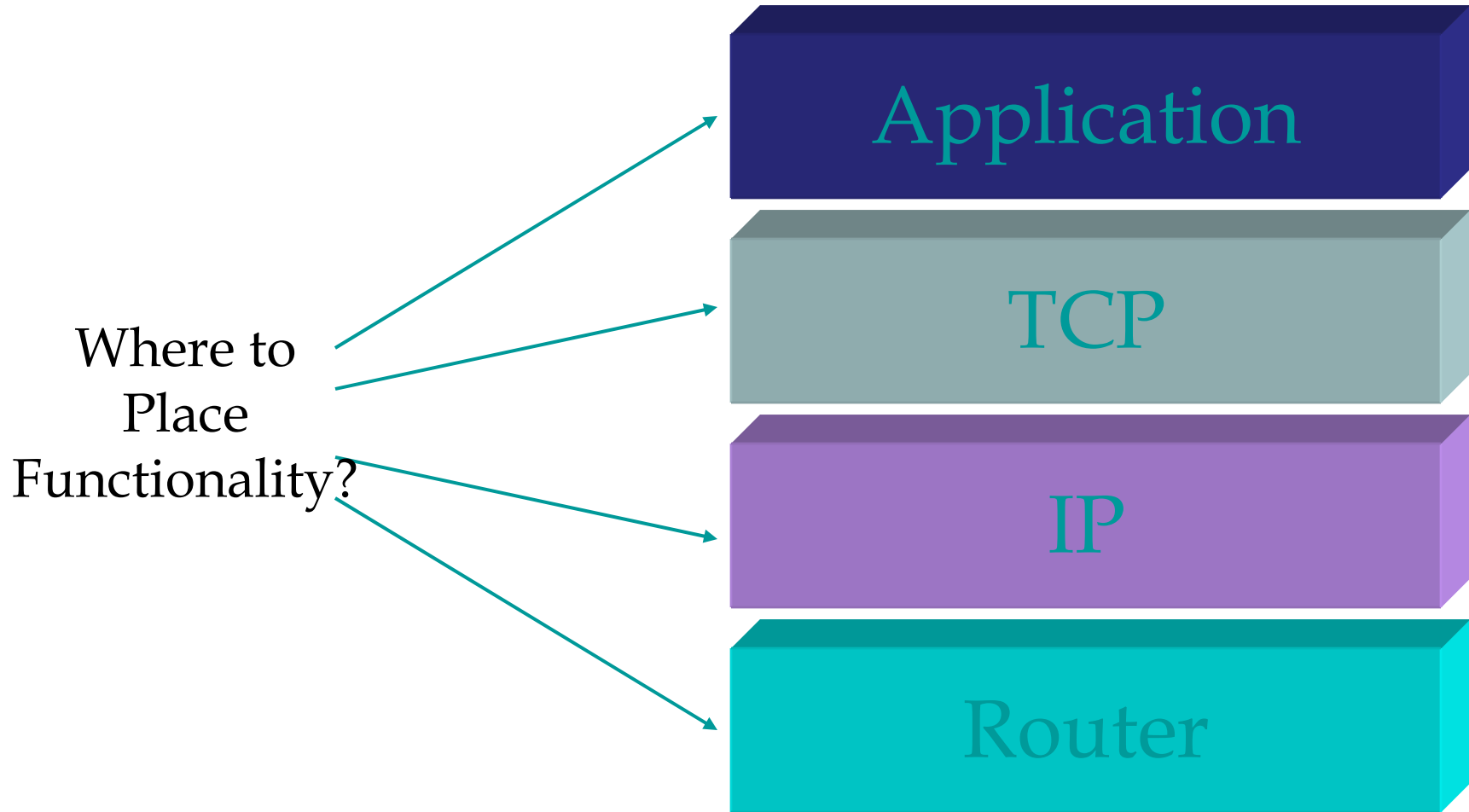


End-to-End Argument

Jeff Chase
Duke University



End-To-End Argument



End-to-End Argument

- Functionality should be implemented at a lower layer “if and only if it can be correctly and completely implemented there”.
 - Avoid at lower level if redundant with higher level
 - Performance optimizations are not a violation
- Early example
 - ARPANet provided reliable link transfers between switches
 - Packets could still get corrupted on host-to-switch link, or inside switches
 - Want to know if host *acted* on the request not whether it *received* it

Saltzer/Kaashoek View

- “The application knows best.”
- “Don’t bury it in a lower layer; let the endpoints deal with it because they know best what they need.”

Questions

1. How does TCP rate control reflect "end-to-end" principles?
2. What is the key drawback of end-to-end rate control?
3. What about SSL/TLS, relative to say, WEP?
4. Network Address Translation? Firewalls?
5. Should SSL be in the kernel or a library?

Example: Reliable File Transfer

- From disk on file (web) server over network to client
 - Disk can introduce bit errors
 - Host I/O buses can introduce bit errors
 - Packets can get garbled, dropped, misordered at any node
- Solution: integrity check on file, not per packet or per hop

Errors

- What should a lower-level component do if it detects a local failure?
 - Mask/recover locally?
 - Propagate to the higher level and let it figure out how to handle it?

Hop by Hop as Performance Optimization

- For file transfer application, consider varying conditions:
 - Prob(corrupted/lost packet per link) = p
 - Prob(packet lost end to end), avg. 15 hops across Internet
 - $p = 0.0001\% \Rightarrow \text{Prob(loss)} = 0.0015\%$
 - $p = 1\% \Rightarrow \text{Prob(loss)} = 14\%$
- Chance of file corruption grows with size of file
 - Potentially retransmit entire file for one lost packet?

The Need for Application-Specific Semantics

- Example: move reliability into the network communication protocol (such as TCP)
 - Overheads to implementing reliable, in-order delivery in the network
 - Not all applications want to pay it (use UDP)
- Applications should be able to pick and choose the semantics they require from underlying system?

Implication of End to End Principle

- Internet assumption: minimal support from underlying network
 - Ensure Internet can run on anything (IP on top of anything)
- Implications
 - Almost everything done at end hosts
 - Requires intelligent end hosts
 - Overlay networks
- Telephone network has stupid endpoints
 - What happens when light switch runs TCP?
 - Should your light switch run TCP?

Examples

- What should be done at the end hosts, and what by the network?
 - Addressing/routing?
 - Reliable delivery?
 - Sequenced delivery?
 - Congestion control/resource allocation?
 - Real-time guarantees?
 - Security?
 - Multicast?

What's Changed?

1980's Internet	2000's Internet
Low bandwidth * delay	High bandwidth * delay
Low drop rates, < 1%	High drop rates, > 5%
Few, long-lived flows	Many short-lived flows
Every host a good citizen	TCP “accelerators”
Symmetric routes & universal reachability	Asymmetric routes & private peering
Hosts powerful & routers overwhelmed	Hosts = toasters & routers intelligent?
Limited understanding of packet switching	ATM and MPP network design experience