Chapter 8
Security

A note on the use of these Powerpoint slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

▪ If you use these slides (e.g., in a class) that you mention their source (after all, we’d like people to use our book!)
▪ If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2016
© J.F Kurose and K.W. Ross, All Rights Reserved
Chapter 8 roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity
8.4 Securing e-mail
8.5 Securing TCP connections: SSL
8.6 Network layer security: IPsec
8.7 Securing wireless LANs
8.8 Operational security: firewalls and IDS
Firewalls

**firewall**

isolates organization’s internal net from larger Internet, allowing some packets to pass, blocking others

administered network

trusted “good guys”

firewall

public Internet

untrusted “bad guys”
Firewalls: why

prevent denial of service attacks:
  - SYN flooding: attacker establishes many bogus TCP connections, no resources left for “real” connections

prevent illegal modification/access of internal data
  - e.g., attacker replaces CIA’s homepage with something else

allow only authorized access to inside network
  - set of authenticated users/hosts

three types of firewalls:
  - stateless packet filters
  - stateful packet filters
  - application gateways
Stateless packet filtering

- internal network connected to Internet via router firewall
- router filters packet-by-packet, decision to forward/drop packet based on:
  - source IP address, destination IP address
  - TCP/UDP source and destination port numbers
  - ICMP message type
  - TCP SYN and ACK bits

Should arriving packet be allowed in? Departing packet let out?
Stateless packet filtering: example

- **example 1**: block incoming and outgoing datagrams with IP protocol field = 17 and with either source or destination port = 23
  - *result*: all incoming, outgoing UDP flows and telnet connections are blocked

- **example 2**: block inbound TCP segments with ACK=0.
  - *result*: prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside.
Stateless packet filtering: more examples

<table>
<thead>
<tr>
<th>Policy</th>
<th>Firewall Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>No outside Web access.</td>
<td>Drop all outgoing packets to any IP address, port 80</td>
</tr>
<tr>
<td>No incoming TCP connections, except those for institution’s public Web server only.</td>
<td>Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80</td>
</tr>
<tr>
<td>Prevent Web-radios from eating up the available bandwidth.</td>
<td>Drop all incoming UDP packets - except DNS and router broadcasts.</td>
</tr>
<tr>
<td>Prevent your network from being used for a smurf DoS attack.</td>
<td>Drop all ICMP packets going to a “broadcast” address (e.g. 130.207.255.255).</td>
</tr>
<tr>
<td>Prevent your network from being tracerouted</td>
<td>Drop all outgoing ICMP TTL expired traffic</td>
</tr>
</tbody>
</table>
**Access Control Lists**

**ACL:** table of rules, applied top to bottom to incoming packets: (action, condition) pairs: looks like OpenFlow forwarding (Ch. 4)!

<table>
<thead>
<tr>
<th>action</th>
<th>source address</th>
<th>dest address</th>
<th>protocol</th>
<th>source port</th>
<th>dest port</th>
<th>flag bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>222.22/16</td>
<td>outside of 222.22/16</td>
<td>TCP</td>
<td>&gt; 1023</td>
<td>80</td>
<td>any</td>
</tr>
<tr>
<td>allow</td>
<td>outside of 222.22/16</td>
<td>222.22/16</td>
<td>TCP</td>
<td>80</td>
<td>&gt; 1023</td>
<td>ACK</td>
</tr>
<tr>
<td>allow</td>
<td>222.22/16</td>
<td>outside of 222.22/16</td>
<td>UDP</td>
<td>&gt; 1023</td>
<td>53</td>
<td>---</td>
</tr>
<tr>
<td>allow</td>
<td>outside of 222.22/16</td>
<td>222.22/16</td>
<td>UDP</td>
<td>53</td>
<td>&gt; 1023</td>
<td>----</td>
</tr>
<tr>
<td>deny</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
</tr>
</tbody>
</table>
Stateful packet filtering

- **stateless packet filter**: heavy handed tool
  - admits packets that “make no sense,” e.g., dest port = 80, ACK bit set, even though no TCP connection established:

<table>
<thead>
<tr>
<th>action</th>
<th>source address</th>
<th>dest address</th>
<th>protocol</th>
<th>source port</th>
<th>dest port</th>
<th>flag bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>outside of 222.22/16</td>
<td>222.22/16</td>
<td>TCP</td>
<td>80</td>
<td>&gt; 1023</td>
<td>ACK</td>
</tr>
</tbody>
</table>

- **stateful packet filter**: track status of every TCP connection
  - track connection setup (SYN), teardown (FIN): determine whether incoming, outgoing packets “makes sense”
  - timeout inactive connections at firewall: no longer admit packets
## Stateful packet filtering

ACL augmented to indicate need to check connection state table before admitting packet

<table>
<thead>
<tr>
<th>action</th>
<th>source address</th>
<th>dest address</th>
<th>proto</th>
<th>source port</th>
<th>dest port</th>
<th>flag bit</th>
<th>check conxion</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>222.22/16</td>
<td>outside of 222.22/16</td>
<td>TCP</td>
<td>&gt; 1023</td>
<td>80</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>allow</td>
<td>outside of 222.22/16</td>
<td>222.22/16</td>
<td>TCP</td>
<td>80</td>
<td>&gt; 1023</td>
<td>ACK</td>
<td>X</td>
</tr>
<tr>
<td>allow</td>
<td>222.22/16</td>
<td>outside of 222.22/16</td>
<td>UDP</td>
<td>&gt; 1023</td>
<td>53</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>allow</td>
<td>outside of 222.22/16</td>
<td>222.22/16</td>
<td>UDP</td>
<td>53</td>
<td>&gt; 1023</td>
<td>----</td>
<td>X</td>
</tr>
<tr>
<td>deny</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
</tr>
</tbody>
</table>
**Application gateways**

- filter packets on application data as well as on IP/TCP/UDP fields.
- *example*: allow select internal users to telnet outside

1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host. Gateway relays data between 2 connections
3. router filter blocks all telnet connections not originating from gateway.
Limitations of firewalls, gateways

- **IP spoofing**: router can’t know if data “really” comes from claimed source
- if multiple app’s. need special treatment, each has own app. gateway
- client software must know how to contact gateway.
  - e.g., must set IP address of proxy in Web browser
- filters often use all or nothing policy for UDP
- **tradeoff**: degree of communication with outside world, level of security
- many highly protected sites still suffer from attacks
Intrusion detection systems

- packet filtering:
  - operates on TCP/IP headers only
  - no correlation check among sessions

- **IDS: intrusion detection system**
  - *deep packet inspection:* look at packet contents (e.g., check character strings in packet against database of known virus, attack strings)
  - examine correlation among multiple packets
    - port scanning
    - network mapping
    - DoS attack
Intrusion detection systems

multiple IDSs: different types of checking at different locations
Network Security (summary)

basic techniques......

• cryptography (symmetric and public)
• message integrity
• end-point authentication

.... used in many different security scenarios

• secure email
• secure transport (SSL)
• IP sec
• 802.11

operational security: firewalls and IDS
A note on the use of these Powerpoint slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we’d like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2016
J.F Kurose and K.W. Ross, All Rights Reserved
Chapter 1: introduction

our goal:
- get “feel” and terminology
- more depth, detail later in course
- approach:
  - use Internet as example

overview:
- what’s the Internet?
- what’s a protocol?
- network edge; hosts, access net, physical media
- network core: packet/circuit switching, Internet structure
- performance: loss, delay, throughput
- security
- protocol layers, service models
- history
What’s the Internet: “nuts and bolts” view

- **Internet**: “network of networks”
  - Interconnected ISPs
- **Protocols** control sending, receiving of messages
  - e.g., TCP, IP, HTTP, Skype, 802.11
- **Internet standards**
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force
What’s the Internet: a service view

- **infrastructure that provides services to applications:**
  - Web, VoIP, email, games, e-commerce, social nets, …

- **provides programming interface to apps**
  - hooks that allow sending and receiving app programs to “connect” to Internet
  - provides service options, analogous to postal service
A closer look at network structure:

- **network edge:**
  - hosts: clients and servers
  - servers often in data centers

- **access networks, physical media:** wired, wireless communication links

- **network core:**
  - interconnected routers
  - network of networks
Host: sends packets of data

host sending function:

- takes application message
- breaks into smaller chunks, known as packets, of length $L$ bits
- transmits packet into access network at transmission rate $R$

- link transmission rate, aka link capacity, aka link bandwidth

$$\text{packet transmission delay} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

Introduction
Packet-switching: store-and-forward

- takes $L/R$ seconds to transmit (push out) $L$-bit packet into link at $R$ bps
- **store and forward**: entire packet must arrive at router before it can be transmitted on next link
- end-end delay = $2L/R$ (assuming zero propagation delay)

**one-hop numerical example:**
- $L = 7.5$ Mbits
- $R = 1.5$ Mbps
- one-hop transmission delay = 5 sec

more on delay shortly …
Packet Switching: queueing delay, loss

queueing and loss:
- if arrival rate (in bits) to link exceeds transmission rate of link for a period of time:
  - packets will queue, wait to be transmitted on link
  - packets can be dropped (lost) if memory (buffer) fills up
Packet switching versus circuit switching

packet switching allows more users to use network!

example:
- 1 Mb/s link
- each user:
  - 100 kb/s when “active”
  - active 10% of time

- **circuit-switching:**
  - 10 users

- **packet switching:**
  - with 35 users, probability > 10 active at same time is less than .0004 *

Q: how did we get value 0.0004?  
Q: what happens if > 35 users ?

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/
But if one global ISP is viable business, there will be competitors … which must be interconnected.
Internet structure: network of networks

... and content provider networks (e.g., Google, Microsoft, Akamai) may run their own network, to bring services, content close to end users.
Internet structure: network of networks

- at center: small # of well-connected large networks
  - “tier-1” commercial ISPs (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
  - content provider network (e.g., Google): private network that connects it data centers to Internet, often bypassing tier-1, regional ISPs
Four sources of packet delay

\[d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}\]

\(d_{\text{proc}}\): nodal processing
- check bit errors
- determine output link
- typically < msec

\(d_{\text{queue}}\): queueing delay
- time waiting at output link for transmission
- depends on congestion level of router

A

B

transmission

propagation

nodal processing

queueing
Four sources of packet delay

\[ d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}} \]

\( d_{\text{trans}} \): transmission delay:
- \( L \): packet length (bits)
- \( R \): link bandwidth (bps)
- \( d_{\text{trans}} = \frac{L}{R} \)

\( d_{\text{prop}} \): propagation delay:
- \( d \): length of physical link
- \( s \): propagation speed (~2x10^8 m/sec)
- \( d_{\text{prop}} = \frac{d}{s} \)

\( d_{\text{trans}} \) and \( d_{\text{prop}} \) very different

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/
* Check out the Java applet for an interactive animation on trans vs. prop delay
**Throughput**

- **Throughput**: rate (bits/time unit) at which bits transferred between sender/receiver
  - *instantaneous*: rate at given point in time
  - *average*: rate over longer period of time

[Diagram showing server sending bits into a pipe, with pipes that can carry fluid at rates $R_s$ and $R_c$ bits/sec.]
Internet protocol stack

- **application**: supporting network applications
  - FTP, SMTP, HTTP
- **transport**: process-process data transfer
  - TCP, UDP
- **network**: routing of datagrams from source to destination
  - IP, routing protocols
- **link**: data transfer between neighboring network elements
  - Ethernet, 802.11 (WiFi), PPP
- **physical**: bits “on the wire”
ISO/OSI reference model

- **presentation**: allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
- **session**: synchronization, checkpointing, recovery of data exchange
- Internet stack “missing” these layers!
  - these services, *if needed*, must be implemented in application
  - needed?
Encapsulation

message
segment
datagram
frame

source

application
transport
network
link
physical

destination

application
transport
network
link
physical

network
link
physical

link
physical

switch

router

Introduction
Chapter 2
Application Layer

A note on the use of these Powerpoint slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we’d like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2016
© J.F Kurose and K.W. Ross, All Rights Reserved
Chapter 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 electronic mail
   • SMTP, POP3, IMAP
2.4 DNS
2.5 P2P applications
2.6 video streaming and content distribution networks
2.7 socket programming with UDP and TCP
Chapter 2: application layer

our goals:
- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
  - content distribution networks
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- creating network applications
  - socket API
Some network apps

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)

- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- ...
- ...
Client-server architecture

server:
- always-on host
- permanent IP address
- data centers for scaling

clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other
P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management
Sockets

- process sends/receives messages to/from its **socket**
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
TCP service:
- reliable transport between sending and receiving process
- flow control: sender won’t overwhelm receiver
- congestion control: throttle sender when network overloaded
- does not provide: timing, minimum throughput guarantee, security
- connection-oriented: setup required between client and server processes

UDP service:
- unreliable data transfer between sending and receiving process
- does not provide: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,

Q: why bother? Why is there a UDP?
Securing TCP

TCP & UDP
- no encryption
- cleartext passwds sent into socket traverse Internet in cleartext

SSL
- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer
- apps use SSL libraries, that “talk” to TCP

SSL socket API
- cleartext passwords sent into socket traverse Internet encrypted
- see Chapter 8
HTTP overview

HTTP: hypertext transfer protocol

- Web’s application layer protocol
- Client/server model
  - client: browser that requests, receives, (using HTTP protocol) and “displays” Web objects
  - server: Web server sends (using HTTP protocol) objects in response to requests

PC running Firefox browser

HTTP request
HTTP response

server running Apache Web server

iPhone running Safari browser
HTTP overview (continued)

uses TCP:
- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is “stateless”
- server maintains no information about past client requests

aside
- protocols that maintain “state” are complex!
  - past history (state) must be maintained
  - if server/client crashes, their views of “state” may be inconsistent, must be reconciled
Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time:
- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time = $2\text{RTT} + \text{file transmission time}$
**Persistent HTTP**

*non-persistent HTTP issues:*
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- 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 sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects
**HTTP request message**

- Two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

```plaintext
GET /index.html HTTP/1.1
Host: www-net.cs.umass.edu
User-Agent: Firefox/3.6.10
Accept: text/html,application/xhtml+xml
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7
Keep-Alive: 115
Connection: keep-alive
```

*Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/*
HTTP response message

status line (protocol status code status phrase)

HTTP/1.1 200 OK
Date: Sun, 26 Sep 2010 20:09:20 GMT
Server: Apache/2.0.52 (CentOS)
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT
ETag: "17dc6-a5c-bf716880"
Accept-Ranges: bytes
Content-Length: 2652
Keep-Alive: timeout=10, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1

data data data data data ...

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/
Cookies: keeping “state” (cont.)

One week later:

Amazon server creates ID 1678 for user

cookie-specific action

create entry

backend database

usual http request msg

usual http response msg

set-cookie: 1678

usual http request msg

cookie: 1678

usual http response msg

cookie file

ebay 8734

amazon 1678

ebay 8734

amazon 1678

usual http request msg

cookie: 1678

usual http response msg

usual http request msg

cookie: 1678

usual http response msg

cookie file

ebay 8734

amazon 1678

ebay 8734

amazon 1678

usual http request msg

cookie: 1678

usual http response msg

usual http request msg

cookie: 1678

usual http response msg

usual http request msg

cookie: 1678

usual http response msg
Web caches (proxy server)

**goal:** satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client
Caching example:

**assumptions:**
- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

**consequences:**
- LAN utilization: 15%
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay = 2 sec + minutes + usecs

problem!
Electronic mail

Three major components:
- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server
Sample SMTP interaction

S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
- **SMTP**: delivery/storage to receiver’s server
- mail access protocol: retrieval from server
  - **POP**: Post Office Protocol [RFC 1939]: authorization, download
  - **IMAP**: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored messages on server
  - **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.
client wants IP for www.amazon.com; 1st approximation:

- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com
TLD, authoritative servers

*top-level domain (TLD) servers:*
  - responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
  - Network Solutions maintains servers for .com TLD
  - Educause for .edu TLD

*authoritative DNS servers:*
  - organization’s own DNS server(s), providing authoritative hostname to IP mappings for organization’s named hosts
  - can be maintained by organization or service provider
DNS name resolution example

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

**iterated query:**
- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”
DNS name resolution example

recursive query:
- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?
DNS records

**DNS:** distributed database storing resource records (RR)

RR format: \((\text{name}, \text{value}, \text{type}, \text{ttl})\)

- **type=A**
  - name is hostname
  - value is IP address

- **type=NS**
  - name is domain (e.g., foo.com)
  - value is hostname of authoritative name server for this domain

- **type=CNAME**
  - name is alias name for some “canonical” (the real) name
  - www.ibm.com is really servereast.backup2.ibm.com
  - value is canonical name

- **type=MX**
  - value is name of mailserver associated with name
Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

Examples:
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)
Client-server vs. P2P: example

client upload rate = $u$, $F/u = 1$ hour, $u_s = 10u$, $d_{min} \geq u_s$
P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

**tracker**: tracks peers participating in torrent

**torrent**: group of peers exchanging chunks of a file

Alice arrives ...
... obtains list of peers from tracker
... and begins exchanging file chunks with peers in torrent
Video Streaming and CDNs: context

- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users
- challenge: scale - how to reach ~1B users?
  - single mega-video server won’t work (why?)
- challenge: heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- solution: distributed, application-level infrastructure
Streaming multimedia: DASH

- **DASH**: Dynamic, Adaptive Streaming over HTTP

**server:**
- divides video file into multiple chunks
- each chunk stored, encoded at different rates
- **manifest file**: provides URLs for different chunks

**client:**
- periodically measures server-to-client bandwidth
- consulting manifest, requests one chunk at a time
  - chooses maximum coding rate sustainable given current bandwidth
  - can choose different coding rates at different points in time (depending on available bandwidth at time)