Applications and Abstractions

David Clark
MIT
May, 2012

A talk in several parts

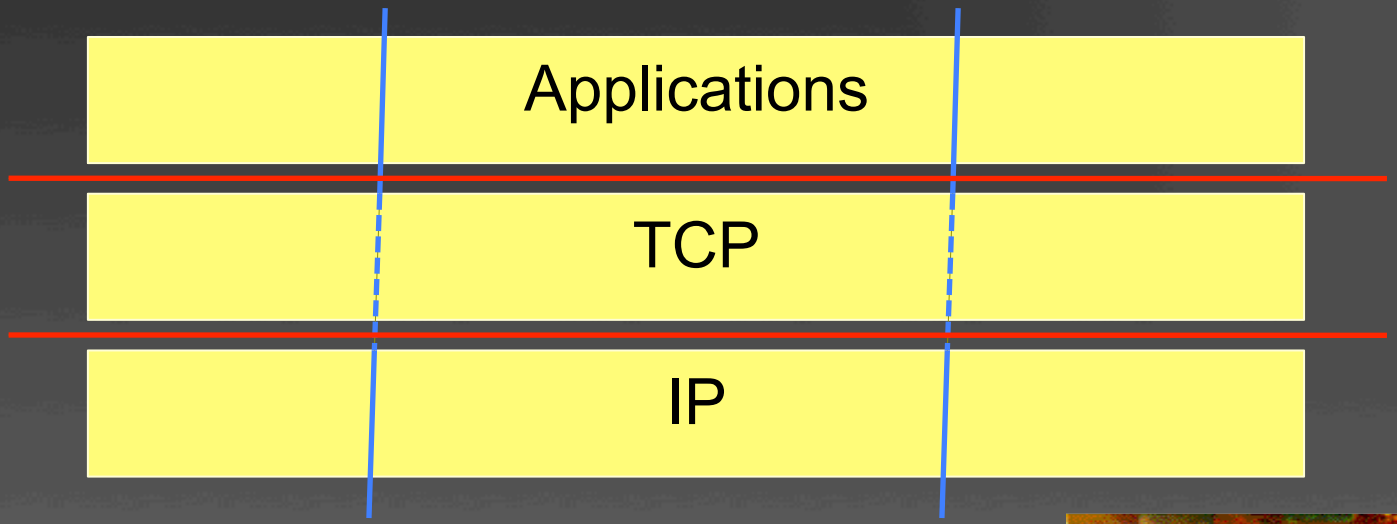
- Some whines about the workshop announcement.
- Approaches to modularity.
- The limits of layering.
- How to specify performance (not).
- Generality vs. specificity.
  - Deviation from the ideal.
- Security and availability.
- Application design and trust.

The workshop announcement

- A few phrases there stung a little bit…
  - "little evidence of modularity"
    - I would claim we have clear modularity.
    - Question is whether it is the *right* modularity.
  - "quirks and deficiencies"
    - We should discuss whether these are design errors or reflection of fundamentals.
  - General claim:
    - Over-specific specification may not be a good thing.

Modularity

- We have two well-known kinds.
  - Modules with horizontal (red) interfaces: layers.
  - Modules with vertical (blue) interfaces: peer or autonomy.

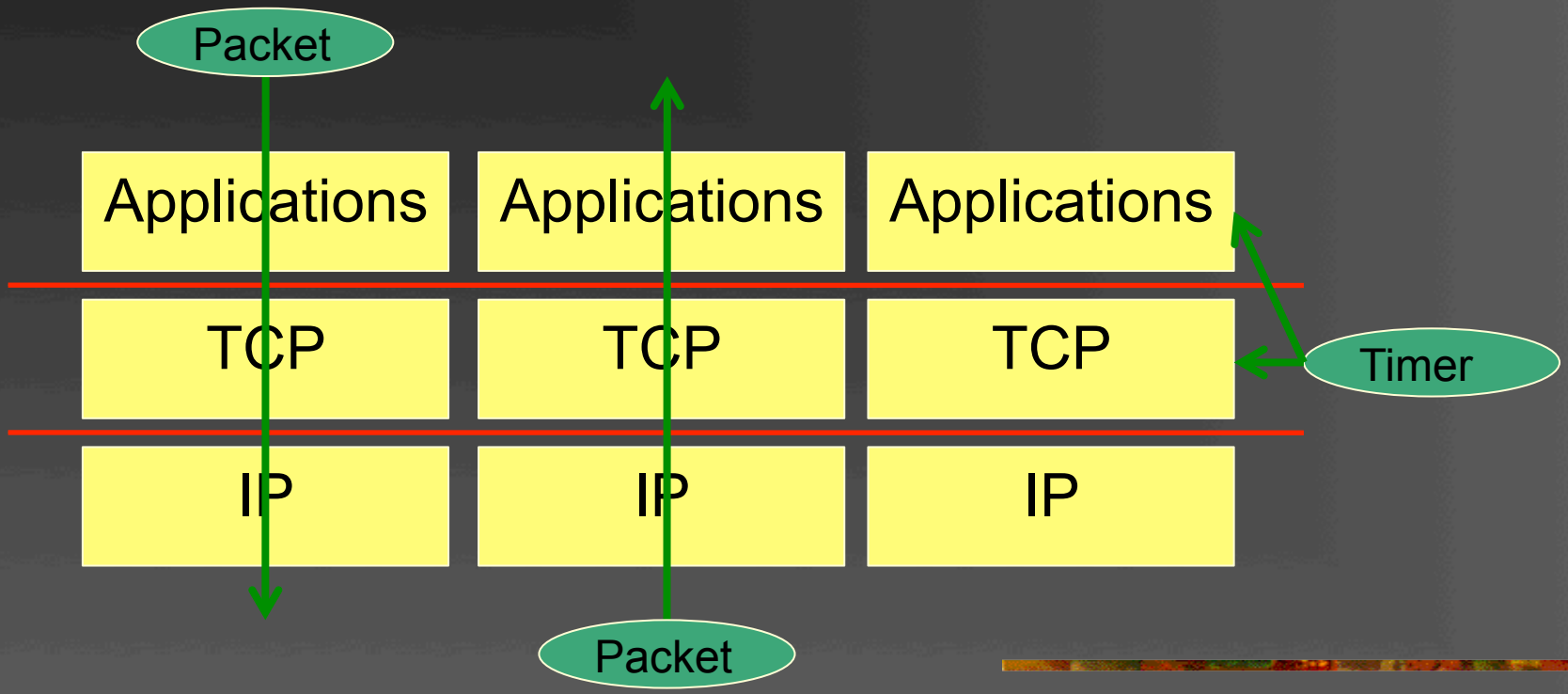


Layers

- Basic principle: dependency.
  - Higher layers “depend” or “make use of” lower layers. Not the other way.
    - Is this a useful basis for modularity?
  - In the larger (socio-economic) context, this is false. All “layers” depend on each other.
- Only other option I know:
  - Event-driven modularity.

Event-driven modularity

- Events:
  - Send a packet, receive a packet, timer goes off.



Why event-driven modules?

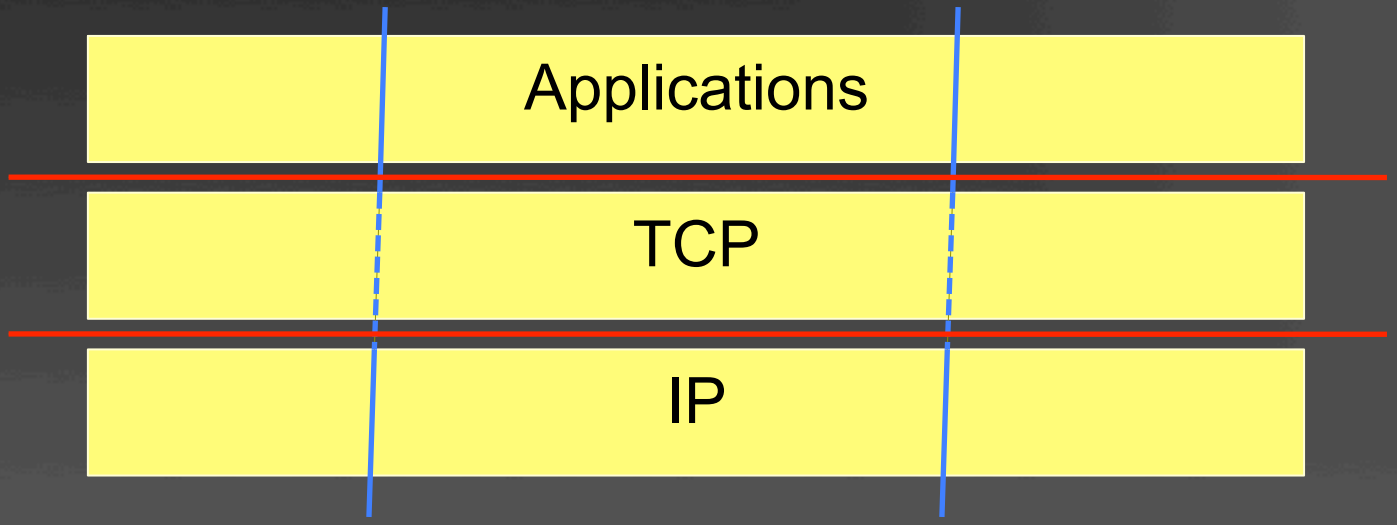
- Helps explicate cross-layer functional relationships, and *dynamic* dependencies.
- Guides implementation.
  - Upcalls.
  - Cross-layer modules.
- Look at events at different granularity.
  - Send/receive a packet.
  - Retrieve and view a web page.

Why study dynamic dependency?

- Helps explicate cross-layer functional relationships.
  - De-emphasize layers.
- Cross-layer functions are a critical and central issue
  - Even if event-driven modules are not.
- Many critical aspect of networking are not “layered”:
  - Performance,
  - Security,
  - Availability,
  - Quality of Experience.
- Some are layered only by force:
  - Economics (structural separation).
- If you are trying to explain why an application “works”, you need to look cross-layers at “what happens” .

Modularity

- We have two kinds.
  - Horizontal (red) interfaces: layers.
  - Vertical (blue) interfaces: peer or autonomy.



Vertical interfaces

- Reflect the distributed aspect of the network.
  - Physical
  - Economic/autonomous modules
- IP
  - BGP among AS regions.
- Applications
  - SMTP between email servers.
- Seems fundamental.
  - Come back to this.

Quirks and deficiencies

- This phrase could have referred to a number of issues.
- My guess: wide variation in observed behavior of network.
  - Throughput, latency, jitter, packet loss, etc.
  - You *cannot* make this variation go away.
  - Only question is how best to characterize it.
    - A different talk, but…

Performance specification

- An *ideal* network would reliably/correctly deliver any amount of desired information to a specified/willing set of receivers in zero time for zero cost.
  - (No receiver should get information it does not desire.)
- Real networks must deviate from the ideal because of real-world impairments.
  - An *efficient* network has only fundamental impairments.
- A *general* network is one that allows the user/application to trade off among the impairments.
  - Throughput, latency, jitter, packet loss, etc.

Performance is cross-layer

- Obtaining desired performance is a cross-layer objective.
  - Physical: buy/configure bandwidth.
  - Physical: deal with technology features.
  - IP: use QoS tools. In future: routing, multipath
  - Application: adapt to observed behavior, reconfigure pattern of distribution, adapt coding/quality, use economic discipline.
- The current term for application performance is Quality of Experience.
  - QoE. (How to measure—good question.)

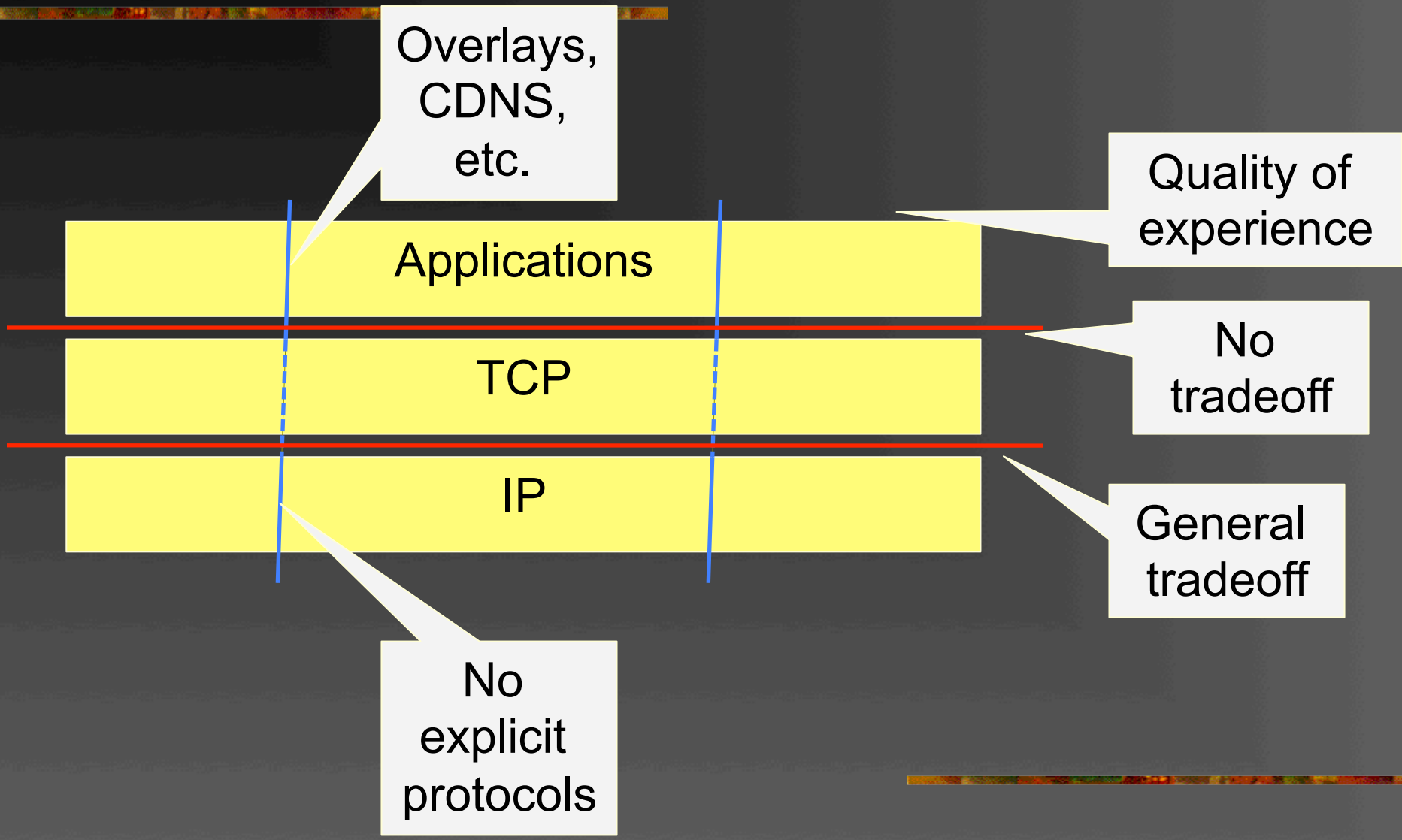
The Internet today

- Today:
  - Internet gives only implicit tools to trade off impairments.
    - Aside from QoS, which is currently MIA.
  - TCP is a fixed tradeoff with a very simple specification. Not general at this level.
  - Applications must measure service quality, not specify what they want, and adapt as best they can.
- Specification of explicit services sounds appealing, but can lead to “problems”.

The Abhay Parekh story

- Application designers demanded predictable and guaranteed bounds on throughput and jitter.
  - Best approach is WFQ, work conserving.
  - Flows characterized by token bucket.
- Parekh, in his MIT PhD thesis, proved the bounds.
  - Bound was ugly. Bound was tight.
- Application designers withdrew their demands and went for probabilistic bounds.

Performance interfaces



Other cross-layer issues

- Security
  - Assume the network itself is working.
  - Ignore attacks on end-nodes for the moment.
  - Focus on attacks on the application and its communication. (An application-centric view.)
- Availability
  - An aspect of security, but also deals with failures and errors.
    - Reliability, resilience, etc.

Security

- The classic taxonomy is confidentiality, integrity, and availability, with other features (e.g. provenance) sometimes added in.
- Confidentiality and integrity during communication.
  - Encryption is a good tool under certain assumptions.
    - You can protect your keys.
    - A simple model of integrity will suffice.
- The classic taxonomy misses a major issue.
  - Talking to actors you don't trust.
    - The norm today—email, web sites, etc.
  - Encrypted communication with an actor you don't trust is like meeting a stranger in a dark alley.
    - No witnesses.
- Availability (see next slides)

Security mechanisms

- Security mechanisms do not ensure correct operation.
  - DNSSEC
  - SSL/TLS
  - IPsec
  - SBGP and its kin
  - Self-signed identifiers (public key hashes, etc)
- They only give a clean indication of failure.
  - What we wanted was success.

Availability

- We lack a general approach. Try this…
  - An application is built of components.
  - The network connects components.
  - The application knows what these components are.
  - The network must either make the correct connection or fail cleanly.
  - The application must recover.

Availability (theory of)

- Detection of failures
  - Make network/communication failures “fail-stop”.
  - Less obvious if more subtle: application-level failures.
  - May occur at end-node or along the invocation chain.
- Isolation of failure
  - (Not good at this today, either in network or in application.)
- Failover to different component.
  - Application must be involved in decision.
  - Implies replication of components, lots of mechanism.
- Reporting and correction of failure.
  - We don’t often support this capability.
- (Security: firewall = ¬ availability)

Cross-layer again

- All these steps are cross-layer:
  - Ensuring failures are fail-stop and detection.
  - Isolation and localization of failure.
  - Bypass and failover.
  - Reporting.
- Conclusion: I am suspicious of using layer boundaries as basic abstraction points.

A look at applications

- After all, they are the justification for networks.
- Look at design issues, modularity, and network dependence.

In the early days…

- We did not give much guidance to application designers.
  - Was not a reliable byte stream and the DNS enough?
- Our view of application design was simple.
  - Two party interaction.
  - (Except email)

As things evolved

- Email came down with spam and viruses.
- The Web came down with caches, proxies and Akamai.
- Structures generally got more complex.

Today and looking forward

- Applications are often very complex.
  - Composed of lots of servers and services.
    - Web 2.0, mashups, cloud computing, huge server farms and the likes.
- Their structure reflects the economic (and other) incentives of the designers.
  - We need to study the “architecture of control” as well as the “architecture of performance”.

Thesis:

- Availability depends on trustworthy components, not security mechanisms.
  - Ability to select among components depends on “vertical” interfaces.
- Future applications will offer a range of “operating modes” or “communication patterns”.
- A major determinant of which pattern or mode is used will be the trust among the communicating parties and the other parties relevant to the situation (e.g. the ISPs, etc.)
  - Trust requires a baseline of identity.
  - Trust and identity will be foundational for tomorrow’s applications.
    - Lots of ways to get these functions wrong.

Trust

- When there is application-level trust among components.
  - Remove constraints and protection.
    - E.g. use end-to-end encryption.
  - More efficient and flexible.
- When there is lack of trust.
  - Must impose constraints to compensate.
    - Email virus checkers, strip macros, etc.

Two quotes from the E2E paper

"In a system that includes communications, one usually draws a modular boundary around the communication subsystem and defines a firm interface between it and the rest of the system."

"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."

Why did we say this?

- The network was and is flakey.
- The end-node was a trustworthy platform on which to depend.
  - Reliability checks can compensate for unreliable network.
- But today the end-node is perhaps the least trustworthy part of the system.
- Today the issue is both technical reliability and tussle (actors with adverse interests.)
  - Does this eliminate the E2E argument, or motivate us to dig deeper?

A more general form

A possible re-framing of E2E is “trust-to-trust”.

- Original: “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.”
- Revised: “The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at points where it can be trusted to perform its job properly.”

Begs the question “trusted by whom”, and who has the power to dictate the answer.

Power, control and choice

- Application designers can design in the preferred range of patterns.
- Initiators of operations (e.g. the invoking end-points) can attempt to pick among these alternatives.
  - The potential for choice
- Network operators can control which patterns are allowed.
  - In the end, topology trumps, but in the end, a blunt instrument. The encryption escalation.

Conclusions for applications

- These patterns transcend specific application objectives. They apply to broad classes of applications.
- Most application designers will benefit from guidance as to how to think about and implement them.
  - What I have been calling design patterns.
- In many cases, the factor that will determine which pattern is preferred is assessment of which actors are most trustworthy.
  - So management of trust, and by extension identity, must be a first order capability, at least inside the application.
  - Trust assumptions will span applications.

Relate to network

- How does this relate to the network and the abstraction of its services?
  - I discussed the performance aspects of the service.
  - Consider the problem of finding a component.
- Many of the FIA proposals include an anycast feature.
  - Anycast to a service or a unit of information.
  - But should the network be picking the preferred component?
    - Must be in an equivalence class with respect to trust, and who defines that class?
      - NDN takes the position that “the network” must be able to detect a forgery.
  - How bypass a failed component using anycast addresses?

Application design

- The function itself:
  - The “purpose” of the application
- Performance
  - Classic parameters
- Correct operation
  - Perhaps in the face of attack
- Availability
  - Dealing with failures as well as attacks.
- Power and balance of control
  - Economics

A new modularity goal--isolation

- Can we isolate those design objectives.
  - Sorry—the answer seems to be *no*.
- When the application picks a version of a component, it must simultaneously take into account:
  - Performance—find one that is close and not congested.
  - Trust—find one that does what I want.
  - Availability—find one that is working.
- Perhaps an ordering can isolate them?

Conclusions

- The variability of the network service is fundamental and intrinsic.
  - Over-specificity either limits the range of applicability or leads to silly behavior.
- Many important functions are cross-layer.
- Applications must build availability out of diverse components.
  - Trust and trustworthy components are the key to availability.
- The details of packet transfer are a small part of the story.