Introduction to:
The Architecture of the Internet

Noah Mendelsohn
Tufts University
Email: noah@cs.tufts.edu
Web: http://www.cs.tufts.edu/~noah
What you should get from today’s session

- You will learn the difference between circuit and packet switching
- You will learn about the End-to-End principle, and why it is so useful as a guide to good systems design
- You will learn about tradeoffs regarding smarts “in the network” vs. at the edges
- You will learn why E2E is a key philosophical underpinning of the Internet
- You will start to think about errors and error recovery as a key aspect of system design
- You will learn why the End-to-End principle enabled Tim BL to innovate without permission from those who control the network
Per Hop Communication
Circuit switching (the way the old phone system worked)

When you make a call...

...operators would plug in connections to the next hop on the route
Circuit switching (automated dialing)

When you make a call...

...switches are set to reserve links for a fixed route for the life of the call
Circuit switching (automated dialing)
Circuit switching (automated dialing)

When you make a call...

...switches are set to reserve links for a fixed route for the life of the call

Can we reroute a connection?

Sometimes, but it’s difficult while the circuit is active.

...switches are set to reserve links for a fixed route for the life of the call
Packet communication

When you communicate...

...packets of digital data flow through the network
When you make a call...

...switches are set to reserve links for a fixed route for the life of the call.

Only if packets are stored and can be retransmitted: this requires digital transmission.

Can we locally recover errors on this hop?

...switches are set to reserve links for a fixed route for the life of the call.
Packet communication

Only if packets are stored and can be retransmitted: this requires *digital* transmission

Can we locally recover errors on this hop?

...packets of digital data flow through the network
Circuit switching (automated dialing)

When you make a call...

...switches are set to reserve links for a fixed route for the life of the call.

Can we reroute a connection?

Sometimes, but it’s difficult while the circuit is active.
Sending Streams over a Packet Network

Reliable, end-to-end streams can be sent using unreliable packets (datagrams)!

Input stream

Output stream
Sending Streams over a Packet Network

Can we reroute a connection?

Input stream

Output stream

Reliable, end-to-end streams can be sent using unreliable packets (datagrams)!
Sending Streams over a Packet Network

Can we reroute a connection?

Rerouting fits naturally with the model

Input stream

Output stream

Reliable, end-to-end streams can be sent using unreliable packets (datagrams)!
Sending Streams over a Packet Network

Reliable, end-to-end streams can be sent using unreliable packets (datagrams)!
End-to-End Arguments
The famous “End-to-End” paper

*End-to-End Arguments in System Design* by Jerry Salzer, David Reed, and David Clark

Abstract:

“This paper presents a design principle that helps guide placement of functions among the modules of a distributed computer system. The principle, called the end-to-end argument, suggests that functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level. Examples discussed in the paper include bit error recovery, security using encryption, duplicate message suppression, recovery from system crashes, and delivery acknowledgement. Low level mechanisms to support these functions are justified only as performance enhancements.”

Significance of the end-to-end “argument”

- Encourages clear thinking about smarts in the network vs at edges

- ACM SOSP says: “This paper gave system designers, and especially Internet designers, an elegant framework for making sound decisions. A paper that launched a revolution and, ultimately, a religion.”

- Became, post-facto, the philosophical rationale for Internet architecture

- Network neutrality: the network should not discriminate among applications requiring the same quality of service between the same nodes

- Tim Berners-Lee will tell you that such “neutrality” was essential to his ability to build and deploy a Web that nobody knew they needed

See 2006 video: [http://www.youtube.com/watch?v=Jev2Um-4_TQ](http://www.youtube.com/watch?v=Jev2Um-4_TQ)


Errors and Error Recovery
Computer systems and networks make mistakes

- **Errors**
  - Memories and disks: bits flipped in memory
  - CPUs: wrong answers
  - Communications: messages corrupted, lost, duplicated, delayed

- **Non-determinism**
  - Many errors are transient
  - *The machine will do something different the next time the program is run!*
  - Dealing with non-determinism is deeply different from traditional programming!

- **We need powerful techniques and clean abstractions for dealing with errors**
Fail stop and error detection

- **What if your machine computes** $2 + 2 = 5$ **and doesn’t warn anyone of the error?**
  - Your program would trust the result!

- **Architectures for dealing with errors**
  - Quietly recover to make the error invisible (e.g. ECC memory)
  - Alert some higher level software or hardware and provide facilities for recovery
  - Fail stop

- **Fail stop**
  - When an error occurs, just stop the machine, kill the program, drop the packet etc.
  - Optionally: alert someone

- **Assuming errors are not too frequent, if you can recover from stoppage, then all errors get recovered the same way!**
  - This is an example of the end-to-end argument
End-to-end recovery and building inexpensive systems

- **Observation:** if we can recover from most any failure of a computer, then failures need not be avoided...

- ...if we don’t mind doing such recovery occasionally, then we can build systems out of less expensive components that fail more often!

- **This is the key idea behind:**
  - Packet switching
  - The architecture of the Googleplex and similar “cloud” server farms
  - Use of inexpensive disks in RAID storage systems (Reliable Arrays of Inexpensive Disks)
Summary
Summary

- The end-to-end argument clarifies many aspects of distributed system design
  - End-to-end guarantees assure correctness
  - In-the-network processing and recovery is sometimes a useful optimization
- Error handling and associated non-determinism is a key challenge in building practical computing systems
- End-to-end strategies for error recovery
  - Often simplify implementation and proof-of-correctness for error recovery
  - Effective end-to-end error recovery can facilitate use of inexpensive components and unreliable delivery algorithms