CompSci 514: Computer Networks

Lecture 3: Design Philosophy

Xiaowei Yang
xwy@cs.duke.edu
- 14 years after Cerf and Khan’s paper

Overview

- The Design Philosophy of the DARPA Internet Protocols
  - What are the important goals
  - How prioritizing these goals shaped the design

- End-to-end arguments in system design
  - Correctness, completeness
  - Not performance
Design Philosophy: Motivation

- Cerf and Karn’s paper missed out some reasoning
- Internet has evolved after the Cerf and Karn’s paper
- Explain “why the protocol is as it is”
  - An explain-why-paper
Meta-points

- No law says Internet must be designed the way it is

- Design was driven by a specific set of goals

- Goals are different today
Three ideas

- Core principles
  - “The core principles and basic design decisions of the architecture.”
    - E.g. “minimalism”

- Mechanisms
  - “The second level of mechanism design that fleshes out the architecture and makes it into a complete implementation.”

- Deployment decisions
  - “The set of decisions related to deployment (e.g. the degree of diversity in paths) that lead to an operational network.”
Fundamental goal

- Inter-networking: an IP layer
  - Alternative: A unified approach
    - Can’t connect existing networks
    - Inflexible

- Packet switching vs circuit switching
  - Applications suitable for packet switching
  - Existing networks were packet switching

- Gateways
  - Chosen from ARPANET
  - Store and forward
  - Question: can we interconnect without gateways?
Background: two different flavors of network architectures

- Packet switching vs circuit switching
- “Stupid” vs “Intelligent”
Public Switched Telephone Network (PSTN)

- Proceeded the Internet by almost 100 years
  - In 1877, the American Bell Telephone Company, named after Alexander Graham Bell, opened the first telephone exchange in New Haven, Connecticut

- [https://en.wikipedia.org/wiki/Bell_System](https://en.wikipedia.org/wiki/Bell_System)
Local Exchange

More pairs than the number of circuits
Circuit Switching

Network resources (e.g., bandwidth) divided into “pieces”
- Pieces allocated to calls
- Resource piece *idle* if not used by owning call (*no sharing*)

- Dividing link bandwidth into “pieces”
  - Frequency division
  - Time division
TDM and FDM

TDM

Example:
4 users

FDM
Problems with FDM and TDM

- What if a user does not have data to send all the time (Over-provision)?
  - Consider web browsing
  - Inefficient use of resources

- Max # of flows is fixed and known ahead of time (Under-provision)
  - Not practical to change the size of quantum or add additional quanta for TDM
  - Nor add more frequencies in FDM
Packet Switching

Each end-end data stream divided into *packets*

- User A, B packets *share* network resources
- Each packet uses full link bandwidth
- Resources used *as needed*

Bandwidth division into “pieces”
Dedicated allocation
Resource reservation
Sequence of A & B packets does not have fixed pattern ➔ *statistical multiplexing*. 
Packet switching versus circuit switching

Packet switching allows more users to use network!

- 1 Mb/s link
- each user:
  - 100 kb/s when “active”
  - active 10% of time
- circuit-switching: fixed capacity
  - 10 users
- packet switching:
  - with 35 users, probability
    > 10 active less than .0004
Secondary goals

- In order of importance
  1. Survivable of network failures
  2. Multiple services
  3. Varieties of networks
  4. Distributed management
  5. Cost effective
  6. Easy attachment
  7. Resource accountable

- How will the order differ in a commercial environment?
Survivability

- **Goals:**
  - Completely mask any transient failure
  - Only total partition will disconnect host communications

- **Solutions**
  - Soft state: no essential state at intermediaries
    - No longer true
  - Fate sharing: critical state stored at end hosts
    - Losing state acceptable only when associated entity also fails
    - Consequences: stateless gateways, end-to-end reliability

- **Alternative:** why not replication?

- **Grade:** A
Types of Service

- **Motivation:** Multiple applications
  - Debugger should not require reliability
  - Realtime apps are more timing sensitive than loss

- **Solution:** separation of TCP/IP
  - TCP, UDP, DCCP, RTP, RCTP, STCP etc.
  - Minimal requirement of the network layer (end-to-end argument)

- **Grade:** A-
  - Supporting guaranteed services still difficult
Varieties of networks

- Minimal requirements of underlying networks
  - Reasonable min packet size

- Explicitly not assumed
  - Reliable or In-order delivery, broadcast/multicast, prioritized transmission, failure notification, support multiple-type of services
  - What would be different if they are assumed?
    - End-to-end arguments

- Grade: A
Other goals: Distributed management

- Autonomy is supported
- Little support for multiple collaborative management
  - Ex: BGP traffic engineering

- Grade: B
Other goals: Cost effectiveness

- Header space, retransmission
  - Much less a problem today

- Grade: A-
Other goals: Easy Attachment

- Host implementation is more complicated than in other architectures
  - Less a concern after open source TCP/IP implementation
  - Grade: A
  - - less robust to misbehaving hosts
Other goals: Accountability

- Almost no accountability
  - Source address spoofing attacks
  - DDoS

- Grade: F
Implementation

- Challenges:
  - Few tools to guide how to design networks with predictable performance other than correctness
  - Still true today
  - Ongoing research
    - Network verification
Motivations for datagram

- Switching unit is datagram
- Why not messages, connections, files etc.?
- Several advantages
  - Stateless gateways: no connection state
  - Multiple types of services
    - Easy to build efficient streaming out of datagram
    - Inefficient to build interactive apps out of circuits
  - Minimal assumptions
Changes to TCP

- Original design provided flow control both on bytes and packets
- Revised design uses bytes only
  - Simplicity
  - Allow to insert control information
  - Allow multiple messages into one TCP segment
  - Break up large packets
  - Push flag
Overview

- The Design Philosophy of the DARPA Internet Protocols
  - What are the important goals
  - How prioritizing these goals shaped the design

- End-to-end arguments in system design (1984)
  - Correctness, completeness
  - Not performance
What is the paper about?

- Where to place functions in a distributed computer system
  - End point, networks, or a joint venture?

- Authors’ arguments:

  “The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)”
End-to-End Argument

- Extremely influential

- “…functions placed at the lower levels may be redundant or of little value when compared to the cost of providing them at the lower level…”

- “…sometimes an incomplete version of the function provided by the communication system (lower levels) may be useful as a performance enhancement…”
The counter argument

- Modularity argument:
  - It is tempting to implement functions at lower layers so that higher level applications can reuse them.

- The end-to-end argument:
  - "The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of communication."
  - "Centrally-provided versions of each of those functions will be incomplete for some applications, and those applications will find it easier to build their own version of the functions starting with datagrams."
Techniques used by the authors

- The authors made their argument by analyzing examples
  - Reliable file transfer
  - Delivery guarantees
  - Secure data transmission
  - Duplicate message suppression
  - FIFO
  - Transaction management
  - Can you think of more examples to argue for or against the end-to-end argument?

- Can be applied generally to system design
Example: Reliable File Transfer

- Solution 1: make each step reliable, and then concatenate them
  - Uneconomical if each step has small error probability
Example: Reliable File Transfer

- Solution 2: end-to-end check and retry
  - Correct and complete
An intermediate solution: the communication system provides internally, a guarantee of reliable data transmission, e.g., a hop-by-hop reliable protocol
- Only reducing end-to-end retries
- No effect on correctness
Question: should lower layer play a part in obtaining reliability?

- **Answer:** it depends
  - **Example:** extremely lossy link
    - One in a hundred packets will be corrupted
    - 1K packet size, 1M file size
    - Probability of no end-to-end retry: \((1-1/100)^{1000} \approx 4.3e-5\)
Performance enhancement

- “put into reliability measures within the data communication system is seen to be an engineering tradeoff based on performance, rather than a requirement for correctness.”
Performance tradeoff is complex

- Example: reliability over a lossy link using retries
Performance tradeoffs

- Example: reliability over a lossy link using retries
  - But they won't help real time applications, applications with built-in error correction mechanisms

- Tradeoffs:
  - Applications that do not need them will pay the cost anyway
  - Low-level subsystems may not have as much information as the higher levels to do the job as efficiently
End-to-End Argument: Discussion

- The original end-to-end argument emphasizes correctness & completeness, not
  - complexity: is complexity at edges result in a “simpler” architecture?
  - evolvability, ease of introduction of new functionality: ability to evolve because easier/cheaper to add new edge applications than change routers?
- Technology penetration: simple network layer makes it “easier” for IP to spread everywhere
Summary of End-to-End Arguments

- If the application can do it, don’t do it at a lower layer -- anyway the application knows the best what it needs
  - add functionality in lower layers iff it is (1) used and improves performances of a large number of applications, and (2) does not hurt other applications

- Success story: Internet
  - a minimalist design
Summary

- The Design Philosophy of the DARPA Internet Protocols
  - What are the important goals
  - How prioritizing these goals shaped the design

- End-to-end arguments in system design
  - Correctness, completeness
  - Not performance