E2E Arguments & Project Suggestions (Lecture 4, cs262a)

Ali Ghodsi and Ion Stoica,
UC Berkeley
Jan 29, 2018

Software Modularity

Break system into modules

- Well defined interface, implementation behind interface
- Separation of concerns, encapsulation, information hiding, …

Main advantages?

- Hides complexity from user, division of labor, scaling teams
- Implementation can change, interface stays the same, apps don't need to change
- SQL! Functional programming, ADT, functions, modules, OS, …
- Main way humans build complex systems (e.g. computer, car, buildings)
- Can be more efficient. Why?

Software Modularity

Disadvantages?

- Can hurt performance
- DLL-hell, can it apply to networking?
- Can it increase complexity?

Network Modularity

Like software modularity, but with a twist:

Implementation distributed across routers and hosts

Must decide:

- How to break system into modules
- Where modules are implemented

Layering

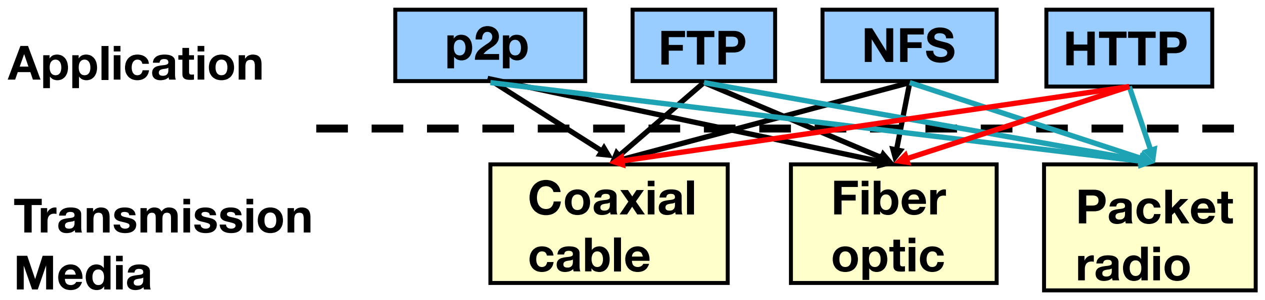
Layering is a particular form of modularization

System is broken into a vertical hierarchy of logically distinct entities (layers)

Service provided by one layer is based solely on the service provided by the layer(s) below

Rigid structure: easy reuse, performance can suffer

The Problem



Re-implement every application for every technology?

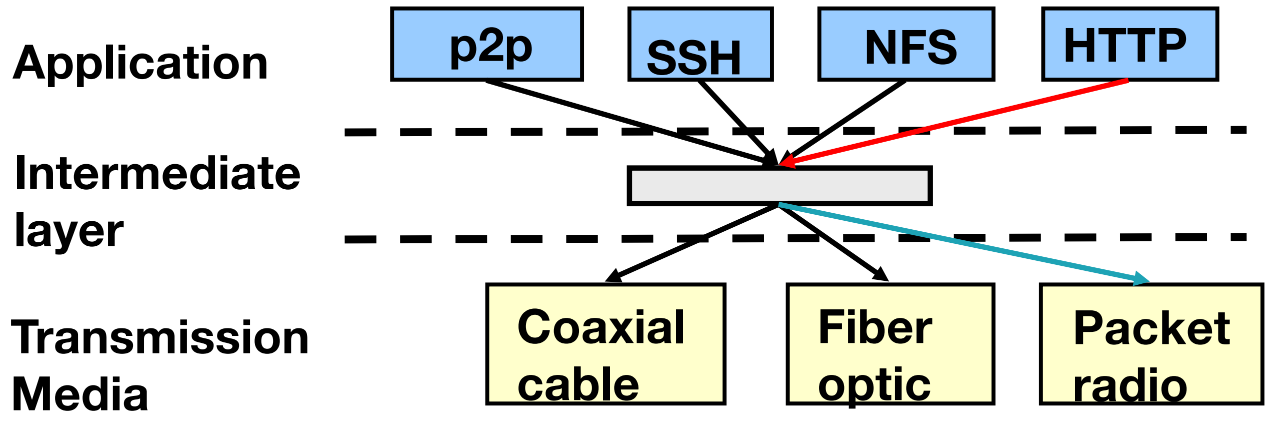
No! But how does the Internet architecture avoid this?

How do you solve problems in CS?

Solution: Intermediate Layer

Introduce an intermediate layer that provides a single abstraction for various network technologies

- A new app/media implemented only once



Placing Functionality

Most influential paper about placing functionality is
“**End-to-End Arguments in System Design**”
by Saltzer, Reed, and Clark

“Sacred Text” of the Internet

- Endless disputes about what it means
- Everyone cites it as supporting their position
- http://www.martingeddes.com/think-tank/the-future-of-the-internet-the-end-to-end-argument/

Basic Observation

Some applications have end-to-end performance requirements

- Reliability, security, etc

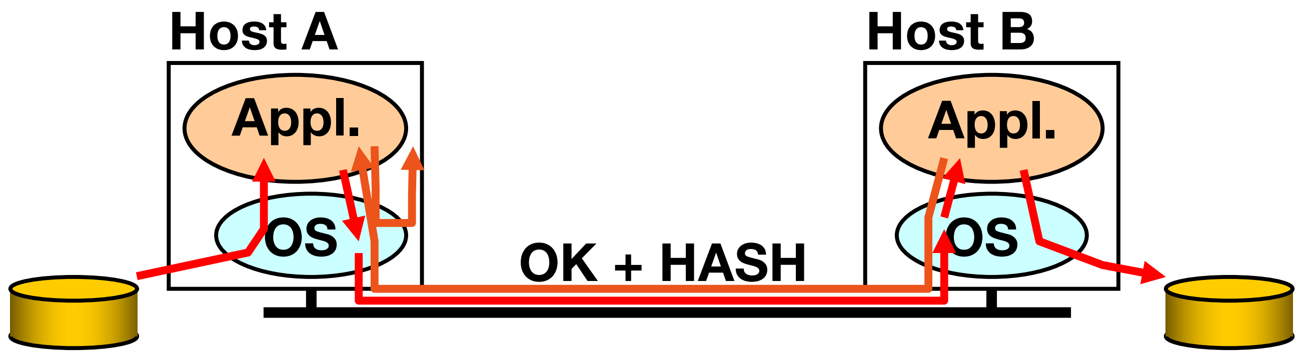
Implementing these in the network is very hard:

- Every step along the way must be fail-proof

Hosts:

- Can satisfy the requirement without the network
- Can't depend on the network

Example: Reliable File Transfer



Solution 1: make each step reliable, and then concatenate them

Solution 2: end-to-end check and retry

Discussion

Problems with Solution 1?

- What happens if any network element misbehaves?
- Receiver has to do the check anyway!
- Doesn’t reduce host implementation complexity
- Does increase network complexity
- Probably imposes delay and overhead on all applications, even if they don’t need functionality

Discussion

Advantages of Solution 2?

- Full functionality can be entirely implemented at application layer with no need for reliability from lower layers
- Simpler solution
- Fewer dependencies, fewer parts
- Changes in software easier than changes in hardware

When would you still want reliability on a link-level?

- Really lossy link → big performance improvement

Conservative Interpretation

“Don’t implement a function at the lower levels of the system unless it can be completely implemented at this level”

Unless you can relieve the burden from hosts, then don’t bother

Lossy-link vs conservative interpretation?

Radical Interpretation

Don’t implement anything in the network that can be implemented correctly by the hosts

- E.g., multicast

Make network layer absolutely minimal

- Ignore performance issues

Moderate Interpretation

Think twice before implementing functionality in the network

If hosts can implement functionality correctly, implement it a lower layer only as a performance enhancement

But do so only if it does not impose burden on applications that do not require that functionality

Discuss examples

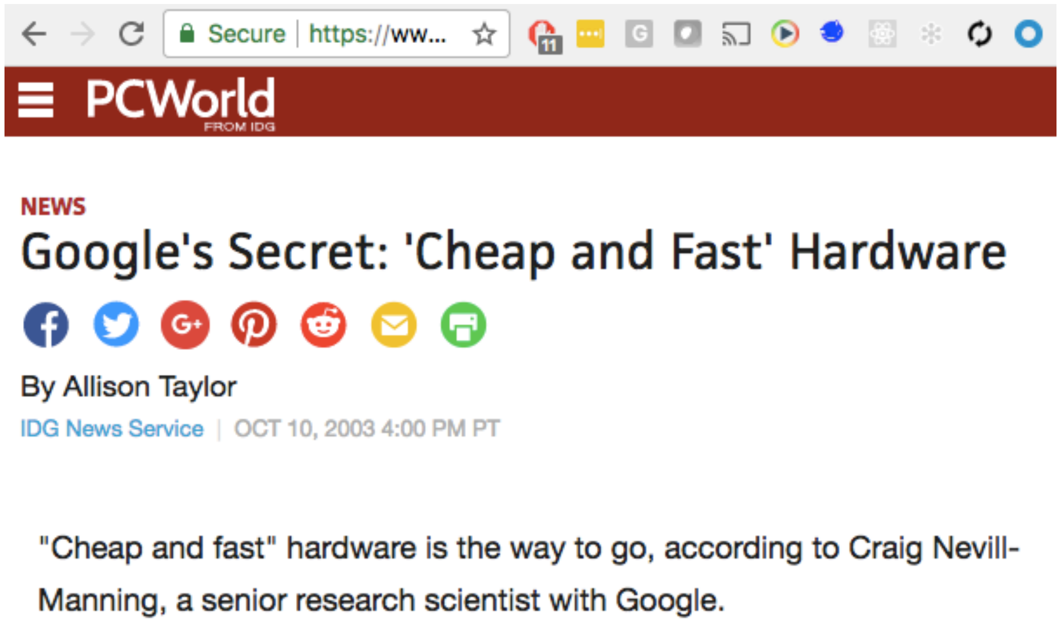
How is it E2E?

- Network Address Translation (NATs)
- Middleboxes
- Congestion control

Discuss: **Google**

By using commodity PC hardware, which is similar to that of home PCs, Google buys cheap and builds high levels of redundancy into its system in an effort to compensate for the fact that one full day of Google use on a server is the equivalent of 40 machine years, Nevill-Manning said.

"Each server has many twins," he said. "Replication is needed for



NEWS

Google's Secret: 'Cheap and Fast' Hardware

By Allison Taylor

IDG News Service | OCT 10, 2003 4:00 PM PT

"Cheap and fast" hardware is the way to go, according to Craig Nevill-Manning, a senior research scientist with Google.

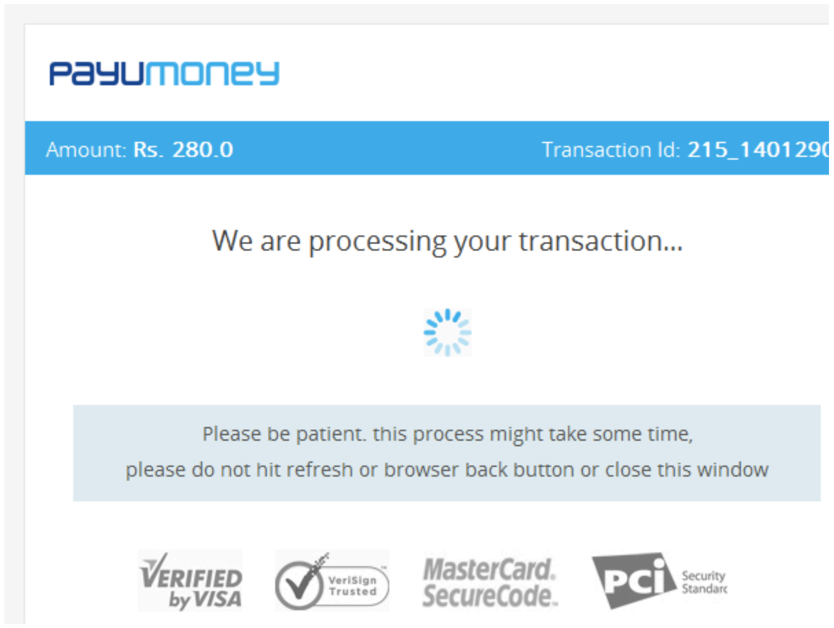
Discuss: **WWW**

Why include this example?

- State management in App Server / Web App?

Can refresh/back-button issues be avoided? How?

- State closer to app (host)



Amount: Rs. 280.0

Transaction Id: 215_1401290

We are processing your transaction...

Please be patient. this process might take some time,
please do not hit refresh or browser back button or close this window

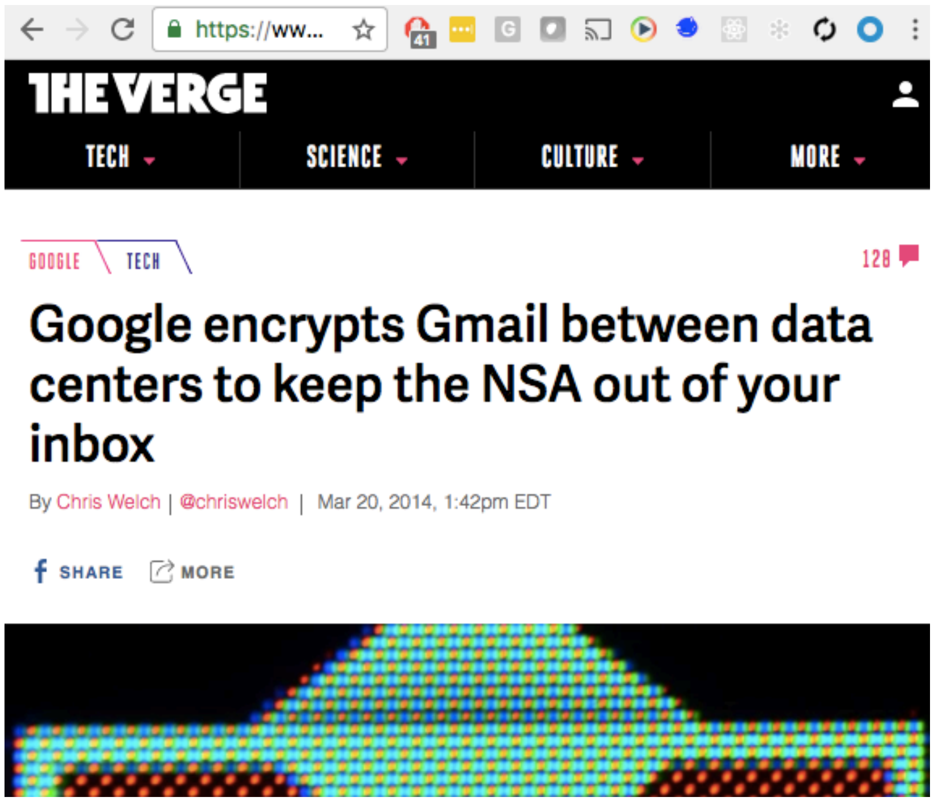
Discuss: **e-mail**

Are we safe now?

- Could still force companies to reveal keys, or they leak keys

What's the bullet proof solution

- Put keys on host


GOOGLE TECH 128

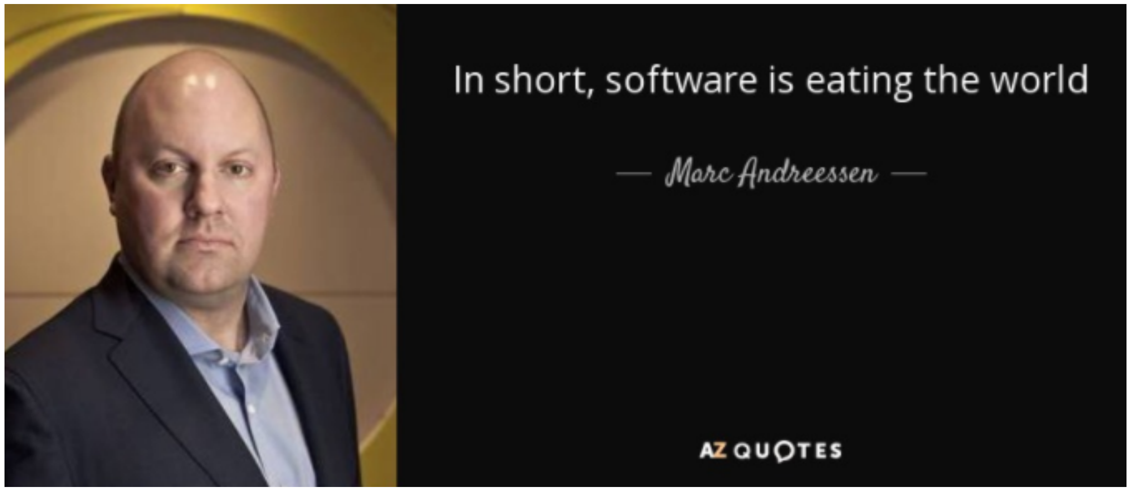
Google encrypts Gmail between data centers to keep the NSA out of your inbox

By Chris Welch | @chriswelch | Mar 20, 2014, 1:42pm EDT

Discuss

Examples

- Software Defined Networking
- Software Defined Storage
- VMs and containers



Summary

Layering is a good way to organize systems (e.g., networks)

Implement functionality at highest level possible, optimize if needed in layers below

E2E argument encourages us to keep lower layers (e.g., IP) simple

Projects Suggestions
(see https://tinyurl.com/y8rfenr5)

Project Dates

Wednesday, **1/31**: add more projects to google sheet

- Add your own project if you are looking for a partner

Wednesday, **2/7**: pick a partner and send your project proposal

- We will send out a google form to fill in for your project proposals

Monday, **3/12**: project progress review

- More details to follow