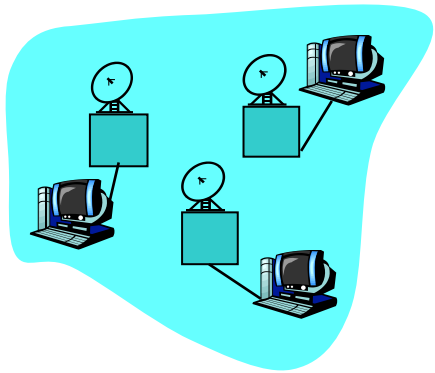
Written when Vint Cerf was an assistant professor at Stanford, and Bob Kahn was working at ARPA.

Life in the 1970s…

- Multiple unconnected networks
  - ARPAnet, data-over-cable, packet satellite (Aloha), packet radio, …
- Heterogeneous designs
  - Addressing, max packet size, handling of lost/corrupted data, fault detection, routing, …



ARPAnet



satellite net

Handling Heterogeneity

- Where to handle heterogeneity?
  - Application process? End hosts? Packet switches?
- Compatible process and host conventions
  - Obviate the need to support all combinations
- Retain the unique features of each network
  - Avoid changing the local network components
- Introduce the notion of a gateway

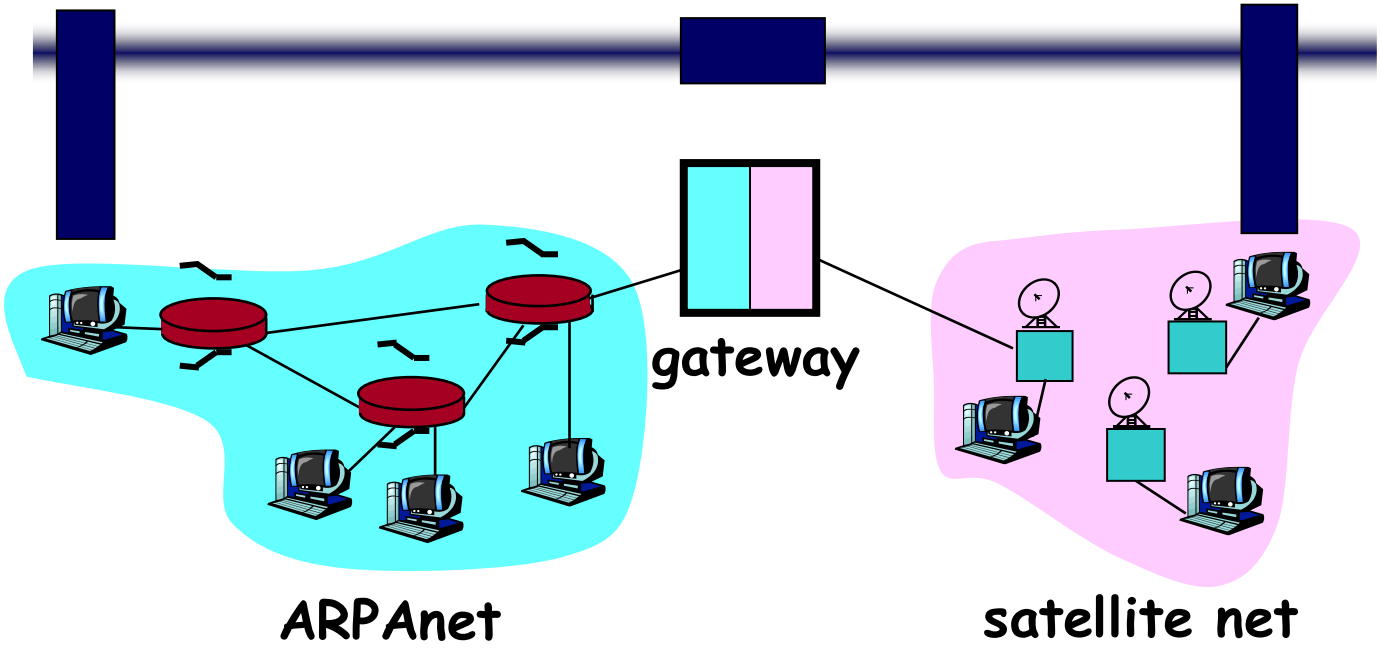
Internetwork Layer and Gateways

Internetwork Layer

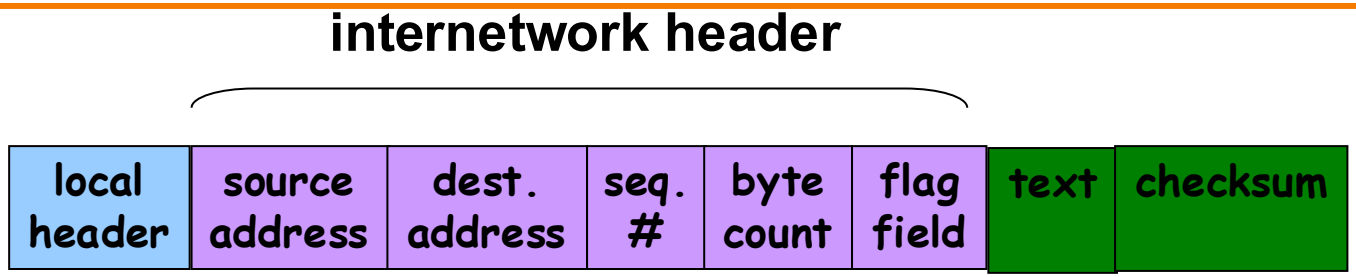
- Internetwork appears as a single, uniform entity
- Despite the heterogeneity of the local networks
- Network of networks

Gateway

- "Embed internetwork packets in local packet format or extract them"
- Route (at internetwork level) to next gateway



Internetwork Packet Format



- Internetwork header in standard format
  - Interpreted by the gateways and end hosts
- Source and destination addresses
  - Uniformly and uniquely identify every host
- Ensure proper sequencing of the data
  - Include a sequence number and byte count
- Enable detection of corrupted text
  - Checksum for an end-to-end check on the text

Process-Level Communication

- Enable pairs of processes to communicate
  - Full duplex
  - Unbounded but finite-length messages
  - E.g., keystrokes or a file
- Key ideas
  - Port numbers to (de)multiplex packets
  - Breaking messages into segments
  - Sequence numbers and reassembly
  - Retransmission and duplicate detection
  - Window-based flow control

Discussion (Perusall)

- What did they get right?
  - Which ideas were key to the Internet's success?
  - Which decisions still seem right today?

- What did they miss?
  - Which ideas had to be added later?
  - Which decisions seem wrong in hindsight?