# Lecture 3
# Design Philosophy & Applications

David Andersen
School of Computer Science
Carnegie Mellon University

15-441 Networking, Spring 2008
http://www.cs.cmu.edu/~dga/15-441/S08/



## Lecture Overview

- Last time:
  - Protocol stacks and layering
  - OSI and TCP/IP models
  - Application requirements from transport protocols
- Internet Architecture
- Application examples.
  - ftp
  - http
- Application requirements.
  - "ilities"
  - Sharing



## Internet Architecture

- Background
  - "The Design Philosophy of the DARPA Internet Protocols" (David Clark, 1988).
- Fundamental goal: Effective network interconnection
- Goals, *in order of priority:*
  1. Continue despite loss of networks or gateways
  2. Support multiple types of communication service
  3. Accommodate a variety of networks
  4. Permit distributed management of Internet resources
  5. Cost effective
  6. Host attachment should be easy
  7. Resource accountability



## Priorities

- The effects of the order of items in that list are still felt today
  - E.g., resource accounting is a hard, current research topic
- Let's look at them in detail

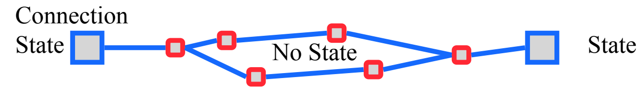
# Survivability

- **If network disrupted and reconfigured**
  - » Communicating entities should not care!
  - » No higher-level state reconfiguration
  - » Ergo, transport interface only knows "working" and "not working." Not working == complete partition.
- **How to achieve such reliability?**
  - » Where can communication state be stored?

| | Network | Host |
|---|---|---|
| Failure handing | Replication | "Fate sharing" |
| Net Engineering | Tough | Simple |
| Switches | Maintain state | Stateless |
| Host trust | Less | More |



# Fate Sharing




- **Lose state information for an entity if (and only if?) the entity itself is lost.**
- **Examples:**
  - » OK to lose TCP state if one endpoint crashes
    - – NOT okay to lose if an intermediate router reboots
  - » Is this still true in today's network?
    - – NATs and firewalls
- **Survivability compromise: Heterogenous network -> less information available to end hosts and Internet level recovery mechanisms**



# Types of Service

- **Recall from last time TCP vs. UDP**
  - » Elastic apps that need reliability: remote login or email
  - » Inelastic, loss-tolerant apps: real-time voice or video
  - » Others in between, or with stronger requirements
  - » Biggest cause of delay variation: reliable delivery
    - – Today's net: ~100ms RTT
    - – Reliable delivery can add *seconds*.
- **Original Internet model: "TCP/IP" one layer**
  - » First app was remote login…
  - » But then came debugging, voice, etc.
  - » These differences caused the layer split, added UDP
- **No QoS support assumed from below**
  - » In fact, some underlying nets only supported reliable delivery
    - – Made Internet datagram service less useful!
  - » Hard to implement without network support
  - » QoS is an ongoing debate…



# Varieties of Networks

- **Discussed a lot of this last time -**
  - » Interconnect the ARPANET, X.25 networks, LANs, satellite networks, packet networks, serial links…
- **Mininum set of assumptions for underlying net**
  - » Minimum packet size
  - » Reasonable delivery odds, but not 100%
  - » Some form of addressing unless point to point
- **Important non-assumptions:**
  - » Perfect reliability
  - » Broadcast, multicast
  - » Priority handling of traffic
  - » Internal knowledge of delays, speeds, failures, etc.
- **Much engineering then only has to be done once**

# The “Other” goals

- **Management**
  - » Today’s Internet is decentralized - BGP
  - » Very coarse tools. Still in the “assembly language” stage
- **Cost effectiveness**
  - » Economies of scale won out
  - » Internet cheaper than most dedicated networks
  - » Packet overhead less important by the year
- **Attaching a host**
  - » Not awful; DHCP and related autoconfiguration technologies helping. A ways to go, but the path is there
- **But…**

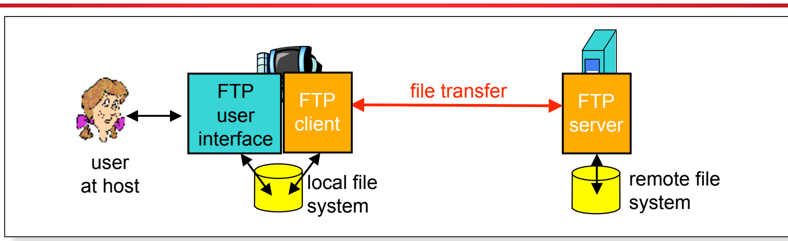


# Accountability

- **Huge problem.**
- **Accounting**
  - » Billing? (mostly flat-rate. But phones are moving that way too - people like it!)
  - » Inter-provider payments
    - – Hornet’s nest. Complicated. Political. Hard.
- **Accountability and security**
  - » Huge problem.
  - » Worms, viruses, etc.
    - – Partly a host problem. But hosts very trusted.
  - » Authentication
    - – Purely optional. Many philosophical issues of privacy vs. security.
- **… Questions before we move on to the project?**



# FTP: The File Transfer Protocol

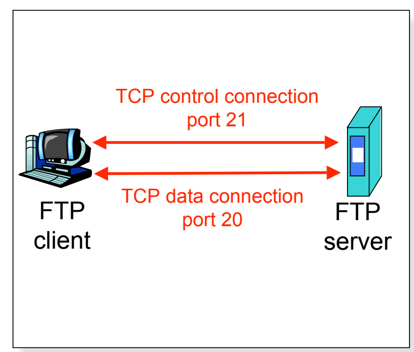



- **Transfer file to/from remote host**
- **Client/server model**
  - » *Client:* side that initiates transfer (either to/from remote)
  - » *Server:* remote host
- **ftp: RFC 959**
- **ftp server: port 21**



# Ftp: Separate Control, Data Connections

- **Ftp client contacts ftp server at port 21, specifying TCP as transport protocol**
- **Two parallel TCP connections opened:**
  - » **Control:** exchange commands, responses between client, server.
    **“out of band control”**
  - » **Data:** file data to/from server
- **Ftp server maintains “state”: current directory, earlier authentication**



# Ftp Commands, Responses

**Sample Commands:**

- **sent as ASCII text over control channel**
- `USER` *`username`*
- `PASS` *`password`*
- `LIST` **return list of files in current directory**
- `RETR filename` **retrieves (gets) file**
- `STOR filename` **stores (puts) file onto remote host**

**Sample Return Codes**

- **status code and phrase**
- `331 Username OK, password required`
- `125 data connection already open; transfer starting`
- `425 Can't open data connection`
- `452 Error writing file`



# HTTP Basics

- **HTTP layered over bidirectional byte stream**
  - » **Almost always TCP**
- **Interaction**
  - » **Client sends request to server, followed by response from server to client**
  - » **Requests/responses are encoded in text**
- **Stateless**
  - » **Server maintains no information about past client requests**

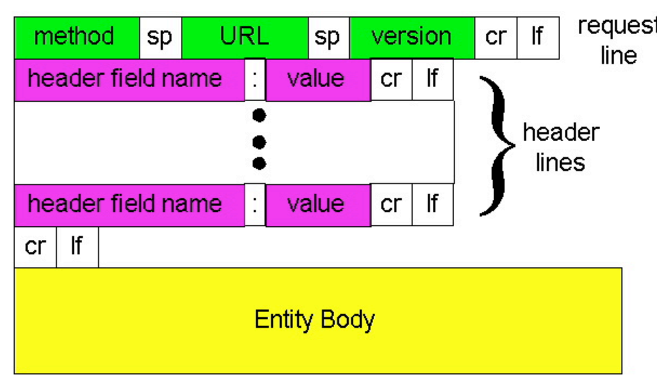


# How to Mark End of Message?

- **Size of message → Content-Length**
  - » **Must know size of transfer in advance**
- **Delimiter → MIME style Content-Type**
  - » **Server must "escape" delimiter in content**
- **Close connection**
  - » **Only server can do this**



# HTTP Request



# HTTP Request

- **Request line**
  - » Method
    - – GET – return URI
    - – HEAD – return headers only of GET response
    - – POST – send data to the server (forms, etc.)
  - » URI
    - – E.g. http://www.intel-iris.net/index.html with a proxy
    - – E.g. /index.html if no proxy
  - » HTTP version



# HTTP Request

- **Request headers**
  - » Authorization – authentication info
  - » Acceptable document types/encodings
  - » From – user email
  - » If-Modified-Since
  - » Referrer – what caused this page to be requested
  - » User-Agent – client software
- **Blank-line**
- **Body**



# HTTP Request Example

GET / HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: www.intel-iris.net
Connection: Keep-Alive



# HTTP Response

- **Status-line**
  - » HTTP version
  - » 3 digit response code
    - – 1XX – informational
    - – 2XX – success
      - 200 OK
    - – 3XX – redirection
      - 301 Moved Permanently
      - 303 Moved Temporarily
      - 304 Not Modified
    - – 4XX – client error
      - 404 Not Found
    - – 5XX – server error
      - 505 HTTP Version Not Supported
  - » Reason phrase

# HTTP Response

- Headers
  - Location – for redirection
  - Server – server software
  - WWW-Authenticate – request for authentication
  - Allow – list of methods supported (get, head, etc)
  - Content-Encoding – E.g x-gzip
  - Content-Length
  - Content-Type
  - Expires
  - Last-Modified
- Blank-line
- Body



# HTTP Response Example

```
HTTP/1.1 200 OK
Date: Tue, 27 Mar 2001 03:49:38 GMT
Server: Apache/1.3.14 (Unix)  (Red-Hat/Linux) mod_ssl/2.7.1
    OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24
Last-Modified: Mon, 29 Jan 2001 17:54:18 GMT
ETag: "7a11f-10ed-3a75ae4a"
Accept-Ranges: bytes
Content-Length: 4333
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
…..
```



# Cookies: Keeping "state"

**Many major Web sites use cookies**

**Four components:**

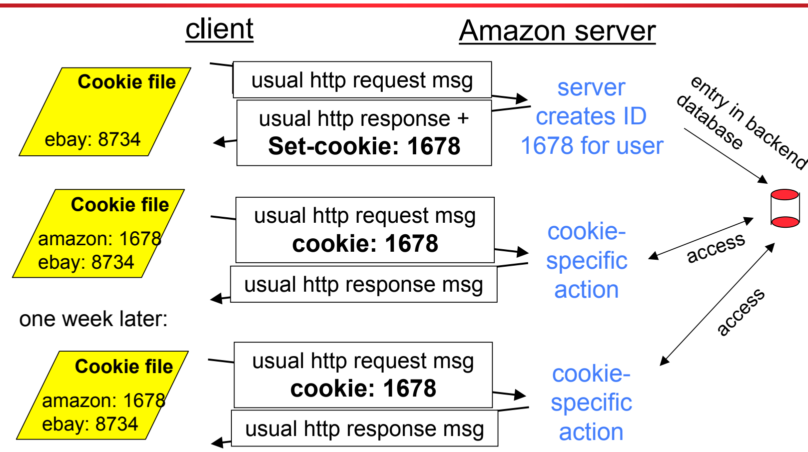
1) Cookie header line in the HTTP response message
2) Cookie header line in HTTP request message
3) Cookie file kept on user's host and managed by user's browser
4) Back-end database at Web site

**Example:**

- Susan accesses Internet always from same PC
- She visits a specific e-commerce site for the first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID



# Cookies: Keeping "State" (Cont.)



# Typical Workload (Web Pages)

- Multiple (typically small) objects per page
- File sizes
  - » Why different than request sizes?
  - » Also heavy-tailed
    - – Pareto distribution for tail
    - – Lognormal for body of distribution
- Embedded references
  - » Number of embedded objects = pareto – $p(x) = ak^a x^{-(a+1)}$

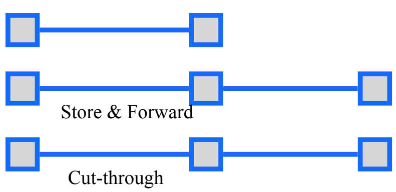


# HTTP 1.1 - new features

- Newer versions of HTTP add several new features (persistent connections, pipelined transfers) to speed things up.
- Let's detour into some performance evaluation and then look at those features



# Packet Delay




Prop + xmit

2*(Prop + xmit)

2*prop + xmit

When does cut-through matter?

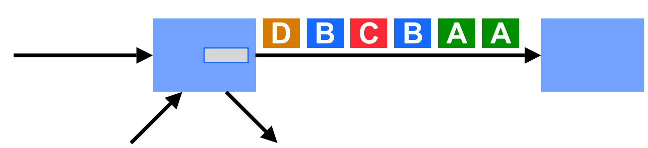
Next: Routers have finite speed (processing delay)

Routers may buffer packets (queueing delay)



# Packet Delay

- Sum of a number of different delay components.
- Propagation delay on each link.
  - » Proportional to the length of the link
- Transmission delay on each link.
  - » Proportional to the packet size and 1/link speed
- Processing delay on each router.
  - » Depends on the speed of the router
- Queuing delay on each router.
  - » Depends on the traffic load and queue size



# A Word about Units

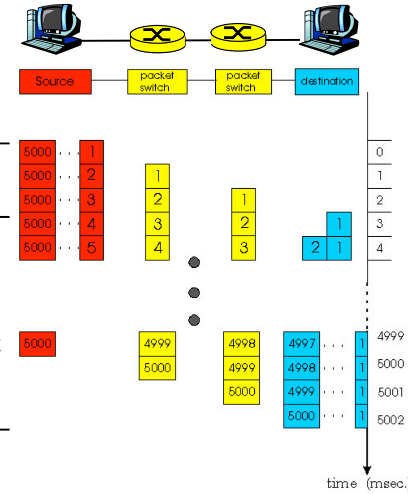
- What do “Kilo” and “Mega” mean?
  - » Depends on context
- Storage works in powers of two.
  - » 1 Byte = 8 bits
  - » 1 KByte = 1024 Bytes
  - » 1 MByte = 1024 Kbytes
- Networks work in decimal units.
  - » Network hardware sends bits, not Bytes
  - » 1 Kbps = 1000 bits per second
  - » To avoid confusion, use 1 Kbit/second
- Why? Historical: CS versus ECE.



# Application-level Delay

Delay of one packet

Average sustained throughput




$$\text{Delay}^* + \frac{\text{Size}}{\text{Throughput}}$$

Units: seconds + bits/(bits/seconds)

* For minimum sized packet



# Some Examples

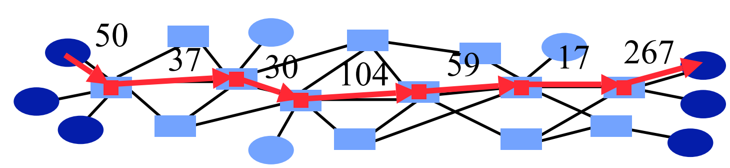
- How long does it take to send a 100 Kbit file?
  - » Assume a perfect world
  - » And a 10 Kbit file

| Throughput / Latency | 100 Kbit/s | 1 Mbit/s | 100 Mbit/s |
|---|---|---|---|
| 500 μsec | | | |
| 10 msec | | | |
| 100 msec | | | |



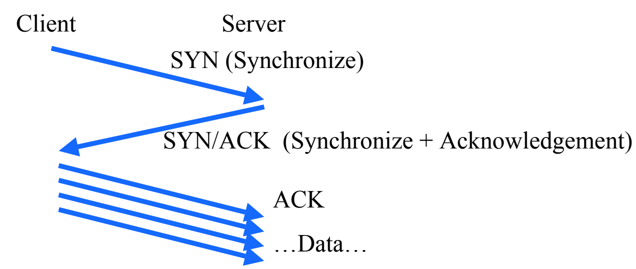
# Sustained Throughput

- When streaming packets, the network works like a pipeline.
  - » All links forward different packets in parallel
- Throughput is determined by the slowest stage.
  - » Called the bottleneck link
- Does not really matter why the link is slow.
  - » Low link bandwidth
  - » Many users sharing the link bandwidth



# One more detail: TCP

- TCP connections need to be set up
  - » "Three Way Handshake":




2: TCP transfers start slowly and then ramp up the bandwidth used (so they don't use too much)

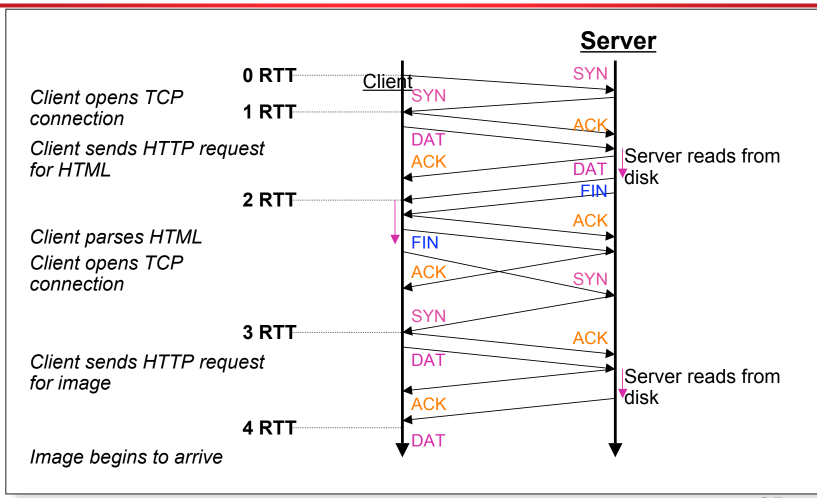


# HTTP 0.9/1.0

- One request/response per TCP connection
  - » Simple to implement
- Disadvantages
  - » Multiple connection setups → three-way handshake each time
    - – Several extra round trips added to transfer
  - » Multiple slow starts



# Single Transfer Example






# Performance Issues

- Short transfers are hard on TCP
  - » Stuck in slow start
  - » Loss recovery is poor when windows are small
- Lots of extra connections
  - » Increases server state/processing
- Servers also hang on to connection state after the connection is closed
  - » Why must server keep these?
  - » Tends to be an order of magnitude greater than # of active connections, why?

# Netscape Solution

- Mosaic (original popular Web browser) fetched one object at a time!
- Netscape uses multiple concurrent connections to improve response time
  - » Different parts of Web page arrive independently
  - » Can grab more of the network bandwidth than other users
- Doesn't necessarily improve response time
  - » TCP loss recovery ends up being timeout dominated because windows are small

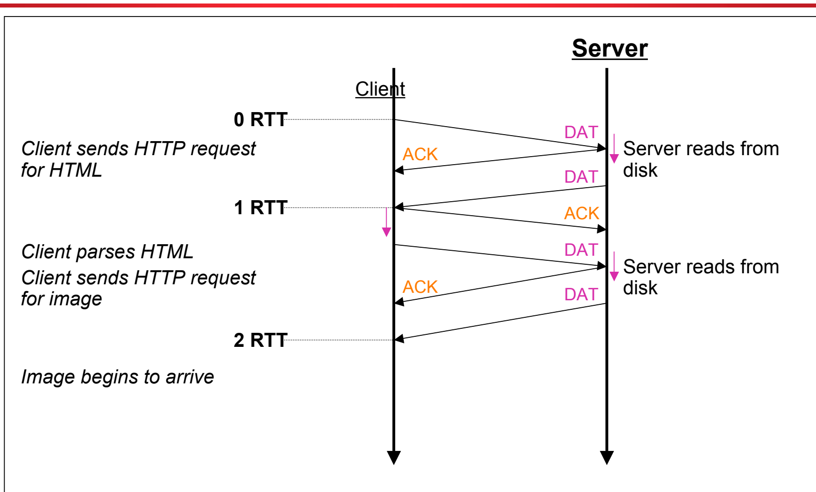


# Persistent Connection Solution

- Multiplex multiple transfers onto one TCP connection
- How to identify requests/responses
  - » Delimiter → Server must examine response for delimiter string
  - » Content-length and delimiter → Must know size of transfer in advance
  - » Block-based transmission → send in multiple length delimited blocks
  - » Store-and-forward → wait for entire response and then use content-length
  - » Solution → use existing methods and close connection otherwise



# Persistent Connection Solution






# Persistent HTTP

**Nonpersistent HTTP issues:**

- Requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- But browsers often open parallel TCP connections to fetch referenced objects

**Persistent HTTP**

- Server leaves connection open after sending response
- Subsequent HTTP messages between same client/server are sent over connection

**Persistent without pipelining:**

- Client issues new request only when previous response has been received
- One RTT for each referenced object

**Persistent with pipelining:**

- Default in HTTP/1.1
- Client sends requests as soon as it encounters a referenced object
- As little as one RTT for all the referenced objects

# Persistent Connection Performance

- **Benefits greatest for small objects**
  - Up to 2x improvement in response time
- **Server resource utilization reduced due to fewer connection establishments and fewer active connections**
- **TCP behavior improved**
  - Longer connections help adaptation to available bandwidth
  - Larger congestion window improves loss recovery



# Remaining Problems

- **Serialized transmission**
  - Much of the useful information in first few bytes
    - May be better to get the 1st 1/4 of all images than one complete image (e.g., progressive JPEG)
  - Can “packetize” transfer over TCP
    - Could use range requests
- **Application specific solution to transport protocol problems. :(**
  - Solve the problem at the transport layer
  - Could fix TCP so it works well with multiple simultaneous connections
    - More difficult to deploy



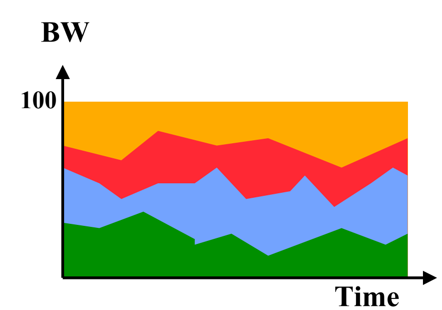
# Back to performance

- **We examined delay,**
- **But what about throughput?**
- **Important factors:**
  - Link capacity
  - *Other traffic*



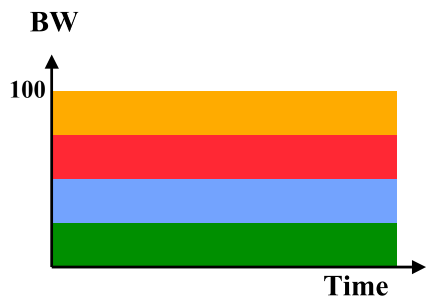
# Bandwidth Sharing

- **Bandwidth received on the bottleneck link determines end-to-end throughput.**
- **Router before the bottleneck link decides how much bandwidth each user gets.**
  - Users that try to send at a higher rate will see packet loss
- **User bandwidth can fluctuate quickly as flows are added or end, or as flows change their transmit rate.**



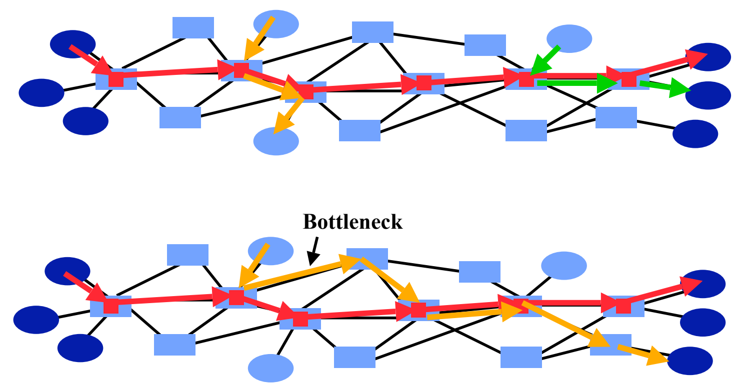
# Fair Sharing of Bandwidth

- All else being equal, fair means that users get equal treatment.
  - » Sounds fair
- When things are not equal, we need a policy that determines who gets how much bandwidth.
  - » Users who pay more get more bandwidth
  - » Users with a higher "rank" get more bandwidth
  - » Certain classes of applications get priority






# But It is Not that Simple






# Network Service Models

- Set of services that the network provides.
- Best effort service: network will do an honest effort to deliver the packets to the destination.
  - » Usually works
- "Guaranteed" services.
  - » Network offers (mathematical) performance guarantees
  - » Can apply to bandwidth, latency, packet loss, ..
- "Preferential" services.
  - » Network gives preferential treatment to some packets
  - » E.g. lower queuing delay
- Quality of Service is closely related to the question of fairness.



# Other Requirements

- Network reliability.
  - » Network service must always be available
- Security: privacy, DOS, ..
- Scalability.
  - » Scale to large numbers of users, traffic flows, ...
- Manageability: monitoring, control, ..
- Requirement often applies not only to the core network but also to the servers.
- Requirements imposed by users and network managers.

# Readings

- “End-to-end arguments in system design”, Saltzer, Reed, and Clark, ACM Transactions on Computer Systems, November 1984.
- “The design philosophy of the DARPA Internet Protocols”, Dave Clark, SIGCOMM 88.